



z/Architecture

Principles of Operation



z/Architecture

Principles of Operation

Note:

Before using this information and the product it supports, be sure to read the general information under "Notices" on page xvii.

Softcopy Note:

The reader should be aware of the fact that this publication contains many symbols, such as superscripts, that may not display correctly with any given hardware or software. The definitive version of this publication is the hardcopy version.

Ninth Edition (August, 2010)

This edition obsoletes and replaces *z/Architecture Principles of Operation*, SA22-7832-07.

This publication is provided for use in conjunction with other relevant IBM publications, and IBM makes no warranty, express or implied, about its completeness or accuracy. The information in this publication is current as of its publication date but is subject to change without notice.

Additional copies of this and other IBM publications may be ordered or downloaded from the IBM publications web site at <http://www.ibm.com/support/documentation>.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

International Business Machines Corporation
Department 55JA
Mail Station P384
2455 South Road
Poughkeepsie, N.Y., 12601-5400
United States of America

FAX (United States and Canada): 1-845-432-9405
FAX (Other Countries): Your International Access Code + 1-845-432-9405
IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: <http://www.ibm.com/servers/eserver/zseries/zos/webqs.html>

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1990-2010. All rights reserved.

US Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	xxiii
Trademarks	xxiii

Preface	xxv
Size and Number Notation.	xxvi
Bytes, Characters, and Codes.	xxvii
Other Publications	xxvii
Summary of Changes in Ninth Edition.	xxvii
Summary of Changes in Eighth Edition.	xxix
Summary of Changes in Seventh Edition	xxx
Summary of Changes in Sixth Edition	xxxii
Summary of Changes in Fifth Edition	xxxiii
Summary of Changes in Fourth Edition.	xxxvi
Summary of Changes in Third Edition.	xxxvii
Summary of Changes in Second Edition.	xxxix

Chapter 1, Introduction	1-1
Highlights of Original z/Architecture	1-2
General Instructions for 64-Bit Integers.	1-2
Other New General Instructions	1-3
Floating-Point Instructions	1-4
Control Instructions	1-4
Trimodal Addressing	1-5
Modal Instructions	1-5
Effects on Bits 0-31 of a General Register .	1-5
Input/Output.	1-6
Additions to z/Architecture.	1-7
ASN-and-LX-Reuse Facility.	1-7
CMPSC-Enhancement Facility	1-7
Compare-and-Swap-and-Store Facility	1-7
Compare-and-Swap-and-Store Facility 2	1-7
Conditional-SSKE Facility	1-7
Configuration-Topology Facility	1-7
DAT-Enhancement Facility 1	1-8
DAT-Enhancement Facility 2	1-8
Decimal-Floating-Point Facility	1-8
Decimal-Floating-Point-Rounding Facility	1-8
Distinct-Operands Facility	1-8
Enhanced-DAT Facility	1-8
Enhanced-Monitor Facility	1-9
ETF2-Enhancement Facility	1-9
ETF3-Enhancement Facility	1-9
Execute-Extensions Facility	1-10
Extended-Immediate Facility	1-10
Extended-I/O-Measurement-Block Facility ..	1-10
Extended-I/O-Measurement-Word Facility ..	1-10
Extended-Translation Facility 2	1-10
Extended-Translation Facility 3	1-11
Extract-CPU-Time Facility	1-11

Fast-BCR-Serialization Facility	1-11
Fibre-Channel Extensions (FCX)	1-11
Floating-Point Extension Facility.	1-12
Floating-Point-Support-Sign- Handling Facility	1-12
FPR-GR-Transfer Facility	1-13
General-Instructions-Extension Facility	1-13
HFP Multiply-and-Add/Subtract Facility	1-13
HFP-Unnormalized-Extensions Facility	1-13
High-Word Facility.	1-13
IEEE-Exception-Simulation Facility.	1-14
Interlocked-Access Facility	1-14
IPTE-Range Facility	1-14
List-Directed Initial Program Load.	1-14
Load/Store-on-Condition Facility.	1-14
Long-Displacement Facility	1-14
Message-Security Assist.	1-15
Message-Security-Assist Extension 1.	1-15
Message-Security-Assist Extension 2.	1-15
Message-Security-Assist Extension 3.	1-15
Message-Security-Assist Extension 4.	1-16
Modified CCW Indirect Data Addressing Facility.	1-17
Move-With-Optional-Specifications Facility. .	1-18
Multiple-Subchannel-Set Facility.	1-18
Nonquiescing Key-Setting Facility	1-18
Parsing-Enhancement Facility	1-18
PER-3 Facility:	1-18
PFPO Facility	1-18
Population-Count Facility	1-18
The Reset-Reference-Bits-Multiple Facility. .	1-19
Restore-Subchannel Facility	1-19
Server-Time-Protocol Facility:	1-19
Store-Clock-Fast Facility.	1-19
Store-Facility-List-Extended Facility:	1-19
TOD-Clock-Steering Facility	1-19
The ESA/390 Base	1-19
The ESA/370 and 370-XA Base	1-25
System Program	1-26
Compatibility	1-26
Compatibility among z/Architecture Systems .	1-26
Compatibility between z/Architecture and ESA/390	1-27
Control-Program Compatibility	1-27
Problem-State Compatibility	1-27
Availability	1-27

Chapter 2, Organization	2-1
Main Storage	2-2
Expanded Storage	2-2
CPU	2-2
PSW	2-3
General Registers	2-3
Floating-Point Registers	2-3

Floating-Point-Control Register	2-3
Control Registers	2-5
Access Registers	2-5
Cryptographic Facility	2-5
External Time Reference	2-5
I/O	2-5
Channel Subsystem	2-5
Channel Paths	2-6
I/O Devices and Control Units	2-6
Operator Facilities	2-6

Chapter 3, Storage 3-1

Storage Addressing	3-2
Information Formats	3-2
Integral Boundaries	3-3
Address Types and Formats	3-4
Address Types	3-4
Absolute Address	3-4
Real Address	3-4
Virtual Address	3-5
Primary Virtual Address	3-5
Secondary Virtual Address	3-5
AR-Specified Virtual Address	3-5
Home Virtual Address	3-5
Logical Address	3-5
Instruction Address	3-5
Effective Address	3-5
Address Size and Wraparound	3-5
Address Wraparound	3-6
Storage Key	3-8
Protection	3-10
Key-Controlled Protection	3-10
Storage-Protection-Override Control	3-11
Fetch-Protection-Override Control	3-12
Access-List-Controlled Protection	3-12
DAT Protection	3-12
Low-Address Protection	3-13
Suppression on Protection	3-14
Enhanced Suppression on Protection	3-15
Reference Recording	3-16
Change Recording	3-16
Change-Recording Override	3-17
Prefixing	3-18
Address Spaces	3-18
Changing to Different Address Spaces	3-19
Address-Space Number	3-20
ASN-Second-Table-Entry Sequence Number	3-20
ASN-Second-Table-Entry Instance Number	3-21
and ASN Reuse	3-21
ASN Translation	3-26
ASN-Translation Controls	3-27
Control Register 14	3-27
ASN-Translation Tables	3-27

ASN-First-Table Entries	3-27
ASN-Second-Table Entries	3-27
ASN-Translation Process	3-29
ASN-First-Table Lookup	3-30
ASN-Second-Table Lookup	3-31
Recognition of Exceptions during ASN	3-31
Translation	3-31
ASN Authorization	3-31
ASN-Authorization Controls	3-31
Control Register 4	3-31
ASN-Second-Table Entry	3-32
Authority-Table Entries	3-32
ASN-Authorization Process	3-32
Authority-Table Lookup	3-32
Recognition of Exceptions during ASN	3-34
Authorization	3-34
Dynamic Address Translation	3-34
Translation Control	3-36
Translation Modes	3-36
Control Register 0	3-37
Control Register 1	3-38
Control Register 7	3-39
Control Register 13	3-40
Translation Tables	3-40
Region-Table Entries	3-41
Segment-Table Entries	3-42
Page-Table Entries	3-44
Translation Process	3-45
Inspection of Real-Space Control	3-46
Inspection of Designation-Type Control	3-46
Lookup in a Table Designated by an	3-48
Address-Space-Control Element	3-48
Lookup in a Table Designated by a Region-	3-50
Table Entry	3-50
Page-Table Lookup	3-51
Formation of the Real and Absolute	3-52
Addresses	3-52
Recognition of Exceptions during	3-52
Translation	3-52
Translation-Lookaside Buffer	3-52
TLB Structure	3-53
Formation of TLB Entries	3-53
Use of TLB Entries	3-54
Modification of Translation Tables	3-55
Address Summary	3-58
Addresses Translated	3-58
Handling of Addresses	3-59
Assigned Storage Locations	3-61

Chapter 4, Control 4-1

Stopped, Operating, Load, and Check-Stop	4-2
States	4-2
Stopped State	4-2

Operating State	4-2	CCW-Type IPL	4-56
Load State	4-3	Store Status	4-56
Check-Stop State.	4-3	Multiprocessing	4-57
Program-Status Word	4-3	Shared Main Storage	4-57
Program-Status-Word Format	4-5	CPU-Address Identification.	4-57
Control Registers.	4-7	CPU Signaling and Response	4-58
Tracing.	4-10	Signal-Processor Orders.	4-58
Control-Register Allocation	4-12	Sense	4-58
Trace Entries	4-12	External Call	4-58
Operation.	4-22	Emergency Signal.	4-58
Program-Event Recording.	4-23	Start.	4-58
PER Instruction-Fetching Nullification	4-23	Stop.	4-58
Control-Register Allocation and Address-		Restart.	4-59
Space-Control Element	4-23	Stop and Store Status.	4-59
Operation	4-25	Initial CPU Reset.	4-59
Identification of Cause	4-26	CPU Reset	4-59
Priority of Indication	4-29	Set Prefix.	4-59
Storage-Area Designation	4-30	Store Status at Address	4-59
PER Events	4-31	Set Architecture.	4-60
Successful Branching	4-31	Conditional Emergency Signal	4-61
Instruction Fetching	4-31	Sense Running Status:	4-62
Storage Alteration	4-32	Conditions Determining Response	4-62
Store Using Real Address	4-32	Conditions Precluding Interpretation of the	
Indication of PER Events Concurrently with		Order Code	4-62
Other Interruption Conditions	4-33	Status Bits.	4-63
Breaking-Event-Address Recording	4-36	Facility Indications.	4-66
Breaking-Event-Address Register	4-37		
Execution-Break Instructions	4-37		
Timing	4-38	Chapter 5, Program Execution 5-1	
Time-of-Day Clock.	4-38	Instructions	5-2
Format	4-38	Operands	5-3
States.	4-38	Instruction Formats	5-3
Changes in Clock State	4-39	Register Operands	5-6
Setting and Inspecting the Clock	4-40	Immediate Operands.	5-6
TOD Programmable Register	4-41	Storage Operands.	5-7
TOD-Clock Synchronization	4-43	Address Generation	5-8
Timing Mode	4-43	Trimodal Addressing.	5-8
Timing State.	4-44	Sequential Instruction-Address Generation	5-8
STP Clock Source State	4-44	Operand-Address Generation.	5-8
TOD-Clock Steering.	4-44	Formation of the Intermediate Value.	5-8
TOD-Clock Steering Overview.	4-45	Formation of the Operand Address.	5-9
TOD-Offset-Update Events	4-47	Branch-Address Generation	5-10
Episodes	4-47	Formation of the Intermediate Value.	5-10
TOD-Clock-Steering Registers	4-47	Formation of the Branch Address	5-10
Clock Comparator	4-48	Instruction Execution and Sequencing	5-11
CPU Timer.	4-49	Decision Making	5-11
Externally Initiated Functions.	4-50	Loop Control	5-11
Resets	4-50	Subroutine Linkage without the Linkage Stack.	5-11
CPU Reset.	4-53	Simple Branch Instructions	5-11
Initial CPU Reset	4-54	Other Linkage Instructions	5-14
Subsystem Reset.	4-54	Interruptions	5-19
Clear Reset	4-54	Types of Instruction Ending	5-19
Power-On Reset	4-55	Completion	5-19
Initial Program Loading	4-56	Suppression	5-19

Nullification	5-20	Control Register 2	5-50
Termination	5-20	Control Register 5	5-50
Interruptible Instructions	5-20	Control Register 8	5-51
Point of Interruption	5-20	Access Registers	5-51
Unit of Operation	5-20	Access-Register-Translation Tables	5-52
Execution of Interruptible Instructions	5-21	Dispatchable-Unit Control Table and	
Condition-Code Alternative to		Access-List Designations	5-52
Interruptibility	5-22	Access-List Entries	5-53
Exceptions to Nullification and Suppression	5-23	ASN-Second-Table Entries	5-54
Storage Change and Restoration for DAT-		Access-Register-Translation Process	5-55
Associated Access Exceptions	5-23	Selecting the Access-List-Entry Token	5-58
Modification of DAT-Table Entries	5-24	Obtaining the Primary or Secondary	
Trial Execution for Editing Instructions and		Address-Space-Control Element	5-58
Translate Instruction	5-24	Checking the First Byte of the ALET	5-58
Authorization Mechanisms	5-24	Obtaining the Effective Access-List	
Mode Requirements	5-24	Designation	5-58
Extraction-Authority Control	5-25	Access-List Lookup	5-58
PSW-Key Mask	5-25	Locating the ASN-Second-Table Entry	5-59
Secondary-Space Control	5-25	Authorizing the Use of the Access-List	
Subsystem-Linkage Control	5-26	Entry	5-59
ASN-Translation Control	5-26	Checking for Access-List-Controlled	
Authorization Index	5-26	Protection	5-60
Instructions and Controls Related to ASN-		Obtaining the Address-Space-Control Element	
and-LX Reuse	5-27	from the ASN-Second-Table Entry	5-60
PC-Number Translation	5-30	Recognition of Exceptions during Access-	
PC-Number Translation Control	5-31	Register Translation	5-60
Control Register 0	5-31	ART-Lookaside Buffer	5-60
Control Register 5	5-32	ALB Structure	5-60
PC-Number Translation Tables	5-32	Formation of ALB Entries	5-60
Linkage-Table Entries	5-33	Use of ALB Entries	5-61
Linkage-First-Table Entries	5-33	Modification of ART Tables	5-61
Linkage-Second-Table Entries	5-33	Subspace Groups	5-62
Entry-Table Entries	5-34	Subspace-Group Tables	5-62
Table Summary	5-35	Subspace-Group Dispatchable-Unit Control	
PC-Number-Translation Process	5-35	Table	5-62
Obtaining the Linkage-Table or Linkage-		Subspace-Group ASN-Second-Table	
First-Table Designation	5-39	Entries	5-64
Linkage-Table Lookup	5-39	Subspace-Replacement Operations	5-65
Linkage-First-Table Lookup	5-39	Linkage-Stack Introduction	5-66
Linkage-Second-Table Lookup	5-40	Summary	5-66
Linkage-Second-Table-Entry-Sequence-		Linkage-Stack Functions	5-67
Number Comparison	5-40	Transferring Program Control	5-67
Entry-Table Lookup	5-40	Branching Using the Linkage Stack	5-69
Recognition of Exceptions during PC-		Adding and Retrieving Information	5-70
Number Translation	5-41	Testing Authorization	5-70
Home Address Space	5-41	Program-Problem Analysis	5-71
Access-Register Introduction	5-42	Linkage-Stack Entry-Table Entries	5-71
Summary	5-42	Linkage-Stack Operations	5-72
Access-Register Functions	5-43	Linkage-Stack-Operations Control	5-74
Access-Register-Specified Address		Control Register 0	5-74
Spaces	5-43	Control Register 15	5-74
Access-Register Instructions	5-49	Linkage Stack	5-74
Access-Register Translation	5-50	Entry Descriptors	5-75
Access-Register-Translation Control	5-50	Header Entries	5-76

Trailer Entries	5-76	Zero ILC	6-6
State Entries	5-77	ILC on Instruction-Fetching Exceptions	6-7
Stacking Process	5-80	Exceptions Associated with the PSW	6-8
Locating Space for a New Entry	5-80	Early Exception Recognition	6-8
Forming the New Entry	5-81	Late Exception Recognition	6-9
Updating the Current Entry	5-82	External Interruption	6-9
Updating Control Register 15	5-82	Clock Comparator	6-10
Recognition of Exceptions during the		CPU Timer	6-11
Stacking Process	5-82	Emergency Signal	6-11
Unstacking Process	5-82	External Call	6-11
Locating the Current Entry and Processing		Interrupt Key	6-11
a Header Entry	5-83	Malfunction Alert	6-12
Checking for a State Entry	5-84	Service Signal	6-12
Restoring Information	5-84	Timing Alert	6-12
Updating the Preceding Entry	5-84	ETR-Timing-Alert Condition	6-12
Updating Control Register 15	5-84	STP-Timing-Alert Condition	6-12
Recognition of Exceptions during the		I/O Interruption	6-13
Unstacking Process	5-85	Machine-Check Interruption	6-14
Sequence of Storage References	5-85	Program Interruption	6-14
Conceptual Sequence	5-85	Data-Exception Code (DXC)	6-14
Overlapped Operation of Instruction		Priority of Program Interruptions for Data	
Execution	5-85	Exceptions	6-15
Divisible Instruction Execution	5-86	Program-Interruption Conditions	6-17
Interlocks for Virtual-Storage References	5-86	Addressing Exception	6-17
Interlocks between Instructions	5-87	AFX-Translation Exception	6-18
Interlocks within a Single Instruction	5-87	ALEN-Translation Exception	6-18
Instruction Fetching	5-89	ALE-Sequence Exception	6-19
ART-Table and DAT-Table Fetches	5-90	ALET-Specification Exception	6-19
Storage-Key Accesses	5-91	ASCE-Type Exception	6-19
Storage-Operand References	5-93	ASTE-Instance Exception	6-20
Storage-Operand Fetch References	5-93	ASTE-Sequence Exception	6-20
Storage-Operand Store References	5-94	ASTE-Validity Exception	6-20
Storage-Operand Update References	5-94	ASX-Translation Exception	6-21
Storage-Operand Consistency	5-95	Crypto-Operation Exception	6-21
Single-Access References	5-95	Data Exception	6-21
Multiple-Access References	5-96	Decimal-Divide Exception	6-22
Block-Concurrent References	5-97	Decimal-Overflow Exception	6-22
Relation between Operand Accesses	5-98	Execute Exception	6-22
Other Storage References	5-99	EX-Translation Exception	6-23
Relation between Storage-Key Accesses	5-99	Extended-Authority Exception	6-23
Serialization	5-99	Fixed-Point-Divide Exception	6-23
CPU Serialization	5-100	Fixed-Point-Overflow Exception	6-24
Specific-Operand Serialization	5-101	HFP-Divide Exception	6-24
Channel-Program Serialization	5-102	HFP-Exponent-Overflow Exception	6-24
Quiescing	5-102	HFP-Exponent-Underflow Exception	6-24
Monitor-Event Counting	5-102	HFP-Significance Exception	6-24
		HFP-Square-Root Exception	6-24
		LFX-Translation Exception	6-25
Chapter 6, Interruptions	6-1	LSTE-Sequence Exception	6-25
Interruption Action	6-2	LSX-Translation Exception	6-25
Interruption Code	6-4	LX-Translation Exception	6-25
Enabling and Disabling	6-5	Monitor Event	6-26
Handling of Floating Interruption Conditions	6-6	Operand Exception	6-27
Instruction-Length Code	6-6	Operation Exception	6-27

Page-Translation Exception	6-27
PC-Translation-Specification Exception . . .	6-28
PER Event	6-28
Primary-Authority Exception	6-28
Privileged-Operation Exception	6-29
Protection Exception	6-30
Region-First-Translation Exception	6-30
Region-Second-Translation Exception . . .	6-31
Region-Third-Translation Exception	6-31
Secondary-Authority Exception	6-32
Segment-Translation Exception	6-32
Space-Switch Event	6-32
Special-Operation Exception	6-33
Specification Exception	6-35
Stack-Empty Exception	6-38
Stack-Full Exception	6-38
Stack-Operation Exception	6-39
Stack-Specification Exception	6-39
Stack-Type Exception	6-39
Trace-Table Exception	6-39
Translation-Specification Exception	6-40
Collective Program-Interruption Names	6-40
Recognition of Access Exceptions	6-40
Multiple Program-Interruption Conditions . . .	6-44
Access Exceptions	6-46
ASN-Translation Exceptions	6-49
Subspace-Replacement Exceptions	6-49
Trace Exceptions	6-49
Restart Interruption	6-49
Supervisor-Call Interruption	6-50
Priority of Interruptions	6-50

Chapter 7, General Instructions7-1

Data Format	7-5
Binary-Integer Representation	7-5
Binary Arithmetic	7-6
Signed Binary Arithmetic	7-6
Addition and Subtraction	7-6
Fixed-Point Overflow	7-7
Unsigned Binary Arithmetic	7-7
Signed and Logical Comparison	7-8
Instructions	7-8
ADD	7-24
ADD IMMEDIATE	7-25
ADD HALFWORD	7-27
ADD HALFWORD IMMEDIATE	7-27
ADD HIGH	7-27
ADD IMMEDIATE HIGH	7-28
ADD LOGICAL	7-28
ADD LOGICAL IMMEDIATE	7-28
ADD LOGICAL HIGH	7-29
ADD LOGICAL WITH CARRY	7-29
ADD LOGICAL WITH SIGNED IMMEDIATE . .	7-30

ADD LOGICAL WITH SIGNED IMMEDIATE HIGH	7-31
AND	7-31
AND IMMEDIATE	7-32
BRANCH AND LINK	7-33
BRANCH AND SAVE	7-34
BRANCH AND SAVE AND SET MODE	7-35
BRANCH AND SET MODE	7-36
BRANCH ON CONDITION	7-36
BRANCH ON COUNT	7-37
BRANCH ON INDEX HIGH	7-38
BRANCH ON INDEX LOW OR EQUAL	7-38
BRANCH RELATIVE AND SAVE	7-39
BRANCH RELATIVE AND SAVE LONG	7-39
BRANCH RELATIVE ON CONDITION	7-40
BRANCH RELATIVE ON CONDITION LONG	7-40
BRANCH RELATIVE ON COUNT	7-41
BRANCH RELATIVE ON COUNT HIGH	7-41
BRANCH RELATIVE ON INDEX HIGH	7-41
BRANCH RELATIVE ON INDEX LOW OR EQUAL	7-41
CHECKSUM	7-43
CIPHER MESSAGE	7-46
CIPHER MESSAGE WITH CHAINING	7-46
KM-Query (KM Function Code 0)	7-50
KM-DEA (KM Function Code 1)	7-51
KM-Encrypted-DEA (KM Function Code 9)	7-51
KM-TDEA-128 (KM Function Code 2)	7-51
KM-Encrypted-TDEA-128 (KM Function Code 10)	7-51
KM-TDEA-192 (KM Function Code 3)	7-53
KM-Encrypted-TDEA-192 (KM Function Code 11)	7-53
KM-AES-128 (KM Function Code 18)	7-54
KM-Encrypted-AES-128 (KM Function Code 26)	7-54
KM-AES-192 (KM Function Code 19)	7-55
KM-Encrypted-AES-192 (KM Function Code 27)	7-55
KM-AES-256 (KM Function Code 20)	7-56
KM-Encrypted-AES-256 (KM Function Code 28)	7-56
KM-XTS-AES-128 (KM Function Code 50)	7-57
KM-XTS-Encrypted-AES-128 (KM Function Code 58)	7-57
KM-XTS-AES-256 (KM Function Code 52)	7-59
KM-XTS-Encrypted-AES-256 (KM Function Code 60)	7-59
KMC-Query (KMC Function Code 0)	7-60
KMC-DEA (KMC Function Code 1)	7-61
KMC-Encrypted-DEA (KMC Function Code 9)	7-61
KMC-TDEA-128 (KMC Function Code 2)	7-62

KMC-Encrypted-TDEA-128 (KMC Function Code 10)	7-62	KMO-Query (Function Code 0)	7-102
KMC-TDEA-192 (KMC Function Code 3)	7-63	KMO-DEA (Function Code 1)	7-103
KMC-Encrypted-TDEA-192 (KMC Function Code 11)	7-63	KMO-Encrypted-DEA (Function Code 9)	7-103
KMC-AES-128 (KMC Function Code 18)	7-65	KMO-TDEA-128 (Function Code 2)	7-104
KMC-Encrypted-AES-128 (KMC Function Code 26)	7-65	KMO-Encrypted-TDEA-128 (Function Code 10)	7-104
KMC-AES-192 (KMC Function Code 19)	7-66	KMO-TDEA-192 (Function Code 3)	7-105
KMC-Encrypted-AES-192 (KMC Function Code 27)	7-66	KMO-Encrypted-TDEA-192 (Function Code 11)	7-105
KMC-AES-256 (KMC Function Code 20)	7-67	KMO-AES-128 (Function Code 18)	7-107
KMC-Encrypted-AES-256 (KMC Function Code 28)	7-67	KMO-Encrypted-AES-128 (Function Code 26)	7-107
KMC-PRNG (KMC Function Code 67)	7-69	KMO-AES-192 (Function Code 19)	7-108
CIPHER MESSAGE WITH CFB	7-71	KMO-Encrypted-AES-192 (Function Code 27)	7-108
KMF-Query (Function Code 0)	7-75	KMO-AES-256 (Function Code 20)	7-110
KMF-DEA (Function Code 1)	7-75	KMO-Encrypted-AES-256 (Function Code 28)	7-110
KMF-Encrypted-DEA (Function Code 9)	7-75	COMPARE	7-112
KMF-TDEA-128 (Function Code 2)	7-77	COMPARE IMMEDIATE	7-112
KMF-Encrypted-TDEA-128 (Function Code 10)	7-77	COMPARE RELATIVE LONG	7-112
KMF-TDEA-192 (Function Code 3)	7-78	COMPARE AND BRANCH	7-113
KMF-Encrypted-TDEA-192 (Function Code 11)	7-78	COMPARE AND BRANCH RELATIVE	7-113
KMF-AES-128 (Function Code 18)	7-80	COMPARE IMMEDIATE AND BRANCH	7-113
KMF-Encrypted-AES-128 (Function Code 26)	7-80	COMPARE IMMEDIATE AND BRANCH RELATIVE	7-114
KMF-AES-192 (Function Code 19)	7-82	COMPARE AND FORM CODEWORD	7-115
KMF-Encrypted-AES-192 (Function Code 27)	7-82	COMPARE AND SWAP	7-122
KMF-AES-256 (Function Code 20)	7-83	COMPARE DOUBLE AND SWAP	7-122
KMF-Encrypted-AES-256 (Function Code 28)	7-83	COMPARE AND SWAP AND STORE	7-124
CIPHER MESSAGE WITH COUNTER	7-86	COMPARE AND TRAP	7-127
KMCTR-Query (Function Code 0)	7-90	COMPARE IMMEDIATE AND TRAP	7-127
KMCTR-DEA (Function Code 1)	7-90	COMPARE HALFWORD	7-128
KMCTR-Encrypted-DEA (Function Code 9)	7-90	COMPARE HALFWORD IMMEDIATE	7-128
KMCTR-TDEA-128 (Function Code 2)	7-91	COMPARE HALFWORD RELATIVE LONG	7-128
KMCTR-Encrypted-TDEA-128 (Function Code 10)	7-91	COMPARE HIGH	7-129
KMCTR-TDEA-192 (Function Code 3)	7-92	COMPARE IMMEDIATE HIGH	7-129
KMCTR-Encrypted-TDEA-192 (Function Code 11)	7-92	COMPARE LOGICAL	7-130
KMCTR-AES-128 (Function Code 18)	7-94	COMPARE LOGICAL IMMEDIATE	7-130
KMCTR-Encrypted-AES-128 (Function Code 26)	7-94	COMPARE LOGICAL RELATIVE LONG	7-130
KMCTR-AES-192 (Function Code 19)	7-95	COMPARE LOGICAL AND BRANCH	7-132
KMCTR-Encrypted-AES-192 (Function Code 27)	7-95	COMPARE LOGICAL AND BRANCH RELATIVE	7-132
KMCTR-AES-256 (Function Code 20)	7-96	COMPARE LOGICAL IMMEDIATE AND BRANCH	7-132
KMCTR-Encrypted-AES-256 (Function Code 28)	7-96	COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE	7-132
CIPHER MESSAGE WITH OFB	7-98	COMPARE LOGICAL AND TRAP	7-133
		COMPARE LOGICAL IMMEDIATE AND TRAP	7-133
		COMPARE LOGICAL CHARACTERS UNDER MASK	7-134
		COMPARE LOGICAL HIGH	7-135
		COMPARE LOGICAL IMMEDIATE HIGH	7-135

COMPARE LOGICAL LONG	7-135	EXECUTE	7-207
COMPARE LOGICAL LONG EXTENDED. . .	7-138	EXECUTE RELATIVE LONG	7-207
COMPARE LOGICAL LONG UNICODE . . .	7-140	EXTRACT ACCESS	7-208
COMPARE LOGICAL STRING	7-143	EXTRACT CACHE ATTRIBUTE	7-208
COMPARE UNTIL SUBSTRING EQUAL . .	7-144	EXTRACT CPU TIME	7-210
COMPRESSION CALL	7-147	EXTRACT PSW	7-210
COMPUTE INTERMEDIATE MESSAGE		FIND LEFTMOST ONE	7-211
DIGEST	7-159	INSERT CHARACTER	7-211
COMPUTE LAST MESSAGE DIGEST . . .	7-159	INSERT CHARACTERS UNDER MASK. . .	7-212
KIMD-Query (KIMD Function Code 0) . .	7-162	INSERT IMMEDIATE	7-212
KIMD-SHA-1 (KIMD Function Code 1). .	7-162	INSERT PROGRAM MASK.	7-213
KIMD-SHA-256 (KIMD Function Code 2). .	7-163	LOAD	7-213
KIMD-SHA-512 (KIMD Function Code 3). .	7-163	LOAD IMMEDIATE	7-213
KIMD-GHASH (Function Code 65). . . .	7-164	LOAD RELATIVE LONG	7-214
KLMD-Query (KLMD Function Code 0) . .	7-164	LOAD ACCESS MULTIPLE	7-214
KLMD-SHA-1 (KLMD Function Code 1). .	7-165	LOAD ADDRESS	7-215
KLMD-SHA-256 (KLMD Function Code 2) .	7-166	LOAD ADDRESS EXTENDED	7-215
KLMD-SHA-512 (KLMD Function Code 3) .	7-168	LOAD ADDRESS RELATIVE LONG.	7-216
COMPUTE MESSAGE AUTHENTICATION		LOAD AND ADD	7-217
CODE	7-171	LOAD AND ADD LOGICAL.	7-217
KMAC-Query (Function Code 0)	7-175	LOAD AND AND	7-218
KMAC-DEA (Function Code 1).	7-175	LOAD AND EXCLUSIVE OR.	7-218
KMAC-Encrypted-DEA (Function Code 9) .	7-175	LOAD AND OR	7-219
KMAC-TDEA-128 (Function Code 2) . . .	7-176	LOAD AND TEST	7-219
KMAC-Encrypted-TDEA-128 (Function		LOAD BYTE	7-220
Code 10)	7-176	LOAD BYTE HIGH	7-221
KMAC-TDEA-192 (Function Code 3) . . .	7-177	LOAD COMPLEMENT	7-221
KMAC-Encrypted-TDEA-192 (Function		LOAD HALFWORD	7-221
Code 11)	7-177	LOAD HALFWORD IMMEDIATE	7-222
KMAC-AES-128 (Function Code 18) . . .	7-178	LOAD HALFWORD RELATIVE LONG	7-222
KMAC-Encrypted-AES-128 (Function		LOAD HALFWORD HIGH	7-222
Code 26)	7-178	LOAD HIGH.	7-223
KMAC-AES-192 (Function Code 19) . . .	7-179	LOAD LOGICAL	7-223
KMAC-Encrypted-AES-192 (Function		LOAD LOGICAL RELATIVE LONG.	7-223
Code 27)	7-179	LOAD LOGICAL CHARACTER.	7-223
KMAC-AES-256 (Function Code 20) . . .	7-179	LOAD LOGICAL CHARACTER HIGH.	7-224
KMAC-Encrypted-AES-256 (Function		LOAD LOGICAL HALFWORD.	7-224
Code 28)	7-179	LOAD LOGICAL HALFWORD RELATIVE	
CONVERT TO BINARY	7-181	LONG	7-224
CONVERT TO DECIMAL.	7-182	LOAD LOGICAL HALFWORD HIGH.	7-225
CONVERT UTF-16 TO UTF-32	7-183	LOAD LOGICAL IMMEDIATE	7-225
CONVERT UTF-16 TO UTF-8	7-186	LOAD LOGICAL THIRTY ONE BITS	7-226
CONVERT UNICODE TO UTF-8.	7-186	LOAD MULTIPLE	7-226
CONVERT UTF-32 TO UTF-16	7-189	LOAD MULTIPLE DISJOINT.	7-227
CONVERT UTF-32 TO UTF-8	7-192	LOAD MULTIPLE HIGH	7-227
CONVERT UTF-8 TO UTF-16	7-195	LOAD NEGATIVE	7-227
CONVERT UTF-8 TO UNICODE.	7-195	LOAD ON CONDITION	7-228
CONVERT UTF-8 TO UTF-32	7-199	LOAD PAIR DISJOINT	7-229
COPY ACCESS	7-203	LOAD PAIR FROM QUADWORD.	7-230
DIVIDE	7-203	LOAD POSITIVE	7-230
DIVIDE LOGICAL.	7-204	LOAD REVERSED	7-230
DIVIDE SINGLE.	7-204	MONITOR CALL	7-231
EXCLUSIVE OR.	7-205	Monitor-Event Program Interruption	7-232
EXCLUSIVE OR IMMEDIATE	7-206	Monitor-Event Counting Operation	7-232

MOVE	7-232	PCC-Compute-XTS-Parameter-Using-Encrypt-	
MOVE INVERSE	7-233	ed-AES-256 (Function Code 60)	7-268
MOVE LONG	7-234	PERFORM LOCKED OPERATION	7-270
MOVE LONG EXTENDED	7-238	Function Codes 0-3 (Compare and Load)	7-273
MOVE LONG UNICODE	7-241	Function Codes 4-7 (Compare and Swap)	7-275
MOVE NUMERICS	7-244	Function Codes 8-11 (Double Compare	
MOVE STRING	7-245	and Swap)	7-275
MOVE WITH OFFSET	7-246	Function Codes 12-15 (Compare and	
MOVE ZONES	7-246	Swap and Store)	7-276
MULTIPLY	7-247	Function Codes 16-19 (Compare and	
MULTIPLY HALFWORD	7-248	Swap and Double Store)	7-277
MULTIPLY HALFWORD IMMEDIATE	7-248	Function Codes 20-23 (Compare and	
MULTIPLY LOGICAL	7-249	Swap and Triple Store)	7-278
MULTIPLY SINGLE	7-249	POPULATION COUNT	7-284
MULTIPLY SINGLE IMMEDIATE	7-250	PREFETCH DATA	7-285
OR	7-250	PREFETCH DATA RELATIVE LONG	7-285
OR IMMEDIATE	7-251	ROTATE LEFT SINGLE LOGICAL	7-286
PACK	7-252	ROTATE THEN AND SELECTED BITS	7-287
PACK ASCII	7-253	ROTATE THEN EXCLUSIVE OR	
PACK UNICODE	7-254	SELECTED BITS	7-287
PERFORM CRYPTOGRAPHIC		ROTATE THEN OR SELECTED BITS	7-287
COMPUTATION	7-255	ROTATE THEN INSERT SELECTED BITS	7-288
PCC-Query (Function Code 0)	7-257	ROTATE THEN INSERT SELECTED BITS	
PCC-Compute-Last-Block-CMAC-Using-		HIGH	7-290
DEA (Function Code 1)	7-258	ROTATE THEN INSERT SELECTED BITS	
PCC-Compute-Last-Block-CMAC-Using-		LOW	7-290
Encrypted-DEA (Function Code 9)	7-258	SEARCH STRING	7-291
PCC-Compute-Last-Block-CMAC-Using-		SEARCH STRING UNICODE	7-293
TDEA-128 (Function Code 2)	7-259	SET ACCESS	7-295
PCC-Compute-Last-Block-CMAC-Using-En-		SET ADDRESSING MODE	7-295
crypt-TDEA-128 (Function Code 10)	7-259	SET PROGRAM MASK	7-296
PCC-Compute-Last-Block-CMAC-Using-		SHIFT LEFT DOUBLE	7-296
TDEA-192 (Function Code 3)	7-260	SHIFT LEFT DOUBLE LOGICAL	7-297
PCC-Compute-Last-Block-CMAC-Using-En-		SHIFT LEFT SINGLE	7-297
crypt-TDEA-192 (Function Code 11)	7-260	SHIFT LEFT SINGLE LOGICAL	7-298
PCC-Compute-Last-Block-CMAC-Using-		SHIFT RIGHT DOUBLE	7-299
AES-128 (Function Code 18)	7-262	SHIFT RIGHT DOUBLE LOGICAL	7-299
PCC-Compute-Last-Block-CMAC-Using-En-		SHIFT RIGHT SINGLE	7-300
crypt-AES-128 (Function Code 26)	7-262	SHIFT RIGHT SINGLE LOGICAL	7-301
PCC-Compute-Last-Block-CMAC-Using-		STORE	7-301
AES-192 (Function Code 19)	7-263	STORE RELATIVE LONG	7-301
PCC-Compute-Last-Block-CMAC-Using-En-		STORE ACCESS MULTIPLE	7-302
crypt-AES-192 (Function Code 27)	7-263	STORE CHARACTER	7-302
PCC-Compute-Last-Block-CMAC-Using-		STORE CHARACTER HIGH	7-303
AES-256 (Function Code 20)	7-264	STORE CHARACTERS UNDER MASK	7-303
PCC-Compute-Last-Block-CMAC-Using-En-		STORE CLOCK	7-304
crypt-AES-256 (Function Code 28)	7-264	STORE CLOCK FAST	7-304
PCC-Compute-XTS-Parameter-Using-		STORE CLOCK EXTENDED	7-305
AES-128 (Function Code 50)	7-266	STORE FACILITY LIST EXTENDED	7-306
PCC-Compute-XTS-Parameter-Using-Encrypt-		STORE HALFWORD	7-307
ed-AES-128 (Function Code 58)	7-266	STORE HALFWORD RELATIVE LONG	7-307
PCC-Compute-XTS-Parameter-Using-		STORE HALFWORD HIGH	7-308
AES-256 (Function Code 52)	7-268	STORE HIGH	7-308
		STORE MULTIPLE	7-308

STORE MULTIPLE HIGH	7-309
STORE ON CONDITION	7-309
STORE PAIR TO QUADWORD	7-310
STORE REVERSED	7-310
SUBTRACT	7-311
SUBTRACT HALFWORD	7-312
SUBTRACT HIGH	7-312
SUBTRACT LOGICAL	7-313
SUBTRACT LOGICAL IMMEDIATE	7-313
SUBTRACT LOGICAL HIGH	7-314
SUBTRACT LOGICAL WITH BORROW	7-314
SUPERVISOR CALL	7-315
TEST ADDRESSING MODE	7-315
TEST AND SET	7-316
TEST UNDER MASK (TEST UNDER MASK HIGH, TEST UNDER MASK LOW)	7-316
TRANSLATE	7-317
TRANSLATE AND TEST	7-318
TRANSLATE AND TEST EXTENDED	7-319
TRANSLATE AND TEST REVERSE EXTENDED	7-319
TRANSLATE AND TEST REVERSE	7-323
TRANSLATE EXTENDED	7-324
TRANSLATE ONE TO ONE	7-327
TRANSLATE ONE TO TWO	7-327
TRANSLATE TWO TO ONE	7-327
TRANSLATE TWO TO TWO	7-327
UNPACK	7-331
UNPACK ASCII	7-332
UNPACK UNICODE	7-333
UPDATE TREE	7-334
Protection of Cryptographic Key	7-339
Protection of DES Keys	7-340
Protection of AES Keys	7-341

Chapter 8, Decimal Instructions8-1

Decimal-Number Formats	8-1
Zoned Format	8-1
Packed-Decimal Formats	8-1
Signed-Packed-Decimal Format	8-1
Unsigned-Packed-Decimal Format	8-2
Decimal Codes	8-2
Decimal Operations	8-3
Decimal-Arithmetic Instructions	8-3
Editing Instructions	8-4
Execution of Decimal Instructions	8-4
Other Instructions for Decimal Operands	8-4
Decimal-Operand Data Exception	8-4
Instructions	8-5
ADD DECIMAL	8-6
COMPARE DECIMAL	8-6
DIVIDE DECIMAL	8-7
EDIT	8-7

EDIT AND MARK	8-10
MULTIPLY DECIMAL	8-11
SHIFT AND ROUND DECIMAL	8-12
SUBTRACT DECIMAL	8-13
TEST DECIMAL	8-13
ZERO AND ADD	8-13

Chapter 9, Floating-Point Overview and Support Instructions 9-1

Sign Bit	9-2
Finite Floating-Point Numbers	9-2
Infinities	9-2
Not-A-Number (NaN)	9-2
Signaling and Quiet NaNs	9-2
Payload	9-3
Propagation of NaNs	9-3
Default QNaN	9-3
Hexadecimal-Floating-Point (HFP)	9-3
Binary Floating-Point (BFP)	9-4
Decimal Floating-Point (DFP)	9-4
Canonical DFP Data	9-4
Comparison of Floating-Point Number Representations	9-4
Floating-Point Number Ranges	9-4
Equivalent Floating-Point Number Representations	9-5
Effective Width	9-7
Floating-Point Data in Storage	9-8
Registers And Controls	9-8
Floating-Point Registers	9-8
Additional Floating-Point (AFP) Registers	9-9
Valid Floating-Point-Register Designations	9-9
Floating-Point-Control (FPC) Register	9-9
IEEE Masks and Flags	9-10
FPC DXC Byte	9-10
Operations on the FPC Register	9-10
AFP-Register-Control Bit	9-11
IEEE Computational Operations	9-11
Intermediate Values	9-11
Precise Intermediate Value	9-12
Precision-Rounded Value	9-12
Denormalized Value	9-12
Functionally-Rounded Value	9-12
Rounded Intermediate Value	9-12
Scaled Value	9-12
Scale Factor (Ψ)	9-12
Unsigned Scaling Exponent (α)	9-12
Signed Scaling Exponent (Ω)	9-12
IEEE Rounding	9-13
Permissible Set	9-13
Selection of Candidates	9-13
Ties	9-13

Voting Digit and Common-Rounding- Point View	9-13	Impacts of the Floating-Point Extension Facility	9-51
Rounding Methods	9-13		
Explicit Rounding Methods	9-16		
Summary of Rounding Action	9-17		
IEEE Exceptions	9-17	Chapter 10, Control Instructions . . . 10-1	
Concurrent IEEE Exceptions	9-18	BRANCH AND SET AUTHORITY	10-6
IEEE Invalid Operation	9-18	BRANCH AND STACK	10-10
IEEE Division-By-Zero	9-19	BRANCH IN SUBSPACE GROUP	10-13
IEEE Overflow	9-19	COMPARE AND SWAP AND PURGE	10-17
IEEE Underflow	9-19	DIAGNOSE	10-19
IEEE Inexact	9-20	EXTRACT AND SET EXTENDED AUTHORITY	10-20
Quantum Exception	9-21	EXTRACT PRIMARY ASN	10-20
Suppression of Certain IEEE Exceptions	9-23	EXTRACT PRIMARY ASN AND INSTANCE	10-20
IEEE Same-Radix Format Conversion	9-23	EXTRACT SECONDARY ASN	10-21
IEEE Comparison	9-24	EXTRACT SECONDARY ASN AND INSTANCE	10-21
Condition Codes for IEEE Instructions	9-24	EXTRACT STACKED REGISTERS	10-21
Instructions	9-24	EXTRACT STACKED STATE	10-23
CONVERT BFP TO HFP	9-27	INSERT ADDRESS SPACE CONTROL	10-25
CONVERT HFP TO BFP	9-28	INSERT PSW KEY	10-26
COPY SIGN	9-29	INSERT STORAGE KEY EXTENDED	10-26
EXTRACT FPC	9-30	INSERT VIRTUAL STORAGE KEY	10-27
LOAD	9-30	INVALIDATE DAT TABLE ENTRY	10-28
LOAD COMPLEMENT	9-30	INVALIDATE PAGE TABLE ENTRY	10-32
LOAD FPC	9-31	LOAD ADDRESS SPACE PARAMETERS	10-34
LOAD FPC AND SIGNAL	9-31	LOAD CONTROL	10-44
LOAD FPR FROM GR	9-33	LOAD PAGE TABLE ENTRY ADDRESS	10-44
LOAD GR FROM FPR	9-33	LOAD PSW	10-47
LOAD NEGATIVE	9-33	LOAD PSW EXTENDED	10-48
LOAD POSITIVE	9-34	LOAD REAL ADDRESS	10-49
LOAD ZERO	9-34	LOAD USING REAL ADDRESS	10-53
PERFORM FLOATING-POINT OPERATION	9-34	MODIFY STACKED STATE	10-53
General Register 0 (GR0)	9-35	MOVE PAGE	10-54
Return Code	9-37	MOVE TO PRIMARY	10-56
Sign Preservation	9-37	MOVE TO SECONDARY	10-56
Preferred Quantum	9-37	MOVE WITH DESTINATION KEY	10-58
NaN Conversion	9-38	MOVE WITH KEY	10-59
Scaled Value and Signed Scaling Exponent (Ω) for PFPO	9-38	MOVE WITH OPTIONAL SPECIFICATIONS	10-60
HFP Values	9-38	MOVE WITH SOURCE KEY	10-63
HFP Zero Result	9-38	PAGE IN	10-64
HFP Overflow and Underflow for PFPO	9-38	PAGE OUT	10-65
IEEE Exceptions for PFPO	9-39	PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATIONS	10-66
PFPO Action Figures	9-44	PCKMO-Query (Function Code 0)	10-67
SET BFP ROUNDING MODE	9-45	PCKMO-Encrypt-DEA-Key (Function Code 1)	10-68
SET DFP ROUNDING MODE	9-45	PCKMO-Encrypt-TDEA-128-Key (Function Code 2)	10-68
SET FPC	9-46	PCKMO-Encrypt-TDEA-192-Key (Function Code 3)	10-68
SET FPC AND SIGNAL	9-46	PCKMO-Encrypt-AES-128-Key (Function Code 18)	10-68
STORE	9-47		
STORE FPC	9-47		
Summary of All Floating-Point Instructions	9-47		
Impacts on ESA/390	9-51		
Impacts of the Decimal-Floating-Point Facility	9-51		

PCKMO-Encrypt-AES-192-Key (Function Code 19)	10-69
PCKMO-Encrypt-AES-256-Key (Function Code 20)	10-69
PERFORM FRAME MANAGEMENT FUNCTION.	10-70
PERFORM TIMING FACILITY FUNCTION.	10-73
PERFORM TOPOLOGY FUNCTION	10-80
Operation of Function Codes 0 and 1	10-81
Operation of Function Code 2:	10-81
PROGRAM CALL.	10-81
PROGRAM RETURN.	10-94
PROGRAM TRANSFER	10-98
PROGRAM TRANSFER WITH INSTANCE.	10-98
PURGE ALB.	10-107
PURGE TLB.	10-107
RESET REFERENCE BIT EXTENDED.	10-107
RESET REFERENCE BITS MULTIPLE	10-108
RESUME PROGRAM	10-108
SET ADDRESS SPACE CONTROL	10-110
SET ADDRESS SPACE CONTROL FAST	10-111
SET CLOCK.	10-112
SET CLOCK COMPARATOR	10-113
SET CLOCK PROGRAMMABLE FIELD	10-113
SET CPU TIMER	10-114
SET PREFIX	10-114
SET PSW KEY FROM ADDRESS.	10-115
SET SECONDARY ASN	10-115
SET SECONDARY ASN WITH INSTANCE	10-115
SET STORAGE KEY EXTENDED.	10-121
Setting Storage Keys in Multiple 4K-Byte Blocks.	10-123
SET SYSTEM MASK	10-124
SIGNAL PROCESSOR	10-124
STORE CLOCK COMPARATOR.	10-126
STORE CONTROL	10-126
STORE CPU ADDRESS	10-126
STORE CPU ID	10-127
STORE CPU TIMER	10-128
STORE FACILITY LIST	10-128
STORE PREFIX.	10-128
STORE REAL ADDRESS	10-129
STORE SYSTEM INFORMATION.	10-130
CPU Topology Overview	10-145
CPU Slack	10-147
STORE THEN AND SYSTEM MASK	10-149
STORE THEN OR SYSTEM MASK.	10-149
STORE USING REAL ADDRESS	10-150
TEST ACCESS	10-150
TEST BLOCK.	10-152
TEST PROTECTION	10-155
TRACE.	10-157
TRAP	10-159

Chapter 11, Machine-Check

Handling	11-1
Machine-Check Detection	11-2
Correction of Machine Malfunctions	11-2
Error Checking and Correction	11-2
CPU Retry	11-2
Effects of CPU Retry	11-3
Checkpoint Synchronization	11-3
Handling of Machine Checks during	
Checkpoint Synchronization	11-3
Checkpoint-Synchronization Operations	11-3
Checkpoint-Synchronization Action.	11-4
Channel-Subsystem Recovery	11-4
Unit Deletion	11-4
Handling of Machine Checks.	11-4
Validation.	11-5
Invalid CBC in Storage	11-6
Programmed Validation of Storage	11-7
Invalid CBC in Storage Keys.	11-7
Invalid CBC in Registers	11-8
Check-Stop State	11-9
System Check Stop.	11-10
Machine-Check Interruption	11-10
Exigent Conditions	11-10
Repressible Conditions	11-10
Interruption Action	11-11
Point of Interruption.	11-13
Machine-Check-Interruption Code	11-13
Subclass	11-14
System Damage	11-14
Instruction-Processing Damage	11-14
System Recovery	11-14
Timing-Facility Damage.	11-15
External Damage.	11-15
Degradation.	11-15
Warning.	11-16
Channel Report Pending.	11-16
Service-Processor Damage.	11-16
Channel-Subsystem Damage	11-16
Subclass Modifiers	11-16
Backed Up.	11-16
Delayed Access Exception	11-16
Ancillary Report.	11-17
Synchronous Machine-Check-Interruption	
Conditions	11-17
Processing Backup	11-17
Processing Damage	11-17
Storage Errors	11-18
Storage Error Uncorrected	11-18
Storage Error Corrected	11-18
Storage-Key Error Uncorrected	11-18
Storage Degradation	11-18
Indirect Storage Error	11-19

Machine-Check Interruption-Code Validity		Manual Indicator	12-4
Bits	11-19	Power Controls	12-4
PSW-MWP Validity	11-19	Rate Control	12-4
PSW Mask and Key Validity	11-19	Restart Key	12-4
PSW Program-Mask and Condition-Code		Start Key	12-5
Validity	11-19	Stop Key	12-5
PSW-Instruction-Address Validity	11-20	Store-Status Key	12-5
Failing-Storage-Address Validity	11-20	System-Reset-Clear Key	12-5
External-Damage-Code Validity	11-20	System-Reset-Normal Key	12-5
Floating-Point-Register Validity	11-20	Test Indicator	12-5
General-Register Validity	11-20	TOD-Clock Control	12-6
Control-Register Validity	11-20	Wait Indicator	12-6
Storage Logical Validity	11-20	Multiprocessing Configurations	12-6
Access-Register Validity	11-20		
TOD-Programmable-Register Validity	11-20	Chapter 13, I/O Overview	13-1
Floating-Point-Control-Register Validity	11-20	Input/Output (I/O)	13-1
CPU-Timer Validity	11-20	The Channel Subsystem	13-1
Clock-Comparator Validity	11-20	Subchannel Sets	13-2
Machine-Check Extended Interruption		Subchannels	13-2
Information	11-21	Attachment of Input/Output Devices	13-3
Register-Save Areas	11-21	Channel Paths	13-3
External-Damage Code	11-21	Control Units	13-4
Failing-Storage Address	11-22	I/O Devices	13-5
Handling of Machine-Check Conditions	11-22	I/O Addressing	13-5
Floating Interruption Conditions	11-22	Subchannel-Set Identifier	13-5
Floating Machine-Check-Interruption		Channel-Path Identifier	13-5
Conditions	11-23	Subchannel Number	13-5
Floating I/O Interruptions	11-23	Device Number	13-6
Machine-Check Masking	11-23	Device Identifier	13-6
Channel-Report-Pending Subclass Mask	11-23	Fibre-Channel Extensions	13-6
Recovery Subclass Mask	11-23	I/O-Command Words	13-6
Degradation Subclass Mask	11-24	Transport Command Word (TCW)	13-7
External-Damage Subclass Mask	11-24	Channel Program Organization	13-7
Warning Subclass Mask	11-24	CCW Channel Program	13-7
Machine-Check Logout	11-24	TCW Channel Program	13-7
Summary of Machine-Check Masking	11-24	Execution of I/O Operations	13-8
		Start-Function Initiation	13-8
Chapter 12, Operator Facilities . . .	12-1	Subchannel Operation Modes	13-9
Manual Operation	12-1	Path Management	13-9
Basic Operator Facilities	12-1	Channel-Program Execution	13-9
Address-Compare Controls	12-1	Conclusion of I/O Operations	13-10
Alter-and-Display Controls	12-2	Chaining When Using a CCW Channel	
Architectural-Mode Indicator	12-2	Program	13-11
Architectural-Mode-Selection Controls	12-2	Chaining When Using a TCW Channel	
Check-Stop Indicator	12-2	Program	13-11
IML Controls	12-3	Premature Conclusion of I/O Operations	13-12
Interrupt Key	12-3	I/O Interruptions	13-12
Load Indicator	12-3		
Load-Clear Key	12-3	Chapter 14, I/O Instructions	14-1
Load-Clear-List-Directed Key	12-3	I/O-Instruction Formats	14-1
Load-Normal Key	12-3	I/O-Instruction Execution	14-2
Load-with-Dump Key	12-3	Serialization	14-2
Load-Unit-Address Controls	12-4	Operand Access	14-2

Condition Code	14-2
Program Exceptions	14-2
Instructions	14-2
CANCEL SUBCHANNEL	14-4
CLEAR SUBCHANNEL	14-5
HALT SUBCHANNEL	14-6
MODIFY SUBCHANNEL	14-8
RESET CHANNEL PATH	14-9
RESUME SUBCHANNEL	14-10
SET ADDRESS LIMIT	14-12
SET CHANNEL MONITOR	14-13
START SUBCHANNEL	14-15
STORE CHANNEL PATH STATUS	14-17
STORE CHANNEL REPORT WORD	14-17
STORE SUBCHANNEL	14-18
TEST PENDING INTERRUPTION	14-19
TEST SUBCHANNEL	14-20

Chapter 15, Basic I/O Functions 15-1

Control of Basic I/O Functions	15-1
Subchannel-Information Block	15-2
Path-Management-Control Word	15-2
Subchannel-Status Word	15-9
Model-Dependent Area/Measurement	
Block Address	15-9
Summary of Modifiable Fields	15-10
Channel-Path Allegiance	15-12
Working Allegiance	15-12
Working Allegiance for Subchannels	
Operating in Command Mode	15-12
Working Allegiance for Subchannels	
Operating in Transport Mode	15-12
Active Allegiance	15-13
Dedicated Allegiance	15-13
Channel-Path Availability	15-13
Control-Unit Type	15-14
Clear Function	15-14
Clear-Function Path Management	15-14
Clear-Function Subchannel Modification	15-15
Clear-Function Signaling and Completion	15-15
Halt Function	15-16
Halt-Function Path Management	15-16
Halt-Function Signaling and Completion	15-17
Start Function and Resume Function	15-20
Start-Function and Resume-Function Path	
Management	15-20
Interrogate Function	15-22
Interrogate-Function Path Management	15-22
Interrogate TCCB Transportation and	
Completion	15-22
Execution of I/O Operations	15-23
Blocking of Data	15-24
Operation-Request Block	15-24

Command-Mode ORB	15-25
Transport-Mode ORB	15-30
Channel-Command Word	15-31
Transport Control Word	15-33
Transport-Command-Control Block	15-36
Transport-Command-Area Header	15-36
Transport-Command Area	15-37
Device-Command Word	15-37
Transport-Command-Area Trailer	15-38
Transfer-CBC-Offset Block DCW	15-40
CBC-Offset Block	15-40
Interrogate TCCB	15-41
Interrogate DCW	15-41
Interrogate Data	15-41
Transport Status Block	15-43
Transport-Status Header (TSH)	15-43
I/O-Status TSA	15-44
Device-Detected-Program-Check TSA	15-45
Interrogate TSA	15-50
Command Code	15-52
Designation of Storage Area	15-52
CCW Channel Program Chaining	15-55
Data Chaining	15-56
Command Chaining	15-58
TCW Channel Program Chaining	15-58
Skipping	15-59
Program-Controlled Interruption	15-59
Indirect-Storage Designator (ISD)	15-61
CCW Indirect Data Addressing	15-61
Modified CCW Indirect Data Addressing	15-63
Transport Indirect Data Addressing	15-65
Suspension of CCW Channel-Program	
Execution	15-68
Commands and Flags for CCWs	15-70
Branching in CCW Channel Programs	15-71
Transfer in Channel	15-72
Command Retry	15-72
Concluding I/O Operations before Initiation	15-72
Concluding I/O Operations during Initiation	15-73
Immediate Conclusion of Command-Mode I/O	
Operations	15-73
Concluding I/O Operations During Data	
Transfer	15-74
Channel-Path-Reset Function	15-75
Channel-Path-Reset-Function Signaling	15-76
Channel-Path-Reset-Function-Completion	
Signaling	15-76

Chapter 16, I/O Interruptions 16-1

Interruption Conditions	16-2
Intermediate Interruption Condition	16-4
Primary Interruption Condition	16-4
Secondary Interruption Condition	16-5

Channel-Path Reset	17-12
I/O-System Reset	17-13
Externally Initiated Functions	17-16
Initial Program Loading	17-16
CCW-type IPL	17-17
List-Directed IPL	17-19
Reconfiguration of the I/O System	17-22
Status Verification	17-22
Address-Limit Checking	17-22
Configuration Alert	17-23
Incorrect-Length-Indication Suppression	17-23
Concurrent Sense	17-24
Channel-Subsystem Recovery	17-24
Channel Report	17-25
Channel-Report Word	17-26
Restore-Subchannel Facility	17-28
Extended-Subchannel-Logout Facility	17-28
Channel-Subsystem-I/O-Priority Facility	17-29
Number of Channel-Subsystem-Priority Levels	17-29
Multiple-Subchannel-Set Facility	17-29

Chapter 18, Hexadecimal-Floating-Point Instructions18-1

HFP Arithmetic	18-1
HFP Number Representation	18-1
Normalization	18-2
HFP Data Formats	18-2
HFP Short Format	18-4
HFP Long Format	18-4
HFP Extended Format	18-4
Instructions	18-5
ADD NORMALIZED	18-8
ADD UNNORMALIZED	18-9
COMPARE	18-10
CONVERT FROM FIXED	18-10
CONVERT TO FIXED	18-11
DIVIDE	18-11
HALVE	18-13
LOAD AND TEST	18-13
LOAD COMPLEMENT	18-14
LOAD FP INTEGER	18-14
LOAD LENGTHENED	18-15
LOAD NEGATIVE	18-15
LOAD POSITIVE	18-16
LOAD ROUNDED	18-16
MULTIPLY	18-17
MULTIPLY AND ADD	18-18
MULTIPLY AND SUBTRACT	18-19
MULTIPLY AND ADD UNNORMALIZED	18-20
MULTIPLY UNNORMALIZED	18-21
SQUARE ROOT	18-23

SUBTRACT NORMALIZED	18-24
SUBTRACT UNNORMALIZED	18-24

Chapter 19, Binary-Floating-Point

Instructions 19-1

Binary-Floating-Point Facility	19-1
Floating-Point-Control (FPC) Register	19-2
BFP Arithmetic	19-2
BFP Data Formats	19-2
BFP Short Format	19-2
BFP Long Format	19-2
BFP Extended Format	19-2
Biased Exponent	19-2
Significand	19-3
Values of Nonzero Numbers	19-3
Classes of BFP Data	19-4
Zeros	19-4
Subnormal Numbers	19-4
Normal Numbers	19-4
Infinites	19-4
Signaling and Quiet NaNs	19-4
BFP-Format Conversion	19-5
BFP Rounding	19-5
BFP Comparison	19-5
Remainder	19-5
IEEE Exceptions	19-7
Result Figures	19-7
Data-Exception Codes (DXC) and Abbreviations	19-7
Instructions	19-8
ADD	19-11
COMPARE	19-16
COMPARE AND SIGNAL	19-17
CONVERT FROM FIXED	19-18
CONVERT FROM LOGICAL	19-20
CONVERT TO FIXED	19-21
CONVERT TO LOGICAL	19-23
DIVIDE	19-25
DIVIDE TO INTEGER	19-27
LOAD AND TEST	19-30
LOAD COMPLEMENT	19-30
LOAD FP INTEGER	19-31
LOAD LENGTHENED	19-32
LOAD NEGATIVE	19-33
LOAD POSITIVE	19-33
LOAD ROUNDED	19-34
MULTIPLY	19-35
MULTIPLY AND ADD	19-37
MULTIPLY AND SUBTRACT	19-37
SQUARE ROOT	19-38
SUBTRACT	19-39
TEST DATA CLASS	19-39

Chapter 20, Decimal-Floating-Point

Instructions	20-1
Decimal-Floating-Point Facility	20-1
DFP Arithmetic	20-2
Finite Floating-Point Number	20-2
Cohort	20-2
Quantum	20-2
Preferred Quantum	20-2
Scaled Preferred Quantum	20-3
Delivered Quantum	20-3
Special Quantum-Handling Operations	20-3
DFP Data Formats	20-3
DFP Short Format	20-3
DFP Long Format	20-3
DFP Extended Format	20-4
Sign	20-4
Combination	20-4
Encoded Trailing Significand	20-5
Values of Finite Numbers	20-5
Significand	20-5
DFP Significant Digits	20-5
Canonical Declets	20-6
DFP Canonical Data	20-6
Classes of DFP Data	20-6
Zeros	20-6
Subnormal Numbers	20-6
Normal Numbers	20-7
Infinities	20-7
Signaling and Quiet NaNs	20-7
Canonicalization	20-7
DFP-Format Conversion	20-7
DFP Rounding	20-7
DFP Comparison	20-7
DFP Formatting Instructions	20-7
Signed-Packed-Decimal Format	20-8
Unsigned-Packed-Decimal Format	20-8
IEEE Exceptions	20-8
Summary of Preferred Quantum	20-9
Summary of Rounding And Range Actions	20-9
Result Figures	20-14
Data-Exception Codes (DXC) and Abbreviations	20-14
Instructions	20-15
ADD	20-19
COMPARE	20-21
COMPARE AND SIGNAL	20-22
COMPARE BIASED EXPONENT	20-23
CONVERT FROM FIXED	20-24
CONVERT FROM LOGICAL	20-25
CONVERT FROM SIGNED PACKED	20-26
CONVERT FROM UNSIGNED PACKED	20-27
CONVERT TO FIXED	20-27
CONVERT TO LOGICAL	20-29
CONVERT TO SIGNED PACKED	20-31
CONVERT TO UNSIGNED PACKED	20-32
DIVIDE	20-32
EXTRACT BIASED EXPONENT	20-34
EXTRACT SIGNIFICANCE	20-35
INSERT BIASED EXPONENT	20-35
LOAD AND TEST	20-36
LOAD FP INTEGER	20-37
LOAD LENGTHENED	20-40
LOAD ROUNDED	20-41
MULTIPLY	20-43
QUANTIZE	20-45
REROUND	20-47
SHIFT SIGNIFICAND LEFT	20-49
SHIFT SIGNIFICAND RIGHT	20-50
SUBTRACT	20-51
TEST DATA CLASS	20-51
TEST DATA GROUP	20-52
Densely Packed Decimal (DPD)	20-53
Decimal-to-DPD Mapping	20-53
DPD-to-Decimal Mapping	20-53

Appendix A, Number Representation and Instruction-Use Examples

Number Representation	A-2
Binary Integers	A-2
Signed Binary Integers	A-2
Unsigned Binary Integers	A-3
Decimal Integers	A-4
Hexadecimal-Floating-Point Numbers	A-5
Conversion Example	A-6
Instruction-Use Examples	A-7
Machine Format	A-7
Assembler-Language Format	A-7
Addressing Mode in Examples	A-7
General Instructions	A-7
ADD HALFWORD (AH)	A-8
AND (N, NC, NI, NR)	A-8
NI Example	A-8
Linkage Instructions (BAL, BALR, BAS, BASR, BASSM, BSM)	A-8
Other BALR and BASR Examples	A-10
BRANCH AND STACK (BAKR)	A-10
BAKR Example 1	A-10
BAKR Example 2	A-11
BAKR Example 3	A-11
BRANCH ON CONDITION (BC, BCR)	A-12
BRANCH ON COUNT (BCT, BCTR)	A-12
BRANCH ON INDEX HIGH (BXH)	A-12
BXH Example 1	A-12
BXH Example 2	A-13
BRANCH ON INDEX LOW OR EQUAL (BXLE)	A-13

BXLE Example 1	A-14
BXLE Example 2	A-14
COMPARE AND FORM CODEWORD (CFC)	A-14
COMPARE HALFWORD (CH)	A-14
COMPARE LOGICAL (CL, CLC, CLI, CLR)	A-15
CLC Example	A-15
CLI Example	A-15
CLR Example	A-15
COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)	A-16
COMPARE LOGICAL LONG (CLCL)	A-16
COMPARE LOGICAL STRING (CLST)	A-18
CONVERT TO BINARY (CVB)	A-18
CONVERT TO DECIMAL (CVD)	A-19
DIVIDE (D, DR)	A-19
EXCLUSIVE OR (X, XC, XI, XR)	A-20
XC Example	A-20
XI Example	A-21
EXECUTE (EX)	A-21
FIND LEFTMOST ONE (FLOGR)	A-22
INSERT CHARACTERS UNDER MASK (ICM)	A-23
LOAD (L, LR)	A-23
LOAD ADDRESS (LA)	A-23
LOAD HALFWORD (LH)	A-24
MOVE (MVC, MVI)	A-24
MVC Example	A-24
MVI Example	A-25
MOVE INVERSE (MVCIN)	A-25
MOVE LONG (MVCL)	A-26
MOVE NUMERICS (MVN)	A-26
MOVE STRING (MVST)	A-27
MOVE WITH OFFSET (MVO)	A-27
MOVE ZONES (MVZ)	A-28
MULTIPLY (M, MR)	A-28
MULTIPLY HALFWORD (MH)	A-29
OR (O, OC, OI, OR)	A-29
OI Example	A-29
PACK (PACK)	A-29
ROTATE THEN EXCLUSIVE OR SELECTED BITS	A-30
ROTATE THEN INSERT SELECTED BITS	A-30
ROTATE THEN OR SELECTED BITS	A-30
SEARCH STRING (SRST)	A-31
SRST Example 1	A-31
SRST Example 2	A-31
SHIFT LEFT DOUBLE (SLDA)	A-31
SHIFT LEFT SINGLE (SLA)	A-32
STORE CHARACTERS UNDER MASK (STCM)	A-32
STORE MULTIPLE (STM)	A-32
TEST UNDER MASK (TM)	A-33
TRANSLATE (TR)	A-33
TRANSLATE AND TEST (TRT)	A-34

UNPACK (UNPK)	A-35
UPDATE TREE (UPT)	A-36
Decimal Instructions	A-36
ADD DECIMAL (AP)	A-36
COMPARE DECIMAL (CP)	A-36
DIVIDE DECIMAL (DP)	A-37
EDIT (ED)	A-37
EDIT AND MARK (EDMK)	A-38
MULTIPLY DECIMAL (MP)	A-39
SHIFT AND ROUND DECIMAL (SRP)	A-39
Decimal Left Shift	A-39
Decimal Right Shift	A-39
Decimal Right Shift and Round	A-40
Multiplying by a Variable Power of 10	A-40
ZERO AND ADD (ZAP)	A-41
Hexadecimal-Floating-Point Instructions	A-41
ADD NORMALIZED (AD, ADR, AE, AER, AXR)	A-41
ADD UNNORMALIZED (AU, AUR, AW, AWR)	A-42
COMPARE (CD, CDR, CE, CER)	A-42
DIVIDE (DD, DDR, DE, DER)	A-42
HALVE (HDR, HER)	A-43
MULTIPLY (MD, MDR, MDE, MDER, MXD, MXDR, MXR)	A-43
Hexadecimal-Floating-Point-Number Conversion	A-44
Fixed Point to Hexadecimal Floating Point	A-44
Hexadecimal Floating Point to Fixed Point	A-44
Multiprogramming and Multiprocessing Examples	A-45
Example of a Program Failure Using OR Immediate	A-45
Conditional Swapping Instructions (CS, CDS)	A-46
Setting a Single Bit	A-46
Updating Counters	A-47
Bypassing Post and Wait	A-47
Lock/Unlock	A-47
Lock/Unlock with LIFO Queuing for Contentions	A-48
Lock/Unlock with FIFO Queuing for Contentions	A-49
Free-Pool Manipulation	A-50
PERFORM LOCKED OPERATION (PLO)	A-51
Sorting Instructions	A-53
Tree Format	A-53
Example of Use of Sort Instructions	A-54

Appendix B, Lists of Instructions . . . B-1

Instructions Arranged by Name	B-4
Instructions Arranged by Mnemonic	B-23
Instructions Arranged by Operation Code	B-41

Appendix C, Condition-Code Settings	C-1	Appendix I, EBCDIC and ISO-8 Codes	I-1
Appendix G, Table of Powers of 2	G-1	Control Character Representations	I-2
Appendix H, Hexadecimal Tables	H-1	Formatting Character Representations	I-2
		Additional ISO-8 Control Character Representations	I-2
		Index	1

Notices

References in this publication to IBM® products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY, 10504-1785 USA.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX/ESA
BookMaster
CICS
DB2
Enterprise Systems Architecture/370
Enterprise Systems Architecture/390
Enterprise Systems Connection Architecture
ESA/370
ESA/390
ESCON
FICON
IBM
IBMLink
MVS/ESA
OS/390
Processor Resource/Systems Manager
PR/SM
Sysplex Timer
System/370
VM/ESA
z/Architecture
z/OS

ANSI is a registered trademark of the American National Standards Institute in the United States, other countries, or both.

IEEE is a trademark of the Institute of Electrical and Electronics Engineers, Inc. in the United States, other countries, or both.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other Countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries or both.

Unicode is a registered trademark of Unicode, Incorporated in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other trademarks and registered trademarks are the properties of their respective companies.

Preface

This publication provides, for reference purposes, a detailed z/Architecture™ description.

The publication applies only to systems operating as defined by z/Architecture. For systems operating in accordance with the Enterprise Systems Architecture/390® (ESA/390™) definition, the *IBM ESA/390 Principles of Operation*, SA22-7201, should be consulted.

The publication describes each function at the level of detail needed to prepare an assembler-language program that relies on that function. It does not, however, describe the notation and conventions that must be employed in preparing such a program, for which the user must instead refer to the appropriate assembler-language publication.

The information in this publication is provided principally for use by assembler-language programmers, although anyone concerned with the functional details of z/Architecture will find it useful.

This publication is written as a reference and should not be considered an introduction or a textbook. It assumes the user has a basic knowledge of data-processing systems.

All facilities discussed in this publication are not necessarily available on every model. Furthermore, in some instances the definitions have been structured to allow for some degree of extendibility, and therefore certain capabilities may be described or implied that are not offered on any model. Examples of such capabilities are the use of a 16-bit field in the sub-system-identification word to identify the subchannel number, the size of the CPU address, and the number of CPUs sharing main storage. The allowance for this type of extendibility should not be construed as implying any intention by IBM to provide such capabilities. For information about the characteristics and availability of facilities on a specific model, see the functional characteristics publication for that model.

Largely because this publication is arranged for reference, certain words and phrases appear, of necessity, earlier in the publication than the principal discussions explaining them. The reader who encounters a problem because of this arrangement

should refer to the index, which indicates the location of the key description.

The information presented in this publication is grouped in 20 chapters and several appendixes:

Chapter 1, Introduction, highlights the major facilities of z/Architecture.

Chapter 2, Organization, describes the major groupings within the system — main storage, expanded storage, the central processing unit (CPU), the external time reference (ETR), and input/output — with some attention given to the composition and characteristics of those groupings.

Chapter 3, Storage, explains the information formats, the addressing of storage, and the facilities for storage protection. It also deals with dynamic address translation (DAT), which, coupled with special programming support, makes the use of a virtual storage possible.

Chapter 4, Control, describes the facilities for the switching of system status, for special externally initiated operations, for debugging, and for timing. It deals specifically with CPU states, control modes, the program-status word (PSW), control registers, tracing, program-event recording, timing facilities, resets, store status, and initial program loading.

Chapter 5, Program Execution, explains the role of instructions in program execution, looks in detail at instruction formats, and describes briefly the use of the program-status word (PSW), of branching, and of interruptions. It contains the principal description of the advanced address-space facilities that were introduced in ESA/370™. It also details the aspects of program execution on one CPU as observed by other CPUs and by channel programs.

Chapter 6, Interruptions, details the mechanism that permits the CPU to change its state as a result of conditions external to the system, within the system, or within the CPU itself. Six classes of interruptions are identified and described: machine-check interruptions, program interruptions, supervisor-call interruptions, external interruptions, input/output interruptions, and restart interruptions.

Chapter 7, General Instructions, contains detailed descriptions of logical and binary-integer data formats and of most of the unprivileged instructions. Other unprivileged instructions are described in chapters 8-10 and 18-20.

Chapter 8, Decimal Instructions, describes in detail decimal data formats and the decimal instructions.

Chapter 9, Floating-Point Overview and Support Instructions, includes an introduction to the floating-point operations, detailed descriptions of those instructions common to binary-floating-point, decimal-floating-point, and hexadecimal-floating-point operations, and summaries of all floating-point instructions.

Chapter 10, Control Instructions, contains detailed descriptions of all of the semiprivileged and privileged instructions except for the I/O instructions.

Chapter 11, Machine-Check Handling, describes the mechanisms for detecting, correcting, and reporting machine malfunctions.

Chapter 12, Operator Facilities, describes the basic manual functions and controls available for operating and controlling the system.

Chapters 13-17 of this publication provide a detailed definition of the functions performed by the channel subsystem and the logical interface between the CPU and the channel subsystem.

Chapter 13, I/O Overview, provides a brief description of the basic components and operation of the channel subsystem.

Chapter 14, I/O Instructions, contains the description of the I/O instructions.

Chapter 15, Basic I/O Functions, describes the basic I/O functions performed by the channel subsystem, including the initiation, control, and conclusion of I/O operations.

Chapter 16, I/O Interruptions, covers I/O interruptions and interruption conditions.

Chapter 17, I/O Support Functions, describes such functions as channel-subsystem usage monitoring, resets, initial-program loading, reconfiguration, and channel-subsystem recovery.

Chapter 18, Hexadecimal-Floating-Point Instructions, contains detailed descriptions of the hexadecimal-floating-point (HFP) data formats and the HFP instructions.

Chapter 19, Binary-Floating-Point Instructions, contains detailed descriptions of the binary-floating-point (BFP) data formats and the BFP instructions.

Chapter 20, Decimal-Floating-Point Instructions, contains detailed descriptions of the decimal-floating-point (DFP) data formats and the DFP instructions.

The *Appendixes* include:

- Information about number representation
- Instruction-use examples
- Lists of the instructions arranged in several sequences
- A summary of the condition-code settings
- A table of the powers of 2
- Tabular information helpful in dealing with hexadecimal numbers
- A table of EBCDIC and other codes.

Size and Number Notation

In this publication, the letters K, M, G, T, P, and E denote the multipliers 2^{10} , 2^{20} , 2^{30} , 2^{40} , 2^{50} , and 2^{60} , respectively. Although the letters are borrowed from the decimal system and stand for kilo (10^3), mega (10^6), giga (10^9), tera (10^{12}), peta (10^{15}), and exa (10^{18}), they do not have the decimal meaning but instead represent the power of 2 closest to the corresponding power of 10. Their meaning in this publication is as follows:

Symbol	Value
K (kilo)	$1,024 = 2^{10}$
M (mega)	$1,048,576 = 2^{20}$
G (giga)	$1,073,741,824 = 2^{30}$
T (tera)	$1,099,511,627,776 = 2^{40}$
P (peta)	$1,125,899,906,842,624 = 2^{50}$
E (exa)	$1,152,921,504,606,846,976 = 2^{60}$

The following are some examples of the use of K, M, G, T, and E:

2,048 is expressed as 2K.

4,096 is expressed as 4K.
65,536 is expressed as 64K (not 65K).
 2^{24} is expressed as 16M.
 2^{31} is expressed as 2G.
 2^{42} is expressed as 4T.
 2^{64} is expressed as 16E.

When the words “thousand” and “million” are used, no special power-of-2 meaning is assigned to them.

All numbers in this publication are in decimal unless they are explicitly noted as being in binary or hexadecimal (hex).

Bytes, Characters, and Codes

Although the System/360 architecture was originally designed to support the Extended Binary-Coded-Decimal Interchange Code (EBCDIC), the instructions and data formats of the architecture are for the most part independent of the external code which is to be processed by the machine. For most instructions, all 256 possible combinations of bit patterns for a particular byte can be processed, independent of the character which the bit pattern is intended to represent. For instructions which use the zoned format, and for those few instructions which are dependent on a particular external code, the instruction TRANS-LATE may be used to convert data from one code to another code. Thus, a machine operating in accordance with z/Architecture can process EBCDIC, ASCII, or any other code which can be represented in eight or fewer bits per character.

In this publication, unless otherwise specified, the value given for a byte is the value obtained by considering the bits of the byte to represent a binary code. Thus, when a byte is said to contain a zero, the value 00000000 binary, or 00 hex, is meant, and not the value for an EBCDIC character “0,” which would be F0 hex.

Other Publications

The parallel-I/O interface is described in the publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information*, GA22-6974.

The parallel-I/O channel-to-channel adapter is described in the publication *IBM Enterprise Systems Architecture/390 Channel-to-Channel Adapter for the*

System/360 and System/370 I/O Interface, SA22-7091.

The Enterprise Systems Connection Architecture® (ESCON®) I/O interface, referred to in this publication along with the FICON I/O interface as the serial-I/O interface, is described in the publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

The FICON I/O interface is described in the ANSI® standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

The channel-to-channel adapter for the serial-I/O interface is described in the publication *IBM Enterprise Systems Architecture/390 ESCON Channel-to-Channel Adapter*, SA22-7203.

The commands, status, and sense data that are common to all I/O devices that comply with z/Architecture are described in the publication *IBM Enterprise Systems Architecture/390 Common I/O-Device Commands and Self Description*, SA22-7204.

The compression facility is described in the publication *IBM Enterprise Systems Architecture/390 Data Compression*, SA22-7208. The z/Architecture form of the COMPRESSION CALL instruction is described in this publication.

The interpretive-execution facility is described in the publication *IBM 370-XA Interpretive Execution*, SA22-7095.

The set-program-parameter and CPU-measurement facilities are described in the publication *The Set-Program-Parameter and CPU-Measurement Facilities*, SA23-2260.

Summary of Changes in Ninth Edition

The ninth edition of this publication differs from the previous edition principally by containing the definitions of the following facilities:

- CMPSC-enhancement facility
- Distinct-operands facility
- Enhanced-monitor facility
- Fast-BCR-serialization facility

- Floating-point extension facility
- High-word facility
- Interlocked-access facility
- IPTE-range facility
- Load/store-on-condition facility
- Message-security-assist extension 3
- Message-security-assist extension 4
- Nonquiescing key-setting facility
- Population-count facility
- The Reset-reference-bits-multiple facility

The ninth edition contains the following significant changes relative to the previous edition:

- In Chapter 3, “Storage:”
 - Changes introduced by the reset-reference-bits-multiple are added to various sections.
 - Changes introduced by the enhanced-monitor facility are added.
 - Changes introduced by the access-exception-fetch/store-indication facility are added.
 - Clarification is added to the description of the translation-exception identification.
- In Chapter 4, “Control:”
 - Changes introduced by the enhanced-monitor facility are added.
 - Clarification is made to the description of the PER ASCE identification (AI) and PER access identification (PAID).
 - Changes introduced by the high-word facility are added.
 - Descriptions of the new facility indications are added.
- In Chapter 5, “Program Execution:”
 - A significant editorial change has been made to the description of the instruction formats. Instruction formats having variations in assembler syntax or field content are designated by an alphabetic suffix which is also reflected in the instruction descriptions in chapters 7-10, 14, and 18-20.
 - Changes introduced by the enhanced-monitor facility are added.
 - Clarification is made to the description of suppression.

- Changes introduced by the message-security assist extensions 3 and 4 are added.
- Changes introduced by the nonquiescing-key-setting facility are added, including the description of quiescing.
- Changes introduced by the interlocked-access facility are added, including the description of an interlocked-fetch reference and specific-operand serialization.
- In Chapter 6, “Interruptions:”
 - Changes introduced by the enhanced-monitor facility are added.
 - Changes introduced by the floating-point extension facility are added.
 - Clarification is provided regarding the handling of data exceptions resulting from the compare-and-trap instructions.
- In Chapter 7, “General Instructions:”
 - Clarification is made as to the content of the chapter.
 - Changes introduced by the distinct-operands facility are added.
 - Changes introduced by the high-word facility are added.
 - Changes introduced by the interlocked-access facility are added.
 - Changes introduced by the load/store-on-condition facility are added.
 - Changes introduced by the message-security-assist extension 3 are added, including a new section on the protection of cryptographic keys.
 - Changes introduced by the message-security-assist extension 4 are added.
 - Changes introduced by the population-count facility are added.
 - Changes introduced by the instruction-format clarifications are added.
 - Clarifications are made to the description of COMPARE AND SWAP AND STORE.
 - Changes introduced by the CMPSC-enhancement facility are added.

- Extended assembler mnemonics are defined for the ROTATE THEN AND SELECTED BITS, ROTATE THEN OR SELECTED BITS, and ROTATE THEN EXCLUSIVE OR SELECTED BITS instructions.
- Clarification is made to the descriptions of the SEARCH STRING and SEARCH STRING UNICODE instructions.
- In Chapter 8, “Decimal Instructions:”
 - Changes introduced by the instruction-format clarifications are added.
 - Clarification is made to the description of the EDIT and EDIT AND MARK instructions.
- In Chapter 9, “Floating-Point Overview and Support Instructions:”
 - Changes introduced by the instruction-format clarifications are added.
 - Changes introduced by the floating-point-extension facility are added.
- In Chapter 10, “Control Instructions:”
 - Clarification is made as to the content of the chapter; specifically, the chapter also contains the description of certain nonprivileged instructions.
 - Changes introduced by the instruction-format clarifications are added.
 - Changes introduced by the IPTE-range facility are added.
 - Changes introduced by the message-security-assist extension 3 are added.
 - Changes introduced by the nonquiescing key-setting facility are added.
 - Changes introduced by the reset-reference-bits-multiple facility are added.
 - Changes are made to the STORE SYSTEM INFORMATION instruction with regards to topology information returned.
- In Chapter 14, “Basic I/O Functions,” the CBC-offset block is added for the fibre-channel-extensions facility.

- In Chapter 18, “Hexadecimal-Floating Instructions,” changes introduced by the instruction-format clarifications are added.
- In Chapter 19, “Binary-Floating-Point Instructions:”
 - Changes introduced by the instruction-format clarifications are added.
 - Changes introduced by the floating-point-extension facility are added.
- In Chapter 20, “Decimal-Floating-Point Instructions:”
 - Changes introduced by the instruction-format clarifications are added.
 - Changes introduced by the floating-point-extension facility are added.

The ninth edition also contains numerous minor corrections and clarifications.

Summary of Changes in Eighth Edition

The eighth edition of this publication differs from the previous edition principally by containing the definitions of the following facility:

- Fibre-channel-extensions (FCX) facility

The eighth edition contains the following significant changes relative to the previous edition:

- In Chapter 1, “Introduction,” a summary of the fibre-channel-extensions facility is added.
- In Chapter 4, “Control,” the clock setting for the leap second on January 1, 2009 is added to the section “Timing.”
- In Chapter 7, “General Instructions,” the clock setting for the leap second on January 1, 2009 is added to the description of STORE CLOCK EXTENDED.
- In Chapter 13, “I/O Overview,” changes introduced by the fibre-channel-extensions facility are added to various sections.

- In Chapter 14, “I/O Instructions,” changes introduced by the fibre-channel-extensions facility are added to various sections.
- In Chapter 15, “Basic I/O Functions,” changes introduced by the fibre-channel-extensions facility are added to various sections.
- In Chapter 16, “I/O Interruptions,” changes introduced by the fibre-channel-extensions facility are added to various sections.
- In Chapter 17, “I/O Support Functions,” changes introduced by the fibre-channel-extensions facility are added to various sections.

The eighth edition also contains numerous minor corrections and clarifications.

Summary of Changes in Seventh Edition

The seventh edition of this publication differs from the previous edition principally by containing the definitions of the following facilities:

- Compare-and-swap-and-store facility 2
- Configuration-topology facility
- Enhanced-DAT facility
- Execute-extensions facility
- General-instructions-extension facility
- Message-security-assist extension 2
- Move-with-optional-specifications facility
- Parsing-enhancement facility
- Restore-subchannel facility

The seventh edition contains the following significant changes relative to the previous edition:

- In Chapter 3, “Storage:”
 - Changes introduced by the enhanced-DAT facility are added to various sections.
 - Changes introduced by the enhanced suppression-on-protection function are added
 - Changes introduced by the execute-extensions facility are added to various sections.
 - Changes introduced by the general-instructions-extension facility are added to the section “Assigned Storage Locations.”

- Changes introduced by the move-with-optional-specifications facility are added to the section “Protection.”
- In Chapter 4, “Control:”
 - Changes introduced by the enhanced-DAT facility are added to various sections.
 - Changes introduced by the execute-extensions facility are added to various sections.
 - Changes introduced by the general-instructions-extension facility are added to various sections.
 - New facility bits for the compare-and-swap-and-store facility 2, configuration-topology facility, enhanced-DAT facility, execute-extensions facility, general-instructions-extension facility, move-with-optional-specifications facility, and parsing-enhancement facility are added to the section “Facility Indications.”
- In Chapter 5, “Program Execution:”
 - In the section “Instruction Formats,” the RIS, RRS, and SIL formats are added. Four new versions of the RIE instruction format and one new version of the RXY format are added.
 - Changes introduced by the enhanced-DAT facility are added to various sections.
 - Changes introduced by the execute-extensions facility are added to various sections.
 - Changes introduced by the general-instructions-extension facility are added to various sections.
 - Changes introduced by the move-with-optional-specifications facility are added to various sections.
 - Changes introduced by the parsing-enhancement facility are added to the section “Multiple-Access References.”
- In Chapter 6, “Interruptions:”
 - Changes introduced by the compare-and-swap-and-store facility 2 are added to the section “Specification Exception.”
 - Changes introduced by the enhanced-DAT facility are added to various sections.

- Changes introduced by the execute-extensions facility are added to various sections.
- Changes introduced by the general-instructions-extension facility are added to various sections.
- Changes introduced by the move-with-optional-specifications facility are added to various sections.
- Changes introduced by the parsing-enhancement facility are added to section “Specification Exception.”
- In Chapter 7, “General Instructions:”
 - For various instruction descriptions in which multiple instruction formats are present, headings have been added to distinguish one format from another.
 - Changes introduced by the compare-and-swap-and-store facility 2 are added to the description of the COMPARE AND SWAP AND STORE instruction.
 - Changes introduced by the message-security-assist extension 2 are added to the descriptions of the CIPHER MESSAGE, CIPHER MESSAGE WITH CHAINING, COMPUTE INTERMEDIATE MESSAGE DIGEST and COMPUTE LAST MESSAGE DIGEST instructions.
 - Programming notes are added to the description of the UTF conversion instructions (CU12, CU14, CU21, CU24, CU41, and CU42) indicating that the instructions support big-endian encoding only.
 - Changes introduced by the execute-extensions facility are added to various sections, including the description of the EXECUTE RELATIVE LONG instruction.
 - Changes introduced by the general-instructions-extension facility are added to various sections, including the descriptions of the facility’s 72 new instructions.
 - The description of the MOVE LONG UNICODE instruction is amended such that an odd length specification is permitted.
 - Changes introduced by the parsing-enhancement facility are added to various sections, including the descriptions of the TRANSLATE AND TEST EXTENDED and TRANSLATE AND TEST REVERSE EXTENDED instructions.
- In Chapter 10, “Control Instructions:”
 - Changes introduced by the move-with-optional-specifications facility are added to various sections, including the description of the MOVE WITH OPTIONAL SPECIFICATIONS instruction.
 - Changes introduced by the enhanced-DAT facility are added to various sections, including the description of the PERFORM FRAME MANAGEMENT FUNCTION instruction.
 - Changes introduced by the configuration-topology facility are added to various sections, including the description of the PERFORM TOPOLOGY FUNCTION instruction and enhancements to the STORE SYSTEM INFORMATION instruction.
 - Changes introduced by the execute-extensions facility are added to various sections.
 - The STORE SYSTEM INFORMATION instruction is enhanced to include the following:
 - Additional information returned in SYSIB 1.1.1.
 - The definition of SYSIB 15.1.2 in support of the configuration-topology facility.
- In Chapter 11, “Machine-Check Handling:”
 - Changes introduced by the enhanced-DAT facility are added to various sections.
 - Changes introduced by the execute-extensions facility are added to various sections.
- In Chapter 13, “I/O Overview,” changes introduced by the multiple-subchannel set facility are described.
- In Chapter 14, “I/O Instructions,” changes introduced by the multiple-subchannel set facility are described.
- In Chapter 17, “I/O Support Functions,” changes introduced by the multiple-subchannel set facility are described.

The seventh edition also contains numerous minor corrections and clarifications.

Summary of Changes in Sixth Edition

The sixth edition of this publication differs from the previous edition principally by containing the definitions of the following facilities:

- Compare-and-swap-and-store facility
- Conditional-SSKE facility
- Decimal-floating-point facility
- Decimal-floating-point-rounding facility
- Extract-CPU-time facility
- Floating-point-support-sign-handling facility
- FPR-GR-transfer facility
- IEEE-exception-simulation facility
- PFPO facility

The sixth edition contains the following significant changes relative to the previous edition:

- In Chapter 3, “Storage”:
 - Changes introduced by the conditional-SSKE facility are added to various sections.
 - In the “Assigned Storage Locations” section, the descriptions of the various fields now include hexadecimal as well as decimal locations.
 - Clarification is made to the definition of the exception access identification and PER access identification at locations 160 and 161, respectively.
- In Chapter 4, “Control”:
 - Changes introduced by the extract-CPU-time facility are added to the section “CPU Timer.”
 - New facility bits for the compare-and-swap-and-store facility, decimal-floating-point facility, decimal-floating-point performance, extract-CPU-time facility, floating-point-support-enhancement facilities (FPR-GR-transfer, FPS-sign-handling, and DFP-rounding), and PFPO (PERFORM FLOATING-POINT OPERATION) facility are added to the section “Facility Indications.”
- In Chapter 5, “Program Execution”:
 - In the section “Instruction Formats”, the RRR and SSF formats are added.
 - Changes introduced by the compare-and-swap-and-store facility are added to various sections.
 - Changes introduced by the extract-CPU-time facility are added to the “Consistency Specification” section.
 - Previous restrictions on storing into the instruction stream in the access-register or home address-space-control modes are removed.
- In Chapter 6, “Interruptions”:
 - Changes introduced by the simulated IEEE exception (IXS) are added to the various sections.
 - Changes introduced by the compare-and-swap-and-store facility are added to various sections.
 - The descriptions of the special-operation and specification program interruptions are updated and rearranged.
 - Clarification is added to the section “Multiple Program-Interruption Conditions.”
- In Chapter 7, “General Instructions”:
 - The descriptions of the COMPARE AND SWAP AND STORE and EXTRACT CPU TIME instructions are added.
 - Programming notes for COMPARE LOGICAL LONG UNICODE and MOVE LONG UNICODE are amended to account for the long-displacement facility.
 - A programming note clarifying the setting of the condition code is added to AND IMMEDIATE, EXCLUSIVE OR IMMEDIATE, and OR IMMEDIATE.
- In Chapter 9, “Floating-Point Overview and Support Instructions”:
 - The concepts of “views” and “quantum” are introduced.
 - The name of the instruction SET ROUNDING MODE is changed to SET BFP ROUNDING MODE and is moved from Chapter 19 to Chapter 9.

- The following instructions operating on the floating-point-control (FPC) register are moved from Chapter 19 to Chapter 9: EXTRACT FPC, LOAD FPC, SET FPC, and STORE FPC.
 - The FPR-GR-transfer facility instructions, LOAD FPR FROM GR and LOAD GR FROM FPR, are added.
 - The four instructions making up the floating-point-support-sign-handling facility are added: COPY SIGN, LOAD COMPLEMENT, LOAD NEGATIVE, and LOAD POSITIVE.
 - The decimal-floating-point-rounding facility is added. This includes a 3-bit DFP rounding mode field in the floating-point control (FPC) register and the instruction SET DFP ROUNDING MODE.
 - The IEEE-exception-simulation facility is added, including the instructions SET FPC AND SIGNAL and LOAD FPC AND SIGNAL.
 - The instruction PERFORM FLOATING-POINT OPERATION (PFPO) is added.
- In Chapter 10, “Control Instructions”, changes introduced by the conditional-SSKE facility are added to the description of SET STORAGE KEY EXTENDED.
 - In Chapter 11, “Machine-Check Handling”, changes introduced by the conditional-SSKE facility are added.
 - In Chapter 12, the operator rate control is now defined to be model dependent.
 - In Chapter 14, clarification is added to the section “Modified CCW Indirect Data Addressing”.
 - In Chapter 18, “Hexadecimal-Floating-Point Instructions”: The names of two of the rounding methods used by the instruction CONVERT TO FIXED are changed from “biased round to nearest” and “round to nearest” to “round to nearest with ties away from 0” and “round to nearest with ties to even”, respectively.
 - In Chapter 19, “Binary-Floating-Point Instructions”: This chapter is updated extensively to be consistent with Chapter 20. This includes the following:
 - The terms “normal” and “subnormal”, referring to BFP data classes, replace the obsolete terms “normalized”, and “denormalized”, respectively.
 - The name of the instruction SET ROUNDING MODE is changed to SET BFP ROUNDING MODE and is moved from Chapter 19 to Chapter 9.
 - The following instructions operating on the floating-point-control (FPC) register are moved from Chapter 19 to Chapter 9: EXTRACT FPC, LOAD FPC, SET FPC, and STORE FPC.
 - The section “Floating-Point-Control (FPC) Register” is moved to Chapter 9.
 - The sections “BFP Rounding”, “BFP Comparison”, “Condition Codes for BFP Instructions”, and “IEEE Exception Conditions” are renamed and moved to Chapter 9.
 - In Chapter 20, “Decimal-Floating-Point Instructions”: This chapter is new, and describes the decimal-floating-point (DFP) facility.

The sixth edition also contains numerous minor corrections and clarifications.

Summary of Changes in Fifth Edition

The fifth edition of this publication differs from the previous edition principally by containing the definitions of the following facilities:

- DAT-enhancement facility 2
- ETF2-enhancement facility
- ETF3-enhancement facility
- Extended-immediate facility
- HFP-unnormalized-extensions facility
- Message-security-assist extension 1
- Modified-CCW-indirect-data-addressing facility
- PER-3 facility
- Server-time-protocol facility
- Store-clock-fast facility
- Store-facility-list-extended facility:
- TOD-clock-steering facility

The fifth edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 3, “Storage”:
 - In the “Information Formats” section, the length of a storage-operand field that is implied by the instruction now includes a 16-byte operand.
 - Changes introduced by the modified-CCW-indirect-data-addressing facility are added to various sections.
 - Changes introduced by the DAT-enhancement facility 2 are added to various sections.
 - Changes introduced by the PER-3 facility are added to the section “Assigned Storage Locations.”
 - Changes introduced by the store-facility-list-extended facility are added to section “Assigned Storage Locations.”
- In Chapter 4, “Control”:
 - Changes introduced by the store-clock-fast facility are added to various sections.
 - Changes introduced by the PER-3 facility, including breaking-event-address recording and PER instruction-fetch nullification, are added to various sections.
 - Changes introduced by the server-time protocol (STP) facility are added to various sections.
 - Changes introduced by the TOD-clock-steering facility are added to various sections.
 - The conditional-emergency-signal and sense-running-status orders are added to the section “Signal-Processor Orders.”
 - The location in which the PSW that is preserved as a result of switching from the z/Architecture to the ESA/390 architectural mode is formally named the *Captured z/Architecture PSW register*.
 - The description of all facility-indication bits has been moved to a new section at the end of the chapter.
- In Chapter 5, “Program Execution”:
 - The instructions of the message-security assist are added to the section “Condition Code Alternative to Interruptibility.”
 - Changes introduced by the DAT-enhancement facility 2 are added to various sections.
 - Changes introduced by the store-clock-fast facility are added to various sections.
 - Changes introduced by the modified-CCW-indirect-data-addressing facility are added to the section “Channel-Program Serialization.”
- In Chapter 6, “Interruptions”:
 - Changes introduced by the PER-3 facility are added to the various sections.
 - Changes introduced by the server-time protocol (STP) facility are added to various sections.
 - Changes introduced by the TOD-clock-steering facility are added to various sections.
 - Changes introduced by the DAT-enhancement facility 2 are added to various sections.
 - Changes introduced by the extended-immediate facility are added to the description of the specification exception program interruption.
- In Chapter 7, “General Instructions”:
 - A note is added to the “Instructions” section indicating that, for certain new or modified instructions, an operand may be optional.
 - Descriptions of thirty-four new instructions introduced by the extended-immediate facility are added.
 - A description of the STORE CLOCK FAST instruction is added.
 - A description of the STORE FACILITY LIST EXTENDED instruction is added.
 - Changes introduced by the TOD-clock-steering facility are added to the STORE CLOCK, STORE CLOCK FAST, and STORE CLOCK EXTENDED instructions.
 - Descriptions of new functions introduced by the message-security-assist extension 1 are added to the CIPHER MESSAGE, CIPHER MESSAGE WITH CHAINING, COMPUTE INTERMEDIATE MESSAGE DIGEST, and COMPUTE LAST MESSAGE DIGEST instructions.

- Changes introduced by the ETF-2 enhancement facility are added to the TRANSLATE ONE TO ONE, TRANSLATE ONE TO TWO, TRANSLATE TWO TO ONE, and TRANSLATE TWO TO TWO instructions.
- Changes introduced by the ETF-3 enhancement are added to the CONVERT UTF-16 TO UTF-32, CONVERT UTF-16 TO UTF-8, CONVERT UTF-8 TO UTF-16, and CONVERT UTF-8 TO UTF-32 instructions.
- In Chapter 10, “Control Instructions”:
 - A description of the LOAD PAGE-TABLE-ENTRY ADDRESS instruction, introduced by the DAT-enhancement facility 2, is added.
 - A description of the PERFORM TIMING FACILITY FUNCTION instruction, introduced by the TOD-clock-steering facility, is added.
 - Changes introduced by the TOD-clock-steering facility are added to the description of the SET CLOCK instruction.
 - The description of all facility-indication bits are moved to Chapter 4 (they are removed from the description of the STORE FACILITY LIST instruction).
 - A model-dependent field is defined in the SYSIB 2.2.2 operand stored by the STORE SYSTEM INFORMATION instruction.
 - Changes introduced by the store-clock-fast facility are added to the description of the TRACE instruction.
- In Chapter 11, “Machine-Check Handling”:
 - Changes introduced by the store-clock-fast facility are added to the section “CPU Retry.”
 - Changes introduced by the modified-CCW-indirect-data-addressing facility are added to the section “Invalid CBC in Storage.”
 - Changes introduced by the TOD-clock-steering facility are added to the section “Timing-Facility Damage.”
 - Changes introduced by the server-time-protocol facility are added to the section “External-Damage Code.”
- In Chapter 13, “I/O Overview”, changes introduced by the modified-CCW-indirect-data-address facility are added to the section “Channel-Program Execution”.
- In Chapter 15, “Basic I/O Functions”:
 - Changes introduced by the modified-CCW-indirect-data-addressing facility are added which include:
 - The modified-indirect-data-address word (MIDAW) is defined.
 - Bit 25 of word 1 of the operation-request block (ORB) is defined as the modified-CCW-indirect-data-addressing control.
 - Bit 7 of the flags field in the channel-command word (CCW) is defined as the modified-indirect-data-addressing flag.
- In Chapter 16, “I/O Interruptions”:
 - Changes introduced by the modified-CCW-indirect-data-address facility are added to the sections “Subchannel-Status Word” and “Extended-Status Word”.
 - Changes are introduced to the section “Interface-Control Check” to support retrying a CCW-type IPL when an interface-control check is detected during the execution of the IPL channel program.
- In Chapter 17, “I/O Support Functions”:
 - Changes are introduced to the section “CCW-Type IPL” to support retrying a CCW-type IPL when an interface-control check is detected during the execution of the IPL channel program.
 - Changes are introduced to the section “List-Directed IPL” to support storing of the sub-system-identification word (SID) for the IPL-device during list-directed IPL, if a subchannel is associated with the IPL-device.
- In Chapter 18, “Hexadecimal-Floating-Point Instructions”, descriptions of the twelve instructions introduced by the HFP-unnormalized-extensions facility are added.
- In Appendix I, character representations of EBCDIC code pages 81C, 94C, 500, and 1047, IBM-PC, and BookMaster symbols are removed. Only the commonly used EBCDIC code page 037 and ISO-8 characters are shown.

Summary of Changes in Fourth Edition

The fourth edition of this publication differs from the previous edition principally by containing the definitions of the extended-translation facility 3 and the ASN-and-LX-reuse facility. The fourth edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 3, “Storage”:
 - Changes introduced by the ASN-and-LX-reuse facility are added, including changes to the “Address Spaces” and “ASN Translation” sections.
 - A programming note is added to the “Dynamic Address Translation” section, describing implications in using common segments.
 - The ATMID and AI fields are corrected in the “Assigned Storage Locations” figure.
- In Chapter 4, “Control”:
 - Changes introduced by the ASN-and-LX-reuse facility are added, including changes to the “Trace” section.
 - The list-directed IPL function is added to the “Initial Program Load” section.
 - In the “Trace” section, serialization requirements for instructions that implicitly store into the trace table or linkage stack are relaxed.
- In Chapter 5, “Program Execution”:
 - A new RI and SS instruction format are included.
 - Changes introduced by the ASN-and-LX-reuse facility are added, including changes to the “Authorization Mechanisms”, “PC-Number Translation”, and “Linkage-Stack Operations” sections.
 - The instructions of the extended-translation facility 3 (except TRANSLATE AND TEST REVERSED) are added to the sections “Condition-Code Alternative to Interruptibility” and “Multiple-Access References.”
- Serialization requirements for instructions that implicitly store into the trace table or linkage stack are relaxed.
- In Chapter 6, “Interruptions”:
 - Changes introduced by the ASN-and-LX-reuse facility include the new LFX translation, LSX translation, LSTE sequence, and ASTE instance exceptions.
 - An additional condition for TRAP is added to the list of instructions that can cause a special-operation exception to be recognized.
 - SEARCH STRING UNICODE, a part of the extended-translation facility 3, is added to the list of instructions that can cause a specification exception to be recognized.
- In Chapter 7, “General Instructions”:
 - Six new instructions provided by the extended-translation facility 3 are added. The instructions CONVERT UNICODE TO UTF-8 (CUUTF) and CONVERT UTF-8 TO UNICODE (CUTFU) are renamed to CONVERT UTF-16 TO UTF-8 (CU21) and CONVERT UTF-8 TO UTF-16 (CU12), respectively. The old mnemonics continue to be recognized.
 - The instruction-format illustrations for STAMY and STMY are corrected.
- In Chapter 10, “Control Instructions”:
 - Changes introduced by the ASN-and-LX-reuse facility include the following:
 - Four new instructions provided by the facility are added.
 - The definitions of BRANCH AND STACK, BRANCH IN SUBSPACE GROUP, EXTRACT PRIMARY ASN, EXTRACT SECONDARY ASN, LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, and SET SECONDARY ASN are updated to account for the facility.
 - New facility bits for the ASN-and-LX-reuse facility and the extended-translation facility 3 are added to STORE FACILITY LIST.

- New fields are added to the system-information block (SYSIB) returned by STORE SYSTEM INFORMATION.
 - Serialization requirements for instructions that implicitly store into the trace table or linkage stack are relaxed, including PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER (WITH INSTANCE), SET SECONDARY ASN (WITH INSTANCE), and TRACE.
 - Corrections and clarifications are made to TRAP.
- In Chapter 12, “Operator Facilities”: Changes introduced by the list-directed IPL function are added, including the load-clear-list-directed key and the load-with-dump key.
 - In Chapter 14, “I/O Instructions”, Clarifications are added to programming note 1 of the CANCEL SUBCHANNEL instruction description.
 - In Chapter 16, “I/O Interruptions”:
 - Bit 1 of the extended-report word (ERW) is defined as the request-logging-only (L) bit.
 - Bit 2 of the ERW is defined as the extended-subchannel-logout-pending (E) bit.
 - Words 2-3 of the format-0 extended-status word (ESW) are defined as the failing-storage address when the failing-storage-address-validity (F) bit, bit 6 of the ERW, is one and as the extended-subchannel-logout descriptor (ESLD) when the extended-subchannel-logout-pending (E) bit, bit 2 of the ERW, is one. The E-bit is always zero when the F-bit is one, and the F-bit is always zero when the E-bit is one.
 - In Chapter 17, “I/O Support Functions”, list-directed IPL is added to support initial-program loading (IPL) from devices that are not accessed by CCWs. IPL from devices that are accessed by CCWs is designated CCW-type IPL.
 - In Appendix C, “Condition-Code Settings”: The instructions of the extended-translation facility 3 are added.

Summary of Changes in Third Edition

The third edition of this publication differs from the previous edition principally by containing the definitions of the DAT-enhancement, HFP-multiply-add/subtract, and long-displacement facilities and the message-security assist. The third edition contains minor clarifications and corrections and also the following significant changes relative to the previous edition:

- In Chapter 3, “Storage”:
 - Clarifications are added to the description of dynamic-address-translation process.
 - The primary address-space-control element (ASCE) in control register 1 is an attaching ASCE even when the CPU is in the home-space mode, and the home ASCE in control register 13 is an attaching ASCE even when the CPU is in the secondary-space mode.
 - The illustration of the PER code in Figure 3-13 is corrected.
- In Chapter 4, “Control”:
 - The relationships between ETR time (TOD-clock time), UTC, and International Atomic Time are described in programming note 3 on page 4-41.
 - Code 0 of the SIGNAL PROCESSOR set-architecture order, and also a CPU reset due to activation of the load normal key, are changed to save the current z/Architecture PSW when switching to the ESA/390 architectural mode. Also, code 2 of the order is added, and this restores, for CPUs other than the one executing SIGNAL PROCESSOR, the saved PSW when switching to the z/Architecture architectural mode, provided that the saved PSW has not been set to all zeros by certain resets.
- In Chapter 5, “Program Execution”:
 - The I format, the RI format with a M_1 operand, and the SS format with the I_3 operand are added.
 - The RSY, RXY, and SIY instruction formats are added, and the RSE format is deleted.

(All instructions that were of format RSE are now referred to as being of format RSY.)

- The formation of an operand address using the 20-bit signed displacement of instructions of formats RSY, RXY, and SIY is described.
- The results when a PER instruction-fetching event occurs along with certain exceptions or exception conditions are clarified. See “Indication of PER Events Concurrently with Other Interruption Conditions” on page 4-33
- The fetch of the address-space-control element from the ASN-second-table entry during access-register translation is doubleword concurrent as observed by other CPUs.
- The change bit is not necessarily set to one currently with the related storage reference, as observed by other CPUs; it may be set to one before or after the reference, within certain limits. See “Storage-Key Accesses” on page 5-91 for a detailed description of when the change bit is set.
- The five instructions of the message-security assist are added to the list of instructions having multiple-access references.
- In Chapter 6, “Interruptions,” the list of conditions causing a specification exception to be recognized is extended to include those caused by the message-security assist instructions.
- In Chapter 7, “General Instructions”:
 - Thirty-nine instructions provided by the long-displacement facility are added. With the exception of the new LOAD BYTE instruction, the instructions added by the long-displacement facility have names and functions that are the same as existing instructions (but the mnemonics and opcodes are new). The new instructions are of formats RSY, RXY, and SIY and have a 20-bit signed displacement instead of a 12-bit unsigned displacement.
 - All previously existing format-RSE and format-RXE instructions are changed to be of formats RSY and RXY, respectively, by use of a previously unused byte in the instructions. These changes are not marked by a bar in the margin.
 - Five instructions provided by the message-security assist are added.
 - The instruction format of SUPERVISOR CALL is changed to I.
- In Chapter 9, “Floating-Point Overview and Support Instructions,” four instructions provided by the long-displacement facility are added. These are the LOAD (long and short) and STORE (long and short) instructions.
- In Chapter 10, “Control Instructions”:
 - The COMPARE AND SWAP AND PURGE (CSPG) and INVALIDATE DAT TABLE ENTRY instructions provided by the DAT-enhancement facility are added. CSPG operates on a doubleword operand in storage.
 - The definition of LOAD ADDRESS SPACE PARAMETERS is clarified.
 - The LOAD REAL ADDRESS (LRAY) instruction provided by the long-displacement facility is added.
 - All previously existing format-RSE instructions are changed to be of format RSY by use of a previously unused byte in the instructions. These changes are not marked by a bar in the margin.
 - The description of the bits set by STORE FACILITY LIST is clarified, and new bits are assigned.
- In Chapter 14, “I/O Instructions”:
 - The definition of MODIFY SUBCHANNEL is modified.
 - The definition of SET CHANNEL MONITOR is modified.
- In Chapter 15, “Basic I/O Functions,” the following changes are made to the subchannel-information-block (SCHIB):
 - Bit 29 of word 6 of the path-management-control word (PMCW) is defined as the measurement-block-format control.
 - Bit 30 of word 6 of the PMCW is defined as the extended-measurement-word-mode enablement bit.
 - The definition of words 10-11 (words 0-1 of the model-dependent area) are changed to

contain a measurement-block address, when the extended-I/O-measurement-block facility is installed.

- In Chapter 16, “I/O Interruptions”, the interruption-response block (IRB) is extended to include the extended-measurement word.
- In Chapter 17, “I/O Support Functions”:
 - The requirement that the measurement block be updated when secondary status is accepted is clarified.
 - The extended-measurement-block facility is added.
 - The extended-measurement-word facility is added.
- In Chapter 18, “Hexadecimal-Floating-Point Instructions,” the MULTIPLY AND ADD (four instructions) and MULTIPLY AND SUBTRACT (four instructions) instructions provided by the HFP-multiply-add/subtract facility are added.

The above changes may affect other chapters besides the ones listed. All technical changes to the text or to an illustration are indicated by a vertical line to the left of the change.

Summary of Changes in Second Edition

The second edition of this publication differs from the previous edition mainly by containing clarifications and corrections. The significant changes are as follows:

- In Chapter 1, “Introduction”:
 - Summaries of DIVIDE LOGICAL and MULTIPLY LOGICAL, TEST ADDRESSING MODE, the set-architecture order of SIGNAL PROCESSOR, and STORE FACILITY LIST are added or improved.
 - An extensive summary of the input/output enhancements placed in z/Architecture is added.
- In Chapter 3, “Storage”:
 - Definitions of absolute locations 0-23 are deleted since they pertain only to an ESA/390 initial program load.
 - The definition of real locations 200-203, stored in by STORE FACILITY LIST, is corrected to state that bit 16 indicates the extended-translation facility 2.
- In Chapter 4, “Control,” a description of unassigned fields in the PSW is corrected to state that bit 4 is unassigned and bit 31 is assigned.
- In Chapter 5, “Program Execution,” the RSL format and an RIL format with an M_1 field are added.
- In Chapter 7, “General Instructions”:
 - The definition of BRANCH AND SET MODE is corrected to state that bit 63 of the R_1 general register remains unchanged in the 24-bit or 31-bit addressing mode; the bit is not set to zero.
 - The definitions of PACK ASCII, PACK UNICODE, UNPACK ASCII, and UNPACK UNICODE are clarified.
 - It is clarified that the following instructions perform multiple-access references to their storage operands:
 - CHECKSUM
 - COMPARE AND FORM CODEWORD
 - CONVERT UNICODE TO UTF-8
 - CONVERT UTF-8 TO UNICODE
 - It is clarified that the following instructions do not necessarily process their storage operands left to right as observed by other CPUs: MOVE LONG, MOVE LONG EXTENDED, and MOVE LONG UNICODE. Special padding characters of MOVE LONG and MOVE LONG EXTENDED specify whether left-to-right processing should be performed, as observed by other CPUs, and whether the data being moved should or should not be placed in the cache for availability for subsequent processing.
- In Chapter 10, “Control Instructions,” it is clarified that the following instructions perform multiple-access references to their storage operands:
 - LOAD ADDRESS SPACE PARAMETERS
 - RESUME PROGRAM

– STORE SYSTEM INFORMATION

Chapters 13-17 contain many clarifying changes, all indicated by a vertical line in the margin, in addition to the significant changes listed below.

- In Chapter 13, “I/O Overview,” statements about the suspend flag in a CCW are clarified to describe the flag being specified as a one and being valid because of a one value of the suspend control in the associated ORB.
- In Chapter 14, “I/O Instructions,” the results of MODIFY SUBCHANNEL when the device-number-valid bit at the designated subchannel is zero are corrected.
- In Chapter 15, “Basic I/O Functions”:
 - It is clarified that unlimited prefetching of data and IDAWs associated with the current and prefetched CCWs is allowed independent of the value of the prefetch control in the associated ORB.
 - A specified control-unit-priority number is ignored if the channel-subsystem-I/O-priority facility is not operational due to an operator action.
- It is clarified that address-limit checking applies to data locations and not to locations containing a CCW or IDAW.
- In Chapter 16, “I/O Interruptions,” the form of the address stored in the failing-storage-address field is described in terms of the format-2-IDAW control instead of an addressing mode.
- In Chapter 17, “I/O Support Functions”:
 - The introduction to the channel-subsystem monitoring facilities is clarified.
 - References to the measurement block by the measurement-block-update facility are single-access references and appear to be word concurrent as observed by CPUs. They do not appear to be block concurrent.
 - The description of the channel-subsystem-I/O-priority facility is corrected by including mention of control-unit priority for fibre-channel-attached control units.

The above changes may affect other chapters besides the ones listed.

Chapter 1. Introduction

Highlights of Original z/Architecture	1-2	HFP-Unnormalized-Extensions Facility	1-13
General Instructions for 64-Bit Integers	1-2	High-Word Facility	1-13
Other New General Instructions	1-3	IEEE-Exception-Simulation Facility	1-14
Floating-Point Instructions	1-4	Interlocked-Access Facility	1-14
Control Instructions	1-4	IPTE-Range Facility	1-14
Trimodal Addressing	1-5	List-Directed Initial Program Load	1-14
Modal Instructions	1-5	Load/Store-on-Condition Facility	1-14
Effects on Bits 0-31 of a General Register	1-5	Long-Displacement Facility	1-14
Input/Output	1-6	Message-Security Assist	1-15
Additions to z/Architecture	1-7	Message-Security-Assist Extension 1	1-15
ASN-and-LX-Reuse Facility	1-7	Message-Security-Assist Extension 2	1-15
CMPSC-Enhancement Facility	1-7	Message-Security-Assist Extension 3	1-15
Compare-and-Swap-and-Store Facility	1-7	Message-Security-Assist Extension 4	1-16
Compare-and-Swap-and-Store Facility 2	1-7	Modified CCW Indirect Data Addressing Facility	1-17
Conditional-SSKE Facility	1-7	Move-With-Optional-Specifications Facility	1-18
Configuration-Topology Facility	1-7	Multiple-Subchannel-Set Facility	1-18
DAT-Enhancement Facility 1	1-8	Nonquiescing Key-Setting Facility	1-18
DAT-Enhancement Facility 2	1-8	Parsing-Enhancement Facility	1-18
Decimal-Floating-Point Facility	1-8	PER-3 Facility	1-18
Decimal-Floating-Point-Rounding Facility	1-8	PFPO Facility	1-18
Distinct-Operands Facility	1-8	Population-Count Facility	1-19
Enhanced-DAT Facility	1-8	The Reset-Reference-Bits-Multiple Facility	1-19
Enhanced-Monitor Facility	1-9	Restore-Subchannel Facility	1-19
ETF2-Enhancement Facility	1-9	Server-Time-Protocol Facility	1-19
ETF3-Enhancement Facility	1-9	Store-Clock-Fast Facility	1-19
Execute-Extensions Facility	1-10	Store-Facility-List-Extended Facility	1-19
Extended-Immediate Facility	1-10	TOD-Clock-Steering Facility	1-19
Extended-I/O-Measurement-Block Facility	1-10	The ESA/390 Base	1-19
Extended-I/O-Measurement-Word Facility	1-10	The ESA/370 and 370-XA Base	1-25
Extended-Translation Facility 2	1-10	System Program	1-26
Extended-Translation Facility 3	1-11	Compatibility	1-27
Extract-CPU-Time Facility	1-11	Compatibility among z/Architecture Systems	1-27
Fast-BCR-Serialization Facility	1-11	Compatibility between z/Architecture and ESA/390	1-27
Fibre-Channel Extensions (FCX)	1-11	Control-Program Compatibility	1-27
Floating-Point Extension Facility	1-12	Problem-State Compatibility	1-27
Floating-Point-Support-Sign- Handling Facility	1-12	Availability	1-28
FPR-GR-Transfer Facility	1-13		
General-Instructions-Extension Facility	1-13		
HFP Multiply-and-Add/Subtract Facility	1-13		

This publication provides, for reference purposes, a detailed description of z/Architecture.™

The architecture of a system defines its attributes as seen by the programmer, that is, the conceptual structure and functional behavior of the machine, as distinct from the organization of the data flow, the logical design, the physical design, and the performance

of any particular implementation. Several dissimilar machine implementations may conform to a single architecture. When the execution of a set of programs on different machine implementations produces the results that are defined by a single architecture, the implementations are considered to be compatible for those programs.

Highlights of Original z/Architecture

z/Architecture is the next step in the evolution from the System/360 to the System/370™, System/370 extended architecture (370-XA), Enterprise Systems Architecture/370™ (ESA/370™), and Enterprise Systems Architecture/390® (ESA/390™). z/Architecture includes all of the facilities of ESA/390 except for the asynchronous-pageout, asynchronous-data-mover, program-call-fast, and vector facilities. z/Architecture also provides significant extensions, as follows:

- Sixty-four-bit general registers and control registers.
- A 64-bit addressing mode, in addition to the 24-bit and 31-bit addressing modes of ESA/390, which are carried forward to z/Architecture.

Both operand addresses and instruction addresses can be 64-bit addresses. The program-status word (PSW) is expanded to 16 bytes to contain the larger instruction address. The PSW also contains a newly assigned bit that specifies the 64-bit addressing mode.

- Up to three additional levels of dynamic-address-translation (DAT) tables, called region tables, for translating 64-bit virtual addresses.

A virtual address space may be specified either by a segment-table designation as in ESA/390 or by a region-table designation, and either of these types of designation is called an address-space-control element (ASCE). An ASCE may alternatively be a real-space designation that causes virtual addresses to be treated simply as real addresses without the use of DAT tables.

- An 8K-byte prefix area for containing larger old and new PSWs and register save areas.
- A SIGNAL PROCESSOR order for switching between the ESA/390 and z/Architecture architectural modes.

Initial program loading sets the ESA/390 architectural mode. The new SIGNAL PROCESSOR order then can be used to set the z/Architecture mode or to return from z/Architecture to ESA/390. This order causes all CPUs in the con-

figuration always to be in the same architectural mode.

- Many new instructions, many of which operate on 64-bit binary integers

Some of the new instructions that do not operate on 64-bit binary integers have also been added to ESA/390.

All of the ESA/390 instructions, except for those of the four facilities named above, are included in z/Architecture.

The bit positions of the general registers and control registers of z/Architecture are numbered 0-63. An ESA/390 instruction that operates on bit positions 0-31 of a 32-bit register in ESA/390 operates instead on bit positions 32-63 of a 64-bit register in z/Architecture.

z/Architecture was announced in October, 2000. The remainder of this section summarizes the original contents of z/Architecture. Subsequent additions are described in "Additions to z/Architecture" on page 1-7.

General Instructions for 64-Bit Integers

The 32-bit-binary-integer instructions of ESA/390 have new analogs in z/Architecture that operate on 64-bit binary integers. There are two types of analogs:

- Analogs that use two 64-bit binary integers to produce a 64-bit binary integer. For example, the ESA/390 ADD instruction (A for a storage-to-register operation or AR for a register-to-register operation) has the analogs AG (adds 64 bits from storage to the contents of a 64-bit general register) and AGR (adds the contents of a 64-bit general register to the contents of another 64-bit general register). These analogs are distinguished by having "G" in their mnemonics.
- Analogs that use a 64-bit binary integer and a 32-bit binary integer to produce a 64-bit binary integer. The 32-bit integer is either sign-extended or extended on the left with zeros, depending on whether the operation is signed or unsigned, respectively. For example, the ESA/390 ADD (A or AR) instruction has the analogs AGF (adds 32

bits from storage to the contents of a 64-bit general register) and AGFR (adds the contents of bit positions 32-63 of a 64-bit general register to the contents of another 64-bit general register). These analogs are distinguished by having “GF” in their mnemonics.

Other New General Instructions

The other additional or significantly enhanced general instructions of z/Architecture are highlighted as follows:

- ADD LOGICAL WITH CARRY and SUBTRACT LOGICAL WITH BORROW operate on either 32-bit or 64-bit unsigned binary integers and include a carry or borrow, as represented by the leftmost bit of the two-bit condition code in the PSW, in the computation. This can improve the performance of operating on extended-precision integers (integers longer than 64 bits).
- AND IMMEDIATE and OR IMMEDIATE combine a two-byte immediate operand with any of the two bytes on two-byte boundaries in a 64-bit general register.
- BRANCH AND SAVE AND SET MODE and BRANCH AND SET MODE are enhanced so that they set bit 63 of the R_1 general register to one if the current addressing mode is the 64-bit mode, and they set the 64-bit addressing mode if bit 63 of the R_2 general register is one. This allows “pointer-directed” linkages between programs in different addressing modes, including any of the 24-bit, 31-bit, and 64-bit modes.
- BRANCH RELATIVE AND SAVE LONG and BRANCH RELATIVE ON CONDITION LONG are like the BRANCH RELATIVE AND SAVE and BRANCH RELATIVE ON CONDITION instructions of ESA/390 except that the new instructions use a 32-bit immediate field. This increases the target range available through relative branching.
- COMPARE AND FORM CODEWORD is enhanced so that, in the 64-bit addressing mode, the comparison unit is six bytes instead of two and the resulting codeword is eight bytes instead of four. UPDATE TREE is enhanced so that, in the 64-bit addressing mode, a node is 16 bytes instead of eight and the codeword in a node is eight bytes instead of four. This improves the performance of sorting records having long keys.
- DIVIDE LOGICAL uses a 64-bit or 128-bit unsigned binary dividend and a 32-bit or 64-bit unsigned binary divisor, respectively, to produce a 32-bit or 64-bit quotient and remainder, respectively. MULTIPLY LOGICAL uses a 32-bit or 64-bit unsigned binary multiplicand and multiplier to produce a 64-bit or 128-bit product, respectively.
- DIVIDE SINGLE divides a 64-bit dividend by a 32-bit or 64-bit divisor and produces a 64-bit quotient and remainder. MULTIPLY SINGLE is enhanced so it can multiply a 64-bit multiplicand by a 32-bit or 64-bit multiplier and produce a 64-bit product.
- EXTRACT PSW extracts bits 0-63 of the current PSW to allow determination of the current machine state, for example, determination of whether the CPU is in the problem state or the supervisor state.
- INSERT IMMEDIATE inserts a two-byte immediate operand into a 64-bit general register on any of the two-byte boundaries in the register. LOAD LOGICAL IMMEDIATE does the same and also clears the remainder of the register.
- LOAD ADDRESS RELATIVE LONG forms an address relative to the current (unupdated) instruction address by means of a signed 32-bit immediate field.
- LOAD LOGICAL THIRTY ONE BITS places the rightmost 31 bits of either a general register or a word in storage, with 33 zeros appended on the left, in a general register.
- LOAD MULTIPLE DISJOINT loads the leftmost 32 bits of each register in a range of general registers from one area in storage and the rightmost 32 bits of each of those registers from another area in storage. This is for use in place of a LOAD MULTIPLE HIGH instruction and a 32-bit LOAD MULTIPLE instruction when one of the storage areas is addressed by one of the registers loaded.
- LOAD MULTIPLE HIGH and STORE MULTIPLE HIGH load or store the leftmost 32 bits of each register in a range of general registers, allowing augmentation of existing programs that load or store the rightmost 32 bits by means of LOAD MULTIPLE and STORE MULTIPLE. (Sixty-four-bit forms of LOAD MULTIPLE and STORE MULTIPLE also are provided.)

- **LOAD PAIR FROM QUADWORD** and **STORE PAIR TO QUADWORD** operate between an even-odd pair of 64-bit general registers and a quadword in storage (16 bytes aligned on a 16-byte boundary). These instructions provide quadword consistency (all bytes appear to be loaded or stored concurrently in a multiple-CPU system). (Only the 64-bit form of **COMPARE DOUBLE** AND **SWAP** also provides quadword consistency.)
- **LOAD REVERSED** and **STORE REVERSED** load or store a two-byte, four-byte, or eight-byte unit in storage with the left-to-right sequence of the bytes reversed. **LOAD REVERSED** also can move a four-byte or eight-byte unit between two general registers. These operations allow conversion between “little-endian” and “big-endian” formats.
- **PERFORM LOCKED OPERATION** is enhanced with two more sets of function codes, with each set providing six different operations. One of the additional sets provides operations on 64-bit operands in 64-bit general registers, and the other provides operations on 128-bit operands in a parameter list.
- **ROTATE LEFT SINGLE LOGICAL** obtains 32 bits or 64 bits from a general register, rotates them (the leftmost bit replaces the rightmost bit), and places the result in another general register (a nondestructive rotate).
- **SET ADDRESSING MODE** can set any of the 24-bit, 31-bit, and 64-bit addressing modes.
- **SHIFT LEFT SINGLE**, **SHIFT LEFT SINGLE LOGICAL**, **SHIFT RIGHT SINGLE**, and **SHIFT RIGHT SINGLE LOGICAL** are enhanced with 64-bit forms that obtain the source operand from one general register and place the result operand in another general register (a nondestructive shift).
- **TEST ADDRESSING MODE** sets the condition code to indicate whether bits 31 and 32 of the current PSW specify the 24-bit, 31-bit, or 64-bit addressing mode.
- **TEST UNDER MASK HIGH** and **TEST UNDER MASK LOW**, which are ESA/390 instructions, are given the alternative name **TEST UNDER MASK**, and two additional forms are added so that a two-byte immediate operand can be used to test

the bits of two bytes located on any of the two-byte boundaries in a 64-bit general register. (The ESA/390 instruction **TEST UNDER MASK**, which uses a one-byte immediate operand to test a byte in storage, continues to be provided.)

Floating-Point Instructions

The z/Architecture floating-point instructions are the same as in ESA/390 except that instructions are added for converting between 64-bit signed binary integers and either hexadecimal or binary floating-point data. These new instructions have “G” in their mnemonics.

Control Instructions

The new or enhanced control instructions of z/Architecture are highlighted as follows:

- **EXTRACT AND SET EXTENDED AUTHORITY** is a privileged instruction for changing the extended authorization index in a control register. This enables real-space designations to be used more efficiently by means of access lists.
- **EXTRACT STACKED REGISTERS** is enhanced to extract optionally all 64 bits of the contents of one or more saved general registers.
- **EXTRACT STACKED STATE** is enhanced to extract optionally the entire contents of the saved PSW, including a 64-bit instruction address.
- **LOAD CONTROL** and **STORE CONTROL** are enhanced for operating optionally on 64-bit control registers.
- **LOAD PSW** uses an eight-byte storage operand as in ESA/390 and expands this operand to a 16-byte z/Architecture PSW.
- **LOAD PSW EXTENDED** directly loads a 16-byte PSW.
- **LOAD REAL ADDRESS** in its ESA/390 form and in the 24-bit or 31-bit addressing mode operates as in ESA/390 if the translation is successful and the obtained real address has a value less than 2G bytes. **LOAD REAL ADDRESS** in its ESA/390 form and in the 64-bit addressing mode, or in its enhanced z/Architecture form in any addressing mode, loads a 64-bit real address.

- **LOAD USING REAL ADDRESS** and **STORE USING REAL ADDRESS** are enhanced to have optionally 64-bit operands.
- **SIGNAL PROCESSOR** has a new order that can be used to switch all CPUs in the configuration either from the ESA/390 architectural mode to the z/Architecture architectural mode or from z/Architecture to ESA/390. (A system that is to operate using z/Architecture must first be IPLed in the ESA/390 mode.)
- **STORE FACILITY LIST** is a privileged instruction that stores at real location 200 an indication of whether z/Architecture is installed and of whether it is active. This instruction is added also to ESA/390 and also stores an indication of whether the new z/Architecture instructions that have been added to ESA/390 are available. Real location 200 has previously contained all zeros in most systems and normally can be examined by a problem-state program whether or not **STORE FACILITY LIST** is installed. The information stored at real location 200 also indicates whether the extended-translation facility 2 is installed.
- **STORE REAL ADDRESS** is like **LOAD REAL ADDRESS** except that **STORE REAL ADDRESS** stores the resulting address instead of placing it in a register.
- **TRACE** is enhanced to record optionally the contents of 64-bit general registers.

Trimodal Addressing

“Trimodal addressing” refers to the ability to switch between the 24-bit, 31-bit, and 64-bit addressing modes. This switching can be done by means of:

- The old instructions **BRANCH AND SAVE AND SET MODE** and **BRANCH AND SET MODE**. Both of these instructions set the 64-bit addressing mode if bit 63 of the R_2 general register is one. If bit 63 is zero, the instructions set the 24-bit or 31-bit addressing mode if bit 32 of the register is zero or one, respectively.
- The new instruction **SET ADDRESSING MODE** (SAM24, SAM31, and SAM64). The instruction sets the 24-bit, 31-bit, or 64-bit addressing mode as determined by the operation code.

Modal Instructions

Trimodal addressing affects the general instructions only in the manner in which logical storage addresses are handled, except as follows.

- The instructions **BRANCH AND LINK**, **BRANCH AND SAVE**, **BRANCH AND SAVE AND SET MODE**, **BRANCH AND SET MODE**, and **BRANCH RELATIVE AND SAVE** place information in bit positions 32-39 of general register R_1 as in ESA/390 in the 24-bit or 31-bit addressing mode or place address bits in those bit positions in the 64-bit addressing mode. The new instruction **BRANCH RELATIVE AND SAVE LONG** does the same.
- The instructions **BRANCH AND SAVE AND SET MODE** and **BRANCH AND SET MODE** place a one in bit position 63 of general register R_1 in the 64-bit addressing mode. In the 24-bit or 31-bit mode, **BRANCH AND SAVE AND SET MODE** sets bit 63 to zero, and **BRANCH AND SET MODE** leaves it unchanged.
- Certain instructions leave bits 0-31 of a general register unchanged in the 24-bit or 31-bit addressing mode but place or update address or length information in them in the 64-bit addressing mode. These are listed in programming note 1 on page 7-9 and are sometimes called modal instructions.

Effects on Bits 0-31 of a General Register

Bits 0-31 of general registers are changed by two types of instructions. The first type is a modal instruction (see the preceding section) when the instruction is executed in the 64-bit addressing mode. The second type is an instruction having, independent of the addressing mode, either a 64-bit result operand in a single general register or a 128-bit result operand in an even-odd general-register pair.

Most of the instructions of the second type are indicated by a “G,” either alone or in “GF,” in their mnemonics. The other instructions that change or may change bits 0-31 of a general register regardless of the current addressing mode are listed in programming note 2 on page 7-9. All of the instructions of the second type are sometimes referred to as “G-type” instructions.

If a program is not executed in the 64-bit addressing mode and does not contain a G-type instruction, it cannot change bits 0-31 of any general register.

Input/Output

Additional I/O functions and facilities are provided when z/Architecture is installed. They are provided in both the ESA/390 and the z/Architecture architectural mode and are as follows:

- Indirect data addressing is enhanced by the provision of a doubleword format-2 IDAW that is intended to allow operations on data at or above the 2G-byte absolute-address boundary in z/Architecture. The previously existing IDAW, a word containing a 31-bit address, is now called a format-1 IDAW. The format-2 IDAW contains a 64-bit address. A bit in the operation-request block (ORB) associated with a channel program specifies whether the program uses format-1 or format-2 IDAWs. A further enhancement is the ability of all format-2 IDAWs of a channel program to specify either 2K-byte or 4K-byte data blocks, as determined by another bit in the ORB. The use of 4K-byte blocks improves the efficiency of data transfers.
 - The *FICON[®]-channel facility* provides the capabilities of attaching FICON-I/O-interface and FICON-converted-I/O-interface channel paths and of fully utilizing these channel-path types. FICON channel paths can significantly enhance overall data throughput by providing increased data-transfer rates in comparison to ESCON channel paths and by allowing multiple commands and associated data to be “streamed” to control units, thus further improving performance. The facility supports the following additional control mechanisms:
 - The modification-control bit in the ORB allows the program to optimize the performance of FICON channel paths when dynamically modifying channel programs.
 - The synchronization-control bit in the ORB ensures data integrity along with maximum channel-path performance by delaying the execution of a write command until the completion of an immediately preceding read command when performing unlimited prefetching of CCWs and when the data to be written may be the data read.
 - The streaming-mode-control bit in the ORB allows the program to prevent command streaming in cases that require such prevention.
 - The secondary-CCW-address field in the extended-status word assists in the recovery of channel programs that terminate abnormally when command streaming to a control unit is being performed. The field identifies a CCW that failed at the control unit.
 - The *ORB-extension facility* expands the size of the ORB from three words to eight words. This makes fields available for use by the channel-subsystem-I/O-priority facility.
 - The *channel-subsystem-I/O-priority facility* allows the program to establish a priority relationship among subchannels that have pending I/O operations. The priority relationship specifies the order in which I/O operations are initiated by the channel subsystem. Additionally, for fibre-channel-attached control units, the facility allows the program to specify the priority in which I/O operations pending at the control unit are performed.
- The input/output enhancements are further highlighted below by describing how they affect the I/O chapters.
- In Chapter 13, “I/O Overview,” FICON and FICON-converted I/O interfaces and the frame-multiplex mode are introduced.
 - In Chapter 14, “I/O Instructions”:
 - The CANCEL SUBCHANNEL instruction is described.
 - TEST PENDING INTERRUPTION, when the second-operand address is zero, stores a three-word I/O-interruption code at real locations 184-195. The new third word contains an interruption-identification word that further identifies the source of the I/O interruption.
 - In Chapter 15, “Basic I/O Functions”:
 - The ORB is extended to eight words and newly contains a streaming-mode control, modification control, synchronization control, format-2-IDAW control, 2K-IDAW control, ORB-extension control, channel-subsystem priority, and control-unit priority.
 - A doubleword format-2 IDAW and 4K-byte data blocks optionally designated by format-2 IDAWs are added.
 - In Chapter 16, “I/O Interruptions”:

- A secondary-CCW-address-validity bit and failing-storage-address-format bit are added to the extended-report word.
 - A two-word failing-storage address and a secondary-CCW address are added to the format-0 extended-status word.
- In Chapter 17, “I/O Support Functions”:
 - Control-unit-defer time is added. This has an effect on the device-connect time and device-disconnect time in the measurement block.
 - References to the measurement block by the measurement-block-update facility are single-access references and appear to be word concurrent as observed by CPUs.
 - Device-active-only time is added to the measurement block.
 - The channel-subsystem-I/O-priority facility, providing channel-subsystem priority and control-unit priority, is added.

Additions to z/Architecture

z/Architecture was announced in October, 2000. Any extension added subsequently is summarized below and has the date of its announcement at the end of its summary.

ASN-and-LX-Reuse Facility

The *ASN-and-LX-reuse facility* may be available on a model implementing z/Architecture. The facility provides the means by which an address-space number (ASN) that is used in certain space-switching linkage instructions may be safely reused. The facility also adds a 32-bit program-call (PC) number, and it also provides the means by which the linkage index that is used in PC-number translation may be safely reused. The facility provides the following instructions:

```
EXTRACT PRIMARY ASN AND INSTANCE
EXTRACT SECONDARY ASN AND INSTANCE
PROGRAM TRANSFER WITH INSTANCE
SET SECONDARY ASN WITH INSTANCE
```

(May, 2004)

CMPSC-Enhancement Facility

The *CMPSC-enhancement facility* may be available on a model implementing z/Architecture. The facility provides performance improvements for the COMPRESSION CALL instruction. (August, 2010)

Compare-and-Swap-and-Store Facility

The *compare-and-swap-and-store facility* may be available on a model implementing z/Architecture. The facility performs compare-and-swap operations on 4- or 8-byte operands using interlocked update. If the operands are equal, a subsequent store operation of 1, 2, 4, or 8 bytes is performed. The facility provides the COMPARE AND SWAP AND STORE instruction. (April, 2007)

Compare-and-Swap-and-Store Facility 2

The *compare-and-swap-and-store facility 2* may be available on a model implementing z/Architecture. The facility extends the compare-and-swap-and-store facility by providing a 16-byte compare-and-swap function and a 16-byte store operation. (February, 2008)

Conditional-SSKE Facility

The *conditional-SSKE facility* may be available on a model implementing z/Architecture. The facility provides performance improvements for the SET STORAGE KEY EXTENDED instruction. (April, 2007)

Configuration-Topology Facility

The *configuration-topology facility* may be available on a model implementing z/Architecture. The facility provides additional topology awareness to the program such that certain optimizations can be performed to improve cache hit ratios and thereby improve overall performance. The facility provides:

- The PERFORM TOPOLOGY FUNCTION (PTF) control instruction.

- A new system-information block (SYSIB 15.1.2) stored by the STORE SYSTEM INFORMATION instruction.

(February, 2008)

DAT-Enhancement Facility 1

The *DAT-enhancement facility 1* may be available on a model implementing z/Architecture. The facility provides the following instructions:

COMPARE AND SWAP AND PURGE (CSPG)
INVALIDATE DAT TABLE ENTRY

COMPARE AND SWAP AND PURGE (CSPG) provides function similar to that of COMPARE AND SWAP AND PURGE (CSP), but CSPG has 64-bit operands whereas CSP has 32-bit operands.

INVALIDATE DAT TABLE ENTRY provides the means by which one or more region-table and segment-table entries in storage may be invalidated, and the corresponding TLB entries may be purged. The instruction provides an invalidation-and-clearing operation which invalidates and clears entries based on a specified virtual address, or a clear-by-ASCE operation which clears TLB entries based on the specified ASCE. (June, 2003)

DAT-Enhancement Facility 2

The *DAT-enhancement facility 2* may be available on a model implementing z/Architecture. When the DAT-enhancement facility 2 is installed, the LOAD PAGE-TABLE-ENTRY ADDRESS instruction is available. Given a virtual address, the LOAD PAGE-TABLE-ENTRY ADDRESS instruction returns the 64-bit real address of the corresponding page-table entry. The address-space-control mode used by the dynamic-address-translation process is specified in the M_4 field of the instruction. (September, 2005)

Decimal-Floating-Point Facility

The *decimal-floating-point facility* supports three decimal-floating-point (DFP) data formats and provides 54 new instructions to operate on data in these formats. The formats: 32-bit (short), 64-bit (long), and 128-bit (extended), were developed in collaboration with the IEEE floating-point working group. (April, 2007)

Decimal-Floating-Point-Rounding Facility

The *decimal-floating-point-rounding facility* provides a 3-bit DFP rounding mode field in the floating-point control (FPC) register, and the instruction SET DFP ROUNDING MODE, which may be used to set this field. The DFP rounding mode can specify any of eight rounding methods, including the five required by the IEEE floating-point working group.

The DFP rounding mode is used by PFPO and the decimal-floating-point instructions. (April, 2007)

Distinct-Operands Facility

The *distinct-operands facility* may be available on a model implementing z/Architecture. The facility provides alternate forms of selected arithmetic and logical instructions in which the result register may be different from either of the source registers. The facility provides alternate forms for the following instructions.

- ADD
- ADD IMMEDIATE
- ADD LOGICAL
- ADD LOGICAL WITH SIGNED IMMEDIATE
- AND
- EXCLUSIVE OR
- OR
- SHIFT LEFT SINGLE
- SHIFT LEFT SINGLE LOGICAL
- SHIFT RIGHT SINGLE
- SHIFT RIGHT SINGLE LOGICAL
- SUBTRACT
- SUBTRACT LOGICAL

(August, 2010)

Enhanced-DAT Facility

The *enhanced-DAT facility* may be available on models implementing z/Architecture. When the facility is installed and enabled, DAT translation may produce either a page-frame real address or a segment-frame absolute address, determined by the STE-format control in the segment-table entry.

When the facility is installed in a configuration, a new bit in control register 0 enables the facility.

Note: The term *enhanced DAT applies* is used pervasively in this document to describe the condition of when the enhanced-DAT facility is installed in the configuration and enabled by control register 0. See “Enhanced-DAT Terminology:” on page 3-37 for details on this terminology.

When enhanced DAT applies, the following additional function is available in the DAT process:

- A DAT-protection bit is added to region-table entries, providing function similar to the DAT-protection bits in the segment- and page-table entries.
- A STE-format control is added to the segment-table entry. When the STE-format control is zero, DAT proceeds as is currently defined, except that a change-recording override in the page-table entry indicates whether setting of the change bit may be bypassed for the page.
- When the STE-format control is one, the segment-table entry also contains the following:
 - A segment-frame absolute address (rather than a page-table origin) specifying the absolute storage location of the 1M-byte block.
 - Access-control bits and a fetch-protection bit which optionally may be used in lieu of the corresponding bits in the segment’s individual storage keys
 - A bit which determines the validity of the access-control bits and a fetch-protection bit in the segment-table entry
 - A change-recording override which indicates whether setting of the change bit may be bypassed in the segment’s individual storage keys.

The facility adds the PERFORM FRAME MANAGEMENT FUNCTION control instruction. The facility includes enhancements or changes to the following control instructions:

- LOAD PAGE-TABLE-ENTRY ADDRESS
- MOVE PAGE
- SET STORAGE KEY EXTENDED

(February, 2008)

Enhanced-Monitor Facility

The *enhanced-monitor facility* may be available on a model implementing z/Architecture. The facility provides the means by which executions of the MONITOR CALL instruction may be counted in CPU-specific counter arrays where each entry in the array represents a different monitor code.

The enhanced-monitor facility is controlled by the enhanced-monitor masks in control register 8. The enhanced-monitor masks work in conjunction with the monitor masks in CR8. Existing programs that use the MONITOR CALL instruction continue to operate compatibly when the enhanced-monitor masks are zero; the enhanced counting operation is only provided when a monitor mask bit *and* its corresponding enhanced-monitor mask bit are both ones. (August, 2010)

ETF2-Enhancement Facility

The *ETF2-enhancement facility* may be available on a model implementing z/Architecture. The facility includes modifications to the following instructions:

- New function is added to the TRANSLATE ONE TO ONE, TRANSLATE ONE TO TWO, TRANSLATE TWO TO ONE, and TRANSLATE TWO TO TWO instructions, allowing the test-character comparison to be bypassed.
- For TRANSLATE TWO TO ONE and TRANSLATE TWO TO TWO, the alignment requirements for the translate table are relaxed.

(September, 2005)

ETF3-Enhancement Facility

The *ETF3-enhancement facility* may be available on a model implementing z/Architecture. The facility provides improved well-formedness checking for the following instructions:

CONVERT UTF-16 TO UTF-32
CONVERT UTF-16 TO UTF-8
CONVERT UTF-8 TO UTF-16
CONVERT UTF-8 TO UTF-32

(September, 2005)

Execute-Extensions Facility

The *execute-extensions facility* may be available on a model implementing z/Architecture. The facility provides the EXECUTE RELATIVE LONG instruction. (February, 2008)

Extended-Immediate Facility

The *extended-immediate facility* may be available on models implementing z/Architecture. The facility provides 32-bit immediate-operand versions of the following instructions:

ADD IMMEDIATE
ADD LOGICAL IMMEDIATE
AND IMMEDIATE
COMPARE IMMEDIATE
COMPARE LOGICAL IMMEDIATE
EXCLUSIVE OR IMMEDIATE
INSERT IMMEDIATE
LOAD IMMEDIATE
LOAD LOGICAL IMMEDIATE
OR IMMEDIATE
SUBTRACT LOGICAL IMMEDIATE

Other new instructions added as a part of the extended-immediate facility include the following:

FIND LEFTMOST ONE
LOAD AND TEST (32-bit and 64-bit RXY format)
LOAD BYTE (RRE format)
LOAD HALFWORD (RRE format)
LOAD LOGICAL CHARACTER (64-bit RRE format, and 32-bit RXY and RRE formats)
LOAD LOGICAL HALFWORD (64-bit RRE format, and 32-bit RXY and RRE formats)

(September, 2005)

Extended-I/O-Measurement-Block Facility

The *extended-I/O-measurement-block facility* may be available on models implementing z/Architecture. The facility includes the following features:

- A new format of the channel-measurement block. The new measurement block, termed a format-1 channel-measurement block, is expanded to 64 bytes and is addressed using a separate mea-

surement-block address for each subchannel. The new measurement-block format provides additional measurement information and the flexibility to store the measurement blocks in non-contiguous, real storage.

- The previously existing channel-measurement block is termed a format-0 channel-measurement block. A device-busy-time field is added to the format-0 channel-measurement block.

(June, 2003)

Extended-I/O-Measurement-Word Facility

The *extended-I/O-measurement-word facility* may be available on models implementing z/Architecture. The extended-measurement-word (EMW) is an extension to the interruption-response block (IRB) and allows channel-measurement data to be provided on an I/O operation basis. This reduces program overhead by alleviating the previous requirement that the program fetch the measurement block before and after an operation and calculate the difference between the respective measurement data values. (June, 2003)

Extended-Translation Facility 2

The *extended-translation facility 2* may be available on a model implementing z/Architecture. The facility performs operations on double-byte, ASCII, and decimal data. The double-byte data may be Unicode® data — data that uses the binary codes of the Unicode Worldwide Character Standard and enables the use of characters of most of the world's written languages. The facility provides the following instructions:

COMPARE LOGICAL LONG UNICODE
MOVE LONG UNICODE
PACK ASCII
PACK UNICODE
TEST DECIMAL
TRANSLATE ONE TO ONE
TRANSLATE ONE TO TWO
TRANSLATE TWO TO ONE
TRANSLATE TWO TO TWO
UNPACK ASCII
UNPACK UNICODE

The extended-translation facility 2 is called facility 2 since an extended-translation facility, now called facility 1, was introduced in ESA/390. Facility 1 is standard in z/Architecture. Facility 1 provides the instructions:

```
CONVERT UNICODE TO UTF-8
CONVERT UTF-8 TO UNICODE
TRANSLATE EXTENDED
```

For when either or both of facility 1 and facility 2 are not installed on the machine, both facilities are simulated by the MVS CSRUNIC macro instruction, which is provided in OS/390® Release 10 and z/OS®.

OS/390 MVS Assembler Services Reference, GC28-1910-10, contains programming requirements, register information, syntax, return codes, and examples for the CSRUNIC macro instruction.

When CSRUNIC is used, the program exceptions listed in this publication do not cause program interruptions; instead, the exception conditions are indicated by CSRUNIC by means of return codes, as described in GC28-1910-10. (October, 2000)

Extended-Translation Facility 3

The *extended-translation facility 3* may be available on a model implementing z/Architecture. The facility performs operations on Unicode and Unicode-transformation-format (UTF) characters; it also includes a right-to-left TRANSLATE AND TEST operation. The facility provides the following instructions:

```
CONVERT UTF-16 TO UTF-32
CONVERT UTF-32 TO UTF-16
CONVERT UTF-32 TO UTF-8
CONVERT UTF-8 TO UTF-32
SEARCH STRING UNICODE
TRANSLATE AND TEST REVERSE
```

(May, 2004)

Extract-CPU-Time Facility

The *extract-CPU-time facility* may be available on a model implementing z/Architecture. The facility adds the general instruction EXTRACT CPU TIME. This instruction provides an efficient means by which a problem-state program can determine the amount of

CPU time consumed, without requiring a supervisor-state service routine. (April, 2007)

Fast-BCR-Serialization Facility

The *fast-BCR-serialization facility* may be available on a model implementing z/Architecture. When the facility is installed, the execution of BRANCH ON CONDITION with the M₁ field containing 1110 binary and the R₂ field containing 0000 binary causes serialization without checkpoint synchronization. (August, 2010)

Fibre-Channel Extensions (FCX)

The *fibre-channel-extensions (FCX) facility* may be available on models implementing z/Architecture. The facility includes enhancements and performance extensions that provide the program a means to transport lists of commands to an I/O device for processing without the channel overhead of fetching and decoding each command.

Use of the FCX facility may be restricted to devices accessible by certain channel path types.

The FCX facility includes the following:

- Changes to the operation-request block (ORB), interruption-response block (IRB), and subchannel-information block (SCHIB).
- A form of a channel program that consists of a transport-control word (TCW) that designates a transport-command-control block (TCCB) and transport-status block (TSB). The TCCB includes a transport-command area (TCA) which contains a list of up to 30 I/O commands that are in the form of device-command words (DCWs). The TSB contains completion status and other information related to the TCW channel program that is complementary to the completion information contained in the IRB.

A TCW may specify the transfer of either input data or output data.

- The ability to directly or indirectly designate any or all of the TCCB, the input data storage area, and the output data storage area. When a storage area is designated directly, the TCW specifies the location of a single, contiguous block of storage. When a storage area is designated indi-

rectly, the TCW designates the location of a list of one or more transport-indirect-data-address words (TIDAWs). TIDAW lists and the storage area designated by each TIDAW in a list are restricted from crossing 4K-byte boundaries.

- An interrogate operation which may be initiated by CANCEL SUBCHANNEL to determine the state of an I/O operation.

(February, 2009)

Floating-Point Extension Facility

The *floating-point extension facility* may be available on models implementing the DFP facility. The extension includes the following features and instructions:

Additions to Chapter 9, “Floating-Point Overview and Support Instructions:”

- A new floating-point-support instruction, SET BFP ROUNDING MODE (SRNMB), is added.
- A new exception, the quantum exception, is defined for DFP computational operations.
- Bit 5 of the floating-point-control (FPC) register is assigned to the quantum-exception mask
- Bit 13 of the FPC register is assigned to the quantum-exception flag.
- Data-exception code (DXC) 04 (hex) is assigned to the quantum exception.
- DXC 07 (hex) is assigned to the simulated quantum exception.
- The BFP-rounding-mode field in the FPC register is changed to 3 bits to support one additional BFP rounding mode, round to prepare for shorter precision.
- One new value of the effective rounding method field is assigned to support the round to prepare for shorter precision rounding method for CONVERT HFP TO BFP.
- For PFPO with a DFP result, bit 58 of general register 0 is assigned to be the DFP quantum-permission control.

Additions to Chapter 19, “Binary-Floating-Point Instructions:”

- The following new BFP instructions are added:
 - CONVERT FROM LOGICAL (CXLFBF, CDLFBF, CELFBF, CXLGBF, CDLGBF, and CELGBF).

- CONVERT TO LOGICAL (CLFXBR, CLFDBR, CLFEBR, CLGXBR, CLGDBR, and CLGEBR)
- One new value of the effective rounding method field is assigned to support the round to prepare for shorter precision rounding method for CONVERT TO FIXED, DIVIDE TO INTEGER, and LOAD FP INTEGER
- An IEEE-inexact-exception control (XxC) is added to CONVERT TO FIXED and LOAD FP INTEGER.
- An effective rounding method field and an IEEE-inexact-exception control (XxC) are added to CONVERT FROM FIXED and LOAD ROUNDED.

Additions to Chapter 20, “Decimal-Floating-Point Instructions:”

- The following new DFP instructions are added:
 - CONVERT FROM FIXED (CXFTR, CDFTR)
 - CONVERT FROM LOGICAL (CXLGTR, CDLGTR, CXLFTR, CDLFTR)
 - CONVERT TO FIXED (CFXTR, CFDTR)
 - CONVERT TO LOGICAL (CLGXTR, CLGDTR, CLFXTR, CLFDTR)
- All reserved values in the effective rounding method field and a quantum-exception control (XqC) are assigned for LOAD FP INTEGER, LOAD ROUNDED, QUANTIZE, and REROUND.
- All reserved values in the effective rounding method field are assigned for CONVERT TO FIXED.
- An IEEE-inexact-exception control (XxC) is added to CONVERT TO FIXED and LOAD ROUNDED.
- An effective rounding method field and a quantum-exception control (XqC) are added to ADD, DIVIDE, MULTIPLY, and SUBTRACT.
- An effective rounding method field, an IEEE-inexact-exception control (XxC), and a quantum-exception control (XqC) are added to CONVERT FROM FIXED.

(August, 2010).

Floating-Point-Support-Sign-Handling Facility

The *floating-point-support-sign-handling* facility includes four instructions: COPY SIGN, LOAD COMPLEMENT, LOAD NEGATIVE, and LOAD POSITIVE. These instructions operate on a 64-bit (long) floating-

point datum independent of the radix and do not set the condition code. (April, 2007)

FPR-GR-Transfer Facility

The *FPR-GR-transfer facility* includes two instructions: LOAD FPR FROM GR and LOAD GR FROM FPR. These instructions provide the means to move a 64-bit (long) floating-point datum between a floating-point register and a general register. (April, 2007)

General-Instructions-Extension Facility

The *general-instructions-extension facility* may be available on a model implementing z/Architecture. The facility provides the following new instructions:

- ADD LOGICAL WITH SIGNED IMMEDIATE
- COMPARE AND BRANCH
- COMPARE AND BRANCH RELATIVE
- COMPARE AND TRAP
- COMPARE HALFWORD RELATIVE LONG
- COMPARE IMMEDIATE AND BRANCH
- COMPARE IMMEDIATE AND BRANCH RELATIVE
- COMPARE IMMEDIATE AND TRAP
- COMPARE LOGICAL AND BRANCH
- COMPARE LOGICAL AND BRANCH RELATIVE
- COMPARE LOGICAL AND TRAP
- COMPARE LOGICAL IMMEDIATE AND BRANCH
- COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE
- COMPARE LOGICAL IMMEDIATE AND TRAP
- COMPARE LOGICAL RELATIVE LONG
- COMPARE RELATIVE LONG
- EXTRACT CACHE ATTRIBUTE
- LOAD HALFWORD RELATIVE LONG
- LOAD LOGICAL HALFWORD RELATIVE LONG
- LOAD LOGICAL RELATIVE LONG
- LOAD RELATIVE LONG
- MULTIPLY SINGLE IMMEDIATE
- PREFETCH DATA
- PREFETCH DATA RELATIVE LONG
- ROTATE THEN AND SELECTED BITS
- ROTATE THEN EXCLUSIVE OR SELECTED BITS
- ROTATE THEN INSERT SELECTED BITS
- ROTATE THEN OR SELECTED BITS
- STORE HALFWORD RELATIVE LONG
- STORE RELATIVE LONG

Additionally, the following instructions have been enhanced to include additional formats:

- ADD IMMEDIATE
- COMPARE HALFWORD
- COMPARE HALFWORD IMMEDIATE
- COMPARE LOGICAL IMMEDIATE
- LOAD ADDRESS EXTENDED
- LOAD AND TEST
- MOVE
- MULTIPLY
- MULTIPLY HALFWORD

(February, 2008)

HFP Multiply-and-Add/Subtract Facility

The *HFP-multiply-and-add/subtract facility* provides instructions for improved processing of hexadecimal floating-point numbers. The MULTIPLY AND ADD (or SUBTRACT) instruction is intended to be used in place of MULTIPLY followed by ADD (or SUBTRACT) NORMALIZED. (June, 2003)

HFP-Unnormalized-Extensions Facility

The *HFP-unnormalized-extension facility* may be available on a model implementing z/Architecture. The facility provides instructions for improved processing of unnormalized hexadecimal floating-point numbers. It extends the capabilities of the HFP-multiply-and-add/subtract facility, by providing the following instructions that operate on unnormalized operands:

MULTIPLY UNNORMALIZED
MULTIPLY AND ADD UNNORMALIZED

(September, 2005)

High-Word Facility

The *high-word facility* may be available on a model implementing z/Architecture. For selected 32-bit instructions, the high-word facility effectively provides sixteen additional 32-bit registers by utilizing bits 0-31 of the sixteen 64-bit general registers. The facility provides the following instructions.

- ADD HIGH
- ADD IMMEDIATE HIGH
- ADD LOGICAL HIGH
- ADD LOGICAL WITH SIGNED IMMEDIATE HIGH
- BRANCH RELATIVE ON COUNT HIGH
- COMPARE HIGH
- COMPARE IMMEDIATE HIGH
- COMPARE LOGICAL HIGH
- COMPARE LOGICAL IMMEDIATE HIGH
- LOAD BYTE HIGH
- LOAD HALFWORD HIGH
- LOAD HIGH
- LOAD LOGICAL CHARACTER HIGH
- LOAD LOGICAL HALFWORD HIGH
- ROTATE THEN INSERT SELECTED BITS HIGH
- ROTATE THEN INSERT SELECTED BITS LOW
- STORE CHARACTER HIGH
- STORE HALFWORD HIGH
- SUBTRACT HIGH
- SUBTRACT LOGICAL HIGH

(August, 2010)

IEEE-Exception-Simulation Facility

The *IEEE-exception-simulation facility* includes the instructions SET FPC AND SIGNAL and LOAD FPC AND SIGNAL. These instructions provide a means to simulate a data exception program interruption. (April, 2007)

Interlocked-Access Facility

The *interlocked-access facility* may be available on a model implementing z/Architecture. The facility provides the means by which a load, update, and store operation can be performed with interlocked update in a single instruction (as opposed to using a compare-and-swap type of update). The facility also provides an instruction to attempt to load from two distinct storage locations in an interlocked-fetch manner. The facility provides the following instructions.

- LOAD AND ADD
- LOAD AND ADD LOGICAL
- LOAD AND AND
- LOAD AND EXCLUSIVE OR
- LOAD AND OR
- LOAD PAIR DISJOINT

(August, 2010)

IPTE-Range Facility

The *IPTE-range facility* may be available on a model implementing z/Architecture. The facility provides performance improvements for the INVALIDATE PAGE TABLE ENTRY instruction. (August, 2010)

List-Directed Initial Program Load

The *list-directed initial-program-load (IPL)* function may be available on a model. List-directed IPL, also known as SCSI IPL in other System Library publications, provides the means by which a program can be loaded from an I/O device other than a classical channel-attached device (ESCON or FICON). Facilities are also provided for loading a stand-alone dump program using list-directed IPL. (May, 2004)

Load/Store-on-Condition Facility

The *load/store-on-condition facility* may be available on a model implementing z/Architecture. The facility provides the means by which selected operations may be executed only when a condition-code-mask field of the instruction matches the current condition code in the PSW. The facility provides the following instructions.

- LOAD ON CONDITION
- STORE ON CONDITION

(August, 2010)

Long-Displacement Facility

The *long-displacement facility* provides a 20-bit signed-displacement field in 69 previously existing instructions (by using a previously unused byte in the instructions) and 44 new instructions. A 20-bit signed displacement allows relative addressing of up to 524,287 bytes beyond the location designated by a base register or base-and-index-register pair and up to 524,288 bytes before that location. The enhanced previously existing instructions generally are ones that handle 64-bit binary integers. The new instructions generally are new versions of instructions for 32-bit binary integers. The new instructions also include (1) a LOAD BYTE instruction that sign-extends a byte from storage to form a 32-bit or 64-bit result in a general register and (2) new floating-point LOAD and STORE instructions. The long-displace-

ment facility provides register-constraint relief by reducing the need for base registers, code size reduction by allowing fewer instructions to be used, and additional improved performance through removal of possible address-generation interlocks. (June, 2003)

Message-Security Assist

The *message-security assist* (MSA) may be available on a model implementing z/Architecture. The MSA basic facility includes the following instructions:

```
CIPHER MESSAGE  
CIPHER MESSAGE WITH CHAINING  
COMPUTE INTERMEDIATE MESSAGE  
DIGEST  
COMPUTE LAST MESSAGE DIGEST  
COMPUTE MESSAGE AUTHENTICATION  
CODE
```

Also included are five query functions and two functions for generating a message digest based on the secure-hash algorithm (SHA-1). The five query functions, one for each instruction, are used to determine the additional installed MSA facilities, which may include the following.

MSA Data-Encryption-Algorithm (DEA) Facility: The *MSA DEA facility* consists of nine functions for ciphering messages, with or without chaining, and for generating a message-authentication code (MAC) using a 56-bit, 112-bit, or 168-bit cryptographic key.¹ All of these functions are based on the DEA algorithm. (June, 2003)

Message-Security-Assist Extension 1

The *message-security-assist extension 1* may be available on models implementing the message-security assist. The extension provides the following functions:

MSA SHA-256 Facility: This facility consists of two functions, one for generating an intermediate message digest and another for generating a final message digest.

1. These key lengths reflect the cryptographic strength. In subsequent chapters, they are referred to as 64-bit, 128-bit, or 192-bit, respectively, to include the DEA-key-parity bits.

MSA Advanced-Encryption-Standard (AES-128) Facility: This facility consists of two functions for ciphering messages, with or without chaining, using the AES-128 algorithm.

MSA Pseudo-Random-Number Generation (PRNG) Facility: This facility consists of a function for generating a multiple of 64-bit pseudo-random numbers using the ANSI[®] X9.17 pseudo-random-number algorithm.

(September, 2005)

Message-Security-Assist Extension 2

The *message-security-assist extension 2* may be available on models implementing the message-security assist. The extension provides the following functions:

MSA AES-192 Facility: This facility consists of two functions for ciphering a message, with or without chaining, using the AES-192 algorithm.

MSA AES-256 Facility: This facility consists of two functions for ciphering a message, with or without chaining, using the AES-256 algorithm.

MSA SHA-512 Facility: This facility consists of two functions, one for generating an intermediate message digest and another for generating a final message digest, using the SHA-512 algorithm.

(February, 2008)

Message-Security-Assist Extension 3

The *message-security-assist extension 3* provides a means to protect user cryptographic keys by encrypting them under machine-generated wrapping keys. When this extension is installed, two wrapping keys are provided for each configuration: one for protecting user DEA keys and another for protecting user AES keys. The wrapping keys reside in the machine so that, with an appropriate setting of controls, no clear value of user cryptographic keys is observed any where in the system by any program.

The message-security-assist extension 3 may be available on models implementing the message-security assist. The extension provides the following features:

- A 256-Bit AES Wrapping-Key Register: The register contents are used to protect user AES keys.
- A 256-Bit AES Wrapping-Key Verification-Pattern Register: The register contents are used to identify the version of the AES wrapping key.
- A 192-Bit DEA Wrapping-Key Register: The register contents are used to protect user DEA keys.
- A 192-Bit DEA Wrapping-Key Verification-Pattern Register: The register contents are used to identify the version of the DEA wrapping key.
- The following functions under the CIPHER MESSAGE instruction:
 - Encrypted DEA
 - Encrypted TDEA 128
 - Encrypted TDEA 192
 - Encrypted AES 128
 - Encrypted AES 192
 - Encrypted AES 256

These functions use an encrypted cryptographic key.

- The following functions under the CIPHER MESSAGE WITH CHAINING instruction:
 - Encrypted DEA
 - Encrypted TDEA 128
 - Encrypted TDEA 192
 - Encrypted AES 128
 - Encrypted AES 192
 - Encrypted AES 256

These functions use an encrypted cryptographic key.

- The following functions under the COMPUTE MESSAGE AUTHENTICATION CODE instruction:
 - Encrypted DEA
 - Encrypted TDEA 128
 - Encrypted TDEA 192

These functions use an encrypted cryptographic key.

- The PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATION instruction with the following functions:

- Query
- Encrypt DEA Key
- Encrypt TDEA 128 Key
- Encrypt TDEA 192 Key
- Encrypt AES 128 Key
- Encrypt AES 192 Key
- Encrypt AES 256 Key

These functions provide a means for importing clear cryptographic keys.

(August, 2010)

Message-Security-Assist Extension 4

The *message-security-assist extension 4* provides the support for the CFB (cipher feedback) mode, the OFB (output feedback) mode, and the CTR (counter) mode. In addition, a number of primitives are provided to facilitate the support for the CMAC (cipher-based message authentication code) mode, the CCM (counter with cipher block chaining – message authentication code) mode, the GCM (Galois/counter mode), and the XTS mode.

The message-security-assist extension 4 may be available on models implementing the message-security assist. The extension provides the following instructions and functions:

- The CIPHER MESSAGE WITH CFB, CIPHER MESSAGE WITH COUNTER, and CIPHER MESSAGE WITH OFB instructions. Each of these instructions includes the following functions:
 - Query
 - DEA
 - TDEA 128
 - TDEA 192
 - Encrypted DEA
 - Encrypted TDEA 128
 - Encrypted TDEA 192
 - AES 128
 - AES 192
 - AES 256
 - Encrypted AES 128
 - Encrypted AES 192

- Encrypted AES 256
- The PERFORM CRYPTOGRAPHIC COMPUTATION instruction with the following functions:
 - Query
 - Compute Last Block CMAC Using DEA
 - Compute Last Block CMAC Using TDEA 128
 - Compute Last Block CMAC Using TDEA 192
 - Compute Last Block CMAC Using Encrypted DEA
 - Compute Last Block CMAC Using Encrypted TDEA 128
 - Compute Last Block CMAC Using Encrypted TDEA 192
 - Compute Last Block CMAC Using AES 128
 - Compute Last Block CMAC Using AES 192
 - Compute Last Block CMAC Using AES 256
 - Compute Last Block CMAC Using Encrypted AES 128
 - Compute Last Block CMAC Using Encrypted AES 192
 - Compute Last Block CMAC Using Encrypted AES 256
 - Compute XTS Parameter Using AES 128
 - Compute XTS Parameter Using AES 256
 - Compute XTS Parameter Using Encrypted AES 128
 - Compute XTS Parameter Using Encrypted AES 256
- The following functions under the CIPHER MESSAGE instruction:
 - XTS AES 128
 - XTS AES 256
 - XTS Encrypted AES 128
 - XTS Encrypted AES 256
- The following function under the COMPUTE INTERMEDIATE MESSAGE DIGEST instruction:
 - GHASH
- The following functions under the COMPUTE MESSAGE AUTHENTICATION CODE instruction:
 - AES 128
 - AES 192
 - AES 256
 - Encrypted AES 128
 - Encrypted AES 192
 - Encrypted AES 256

The message-security-assist extension 4 requires the message-security-assist extension 3 as a prerequisite.

(August, 2010)

Modified CCW Indirect Data Addressing Facility

The *modified-CCW-indirect-data-addressing (MIDA) facility* may be available on models implementing z/Architecture and provides the program an alternate means to transfer large amounts of data that spans noncontiguous blocks in main storage without the overhead of data chaining and without the strict boundary and count restrictions imposed by CCW indirect data addressing (IDA). Modified CCW indirect data addressing permits a single channel-command word to control the transfer of up to 65,535 bytes of data that spans noncontiguous blocks in main storage. Each block of main storage to be transferred may be specified on any boundary and length up to 4K bytes, provided the specified block does not cross a 4K-byte boundary.

Use of modified CCW indirect data addressing may be restricted to devices accessible by certain channel-path types.

When modified CCW indirect data addressing is used, the CCW data address is not used to directly address data. Instead, the address points to a contiguous list of up to 256 quadwords called the modified-indirect-data-address list (MIDAL). Each quadword in the MIDAL is called a modified-indirect-data-address word (MIDAW) that describes a block of storage to be transferred and contains flags, a byte count, and a 64-bit address designating a data area in absolute storage. When modified CCW indirect data addressing is used, transfer of control from one MIDAW to the next is made when the count of bytes specified by the MIDAW count field has been transferred and the total count of bytes transferred does not yet equal the count specified by the CCW. This is dissimilar to indirect data addressing where the transfer of control from one IDAW to the next is made when a program-specified 2K or 4K boundary is reached and the total count of bytes transferred does not yet equal the count specified in the CCW.

In addition to the MIDAW, the ORB modified-CCW-indirect-data-addressing-control bit and the CCW

modified-indirect-data-address flag are added. (September, 2005)

Move-With-Optional-Specifications Facility

The *move-with-optional-specifications facility* may be available on a model implementing z/Architecture. The facility adds the semiprivileged MOVE WITH OPTIONAL SPECIFICATIONS instruction. This instruction provides the means of moving from a source operand to a destination operand using different address-space-control modes and different keys for each operand. (February, 2008)

Multiple-Subchannel-Set Facility

The *multiple-subchannel-set (MSS) facility* may be available on a model implementing z/Architecture and increases the maximum number of subchannels that can be configured to a program. When the MSS facility is not installed, a single set of subchannels, in the range 0-65,535, may be provided. When the MSS facility is installed, a maximum of four sets of subchannels may be provided for a program. Each sub-channel set provides from one to 64K subchannels in the range 0 to-65,535. (February, 2008)

Nonquiescing Key-Setting Facility

The *nonquiescing key-setting facility* may be available on a model implementing z/Architecture. The facility provides performance improvements for the PERFORM FRAME MANAGEMENT FUNCTION and SET STORAGE KEY EXTENDED instruction. (August, 2010)

Parsing-Enhancement Facility

The *parsing-enhancement facility* may be available on a model implementing z/Architecture. The facility adds the following instructions:

TRANSLATE AND TEST EXTENDED
TRANSLATE AND TEST REVERSE
EXTENDED

These instructions perform functions similar to those of the TRANSLATE AND TEST and TRANSLATE AND TEST REVERSE instructions, respectively, but

provide the capability of processing either single-byte or double-byte argument characters, returning either 8-bit or 16-bit function codes. (February, 2008)

PER-3 Facility:

The *PER-3 facility* may be available on a model implementing z/Architecture. When the facility is installed, the following two functions are available for use by the program:

Breaking-Event Address Register: The breaking-event-address register is a 64-bit CPU register that is updated with the address of any instruction that causes a break in sequential instruction execution (that is, the instruction address in the PSW is replaced, rather than incremented by the length of the instruction). When the PER-3 facility is installed and a program interruption occurs, whether or not PER is indicated, the contents of the breaking-event-address register are stored in real storage locations 272-279. This can be used as a debugging assist for wild-branch detection.

PER Instruction-Fetching Nullification: Bit 39 of control register 9, when one, specifies that PER instruction-fetching events force nullification. Bit 39 of control register 9 is meaningful only when bit 33 of control register 9, the instruction-fetching PER-event mask, is also one. When the PER-3 facility is not installed, or when bit 39 of control register 9 is zero, nullification is not forced for PER instruction-fetching events (called a PER instruction-fetching basic event).

(September, 2005)

PFPO Facility

The *PFPO facility* provides the instruction PERFORM FLOATING-POINT OPERATION. This instruction, designed for future expansion, currently provides 54 conversion functions (from any of nine floating-point data formats to any of the six formats in another floating-point radix) using any of eight rounding methods. All conversions are correctly rounded. (April, 2007)

Population-Count Facility

The *population-count facility* may be available on a model implementing z/Architecture. The facility pro-

vides the POPULATION COUNT instruction which provides a count of one bits in each byte of a general register.

(August, 2010)

The Reset-Reference-Bits-Multiple Facility

The *reset-reference-bits-multiple* facility may be available on a model implementing z/Architecture. The facility provides performance improvements for the inspection and resetting of reference bits by means of the RESET REFERENCE BITS MULTIPLE instruction.

(August, 2010)

Restore-Subchannel Facility

The *restore-subchannel* facility may be available on a model implementing z/Architecture. The facility provides the means for the channel subsystem to recover a damaged subchannel and report the recovery to the program by means of a CRW. (February, 2008)

Server-Time-Protocol Facility:

The *server-time-protocol* facility may be available on a model implementing z/Architecture. The facility provides the means by which the time-of-day (TOD) clocks in various systems can be synchronized using message links. STP operates in conjunction with the TOD-clock-steering facility, providing a new timing mode, new timing states, a new STP-timing-alert external interruption, and a new STP-sync-check machine-check conditions. (September, 2005)

Store-Clock-Fast Facility

The *store-clock-fast* facility may be available on a model implementing z/Architecture. The facility provides a means by which an eight-byte time-of-day clock value may be stored without any artificial delay to ensure uniqueness. When the facility is installed, the TOD-clock bits stored by TRACE (TRACE and TRACG) are subject to additional control by a bit in control register 0. The facility provides the STORE CLOCK FAST instruction. (September, 2005)

Store-Facility-List-Extended Facility:

The *store-facility-list-extended* facility may be available on a model implementing z/Architecture. The facility extends the function provided by the STORE FACILITY LIST (STFL) instruction. Whereas STFL is a control instruction that can store an indication of 32 facilities at real location 200, the new STORE FACILITY LIST EXTENDED (STFLE) instruction is a general instruction that can store an indication of up to 16,384 facilities at a program-specified location. (September, 2005)

TOD-Clock-Steering Facility

The *TOD-clock-steering* facility may be available on a model implementing z/Architecture. The facility provides a means by which apparent stepping rate of the time-of-day clock may be altered without changing the physical hardware oscillator which steps the physical clock. The facility adds the semiprivileged PERFORM TIMING FACILITY FUNCTION (PTFF) instruction which provides the means by which the program can query various timing-related parameters, and, optionally, the means by which an authorized timing-control program can influence certain of these parameters. (September, 2005)

The ESA/390 Base

z/Architecture includes all of the facilities of ESA/390 except for the asynchronous-pageout, asynchronous-data-mover, program-call-fast, and vector facilities. This section briefly outlines most of the remaining facilities that were additions in ESA/390 as compared to ESA/370.

ESA/390 is described in *Enterprise Systems Architecture/390 Principles of Operation*, SA22-7201.

The CPU-related facilities that were new in ESA/390 are summarized below. ESA/390 was announced in September, 1990. Any extension added subsequently has the date of its announcement in parentheses at the end of its summary.

The following extensions are described in detail in SA22-7201 and in this publication:

- *Access-list-controlled protection* allows store-type storage references to an address space to be prohibited by means of a bit in the access-list entry used to access the space. Thus, different users having different access lists can have different capabilities to store in the same address space.
- The *program-event-recording facility 2 (PER 2)* is an alternative to the original PER facility, which is now named PER 1. (Neither of the names PER 1 and PER 2 is used in z/Architecture; only “PER” is used.) PER 2 provides the option of having a successful-branching event occur only when the branch target is within the designated storage area, and it provides the option of having a storage-alteration event occur only when the storage area is within designated address spaces. The use of these options improves performance by allowing only PER events of interest to occur. PER 2 deletes the ability to monitor for general-register-alteration events.

PER 2 includes extensions that provide additional information about PER events. The extensions were described in detail beginning in the fourth edition of SA22-7201.

- *Concurrent sense* improves performance by allowing sense information to be presented at the time of an interruption due to a unit-check condition, thus avoiding the need for a separate I/O operation to obtain the sense information.
- *Broadcasted purging* provides the COMPARE AND SWAP AND PURGE instruction for conditionally updating tables associated with dynamic address translation and access-register translation and clearing associated buffers in multiple CPUs. This is described in detail beginning in the eighth edition of SA22-7201.
- *Storage-protection override* provides a new form of subsystem storage protection that improves the reliability of a subsystem executed in an address space along with possibly erroneous application programs. When storage-protection override is made active by a control-register bit, fetches and stores by the CPU are permitted to storage locations having a storage key of 9 regardless of the access key used by the CPU. If the subsystem is in key-8 storage and is executed with a PSW key of 8, for example, and the application programs are in key-9 storage and are executed with a PSW key of 9, accesses by

the subsystem to the application-program areas are permitted while accesses by the application programs to the subsystem area are denied. (September, 1991)

- *Move-page facility 2* extends the MOVE PAGE instruction introduced in ESA/370 by allowing use of a specified access key for either the source or the destination operand, by allowing improved performance when the destination operand will soon be referenced, and by allowing improved performance when an operand is invalid in both main and expanded storage. The ESA/370 version of MOVE PAGE is now called move-page facility 1 and is in Chapter 7, “General Instructions.” MOVE PAGE of move-page facility 2 is in Chapter 10, “Control Instructions.” Some details about the means for control-program support of MOVE PAGE are not provided. (September, 1991) (The z/Architecture MOVE PAGE instruction is described only in Chapter 10 of this publication. MOVE PAGE no longer can move data to or from expanded storage, and all details about MOVE PAGE are provided.)
- The *square-root facility* consists of the SQUARE ROOT instruction and the square-root exception. The instruction extracts the square root of a floating-point operand in either the long or short format. The instruction is the same as that provided on some models of the IBM 4341, 4361, and 4381 Processors. (September, 1991)
- The *cancel-I/O facility* allows the program to withdraw a pending start function from a designated subchannel without signaling the device, which is useful in certain error-recovery situations. (September, 1991)

The cancel-I/O facility provides the CANCEL SUBCHANNEL instruction and is described in detail beginning in the eighth edition of SA22-7201.

- The *string-instruction facility (or logical string assist)* provides instructions for (1) moving a string of bytes until a specified ending byte is found, (2) logically comparing two strings until an inequality or a specified ending byte is found, and (3) searching a string of a specified length for a specified byte. The first two instructions are particularly useful in a C program in which strings are normally delimited by an ending byte of all zeros. (June, 1992)

- The *suppression-on-protection facility* causes a protection exception due to page protection to result in suppression of instruction execution instead of termination of instruction execution, and it causes the address and an address-space identifier of the protected page to be stored in low storage. This is useful in performing the AIX/ESA[®] copy-on-write function, in which AIX/ESA causes the same page of different address spaces to map to a single page frame of real storage so long as a store in the page is not attempted and then, when a store is attempted in a particular address space, assigns a unique page frame to the page in that address space and copies the contents of the page to the new page frame. (February, 1993)
- The *set-address-space-control-fast facility* consists of the SET ADDRESS SPACE CONTROL FAST (SACF) instruction, which possibly can be used instead of the previously existing SET ADDRESS SPACE CONTROL (SAC) instruction, depending on whether all of the SAC functions are required. SACF, unlike SAC, does not perform the serialization and checkpoint-synchronization functions, nor does it cause copies of prefetched instructions to be discarded. SACF provides improved performance on some models. (February, 1993)
- The *subspace-group facility* includes the BRANCH IN SUBSPACE GROUP instruction, which can be used to give or return control from one address space to another in a group of address spaces called a subspace group, with this giving and returning of control being done with better performance than can be obtained by means of the PROGRAM CALL and PROGRAM RETURN or PROGRAM TRANSFER instructions. One address space in the subspace group is called the base space, and the other address spaces in the group are called subspaces. It is intended that each subspace contain a different subset of the storage in the base space, that the base space and each subspace contain a subsystem control program, such as CICS[®], and application programs, and that each subspace contain the data for a single transaction being processed under the subsystem control program. The placement of the data for each transaction in a different subspace prevents the processing of a transaction from erroneously damaging the data of other transactions. The data of the control program can be protected from the transaction processing by means of the storage-protection-override facility. (April, 1994)
- The *virtual-address enhancement of suppression on protection* provides that if dynamic address translation (DAT) was on when a protection exception was recognized, the suppression-on-protection result is indicated, and the address of the protected location is stored, only if the address is one that was to be translated by DAT; the suppression-on-protection result is not indicated if the address that would be stored is a real address. This enhancement allows the stored address to be translated reliably by the control program to determine if the exception was due to page protection as opposed to key-controlled protection. The enhancement extends the usefulness of suppression on protection to operating systems like MVS/ESA[™] that use key-controlled protection. (September, 1994)
- The *immediate-and-relative-instruction facility* includes 13 new instructions, most of which use a halfword-immediate value for either signed-binary arithmetic operations or relative branching. The facility reduces the need for general registers, and, in particular, it eliminates the need to use general registers to address branch targets. As a result, the general registers and access registers can be allocated more efficiently in programs that require many registers. (September, 1996)
- The *compare-and-move-extended facility* provides new versions of the COMPARE LOGICAL LONG and MOVE LONG instructions. The new versions increase the size of the operand-length specifications from 24 bits to 32 bits, which can be useful when objects larger than 16M bytes are processed through the use of 31-bit addressing. The new versions also periodically complete to allow software polling in a multiprocessing system. (September, 1996)
- The *checksum facility* consists of the CHECKSUM instruction, which can be used to compute a 16-bit or 32-bit checksum in order to improve TCP/IP (transmission-control protocol/internet protocol) performance. (September, 1996)
- The *called-space-identification facility* improves serviceability by further identifying the called address space in a linkage-stack state entry formed by the PROGRAM CALL instruction. (September, 1996)

- The *branch-and-set-authority facility* consists of the BRANCH AND SET AUTHORITY instruction, which can be used to improve the performance of linkages within an address space by replacing PROGRAM CALL, PROGRAM TRANSFER, and SET PSW KEY FROM ADDRESS instructions. (June, 1997)
- The *perform-locked-operation facility* consists of the unprivileged PERFORM LOCKED OPERATION instruction, which appears to provide concurrent interlocked-update references to multiple storage operands. A function code of the instruction can specify any of six operations: compare and load, compare and swap, double compare and swap, compare and swap and store, compare and swap and double store, and compare and swap and triple store. The function code further specifies either word or doubleword operands. The instruction can be used to avoid the use of programmed locks in a multiprocessing system. (June, 1997)
- Four additional floating-point facilities improve the hexadecimal-floating-point (HFP) capability of the machine and add a binary-floating-point (BFP) capability. The facilities are:
 - *Basic floating-point extensions*, which provides 12 additional floating-point registers to make a total of 16 floating-point registers. This facility also includes a floating-point-control register and means for saving the contents of the new registers during a store-status operation or a machine-check interruption.
 - *Floating-point-support (FPS) extensions*, which provides eight new instructions, including four to convert data between the HFP and BFP formats.
 - *Hexadecimal-floating-point (HFP) extensions*, which provides 26 new instructions to operate on data in the HFP format. All of these are counterparts to new instructions provided by the BFP facility, including conversion between floating-point and fixed-point formats, and a more complete set of operations on the extended format.
 - *Binary floating-point (BFP)*, which defines short, long, and extended binary-floating-point (BFP) data formats and provides 87 new instructions to operate on data in these formats. The BFP formats and operations

provide everything necessary to conform to the IEEE standard (ANSI/IEEE Standard 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, dated August 12, 1985) except for conversion between binary-floating-point numbers and decimal strings, which must be provided in software.

(May, 1998)

- The *resume-program facility* consists of the RESUME PROGRAM instruction, which restores, from a specified save area, the instruction address and certain other fields in the current PSW and also the contents of an access-and-general-register pair. RESUME PROGRAM allows a problem-state interruption-handling program to restore the state of an interrupted program and return to that program despite that a register is required for addressing the save area from which the state is restored. (May, 1998)
- The *trap facility* provides the TRAP instructions (a two-byte TRAP2 instruction and a four-byte TRAP4 instruction) that can overlay instructions in an application program to give control to a program that can perform fix-up operations on data being processed, such as dates that may be a “Year-2000” problem. RESUME PROGRAM can be used to return from the fix-up program. TRAP and RESUME PROGRAM can improve performance by avoiding program interruptions that would otherwise be needed to give control to and from the fix-up program. (May, 1998)
- The *extended-TOD-clock facility* includes (1) an extension of the TOD clock from 64 bits to 104 bits, allowing greater resolution; (2) a TOD programmable register, which contains a TOD programmable field that can be used to identify the configuration providing a TOD-clock value in a sysplex; (3) the SET CLOCK PROGRAMMABLE FIELD instruction, for setting the TOD programmable field in the TOD programmable register; and (4) the STORE CLOCK EXTENDED instruction, which stores both the longer TOD-clock value and the TOD programmable field. STORE CLOCK EXTENDED can be used in the future when the TOD clock is further extended to contain time values that exceed the current year-2042 limit (when there is a carry out of the current bit 0 of the TOD clock). (August, 1998)
- The *TOD-clock-control-override facility* provides a control-register bit that allows setting the TOD

clock under program control, without use of the manual TOD-clock control of any CPU. (August, 1998)

- The *store-system-information facility* provides the privileged STORE SYSTEM INFORMATION instruction, which can be used to obtain information about a component or components of a virtual machine, a logical partition, or the basic machine. (January, 1999)
- The *extended-translation facility*, now called the extended-translation facility 1, includes the CONVERT UNICODE TO UTF-8, CONVERT UTF-8 TO UNICODE, and TRANSLATE EXTENDED instructions, all of which can improve performance. TRANSLATE EXTENDED can be used in place of a TRANSLATE AND TEST instruction that locates an escape character, followed by a TRANSLATE instruction that translates the bytes preceding the escape character. (April, 1999)

The following extensions are described in detail in other publications:

- The *Enterprise Systems Connection Architecture[®] (ESCON[®])* introduces a new type of channel that uses an optical-fiber communication link between channels and control units. Information is transferred serially by bit, at 200 million bits per second, up to a maximum distance of 60 kilometers. The optical-fiber technology and serial transmission simplify cabling and improve reliability. See the publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.
- The *ESCON channel-to-channel adapter (ESCON CTCA)* provides the same type of function for serial channel paths as is available for the parallel-I/O-interface channel paths. See the publication *IBM Enterprise Systems Architecture/390 ESCON Channel-to-Channel Adapter*, SA22-7203.
- *I/O-device self-description* allows a device to describe itself and its position in the I/O configuration. See the publication *IBM Enterprise Systems Architecture/390 Common I/O-Device Commands and Self Description*, SA22-7204.
- The *compression facility* performs a Ziv-Lempel type of compression and expansion by means of static (nonadaptive) dictionaries that are to be prepared by a program before the compression and expansion operations. Because the dictionaries are static, the compression facility can provide good compression not only for long sequential data streams (for example, archival or network data) but also for randomly accessed short records (for example, 80 bytes). See the publication *IBM Enterprise Systems Architecture/390 Data Compression*, SA22-7208. (February, 1993) (The z/Architecture COMPRESSION CALL instruction is described in this publication. However, introductory information and information about dictionary formats still is provided only in SA22-7208.)

The remaining extensions of ESA/390, for which detailed descriptions are not provided, are as follows:

- The integrated *cryptographic facility* provides a number of instructions to protect data privacy, to support message authentication and personal identification, and to facilitate key management. The high-performance cipher capability of the facility is designed for financial-transaction and bulk-encryption environments, and it complies with the Data Encryption Standard (DES).
 - Usability of the cryptographic facility is extended to virtual-machine environments, which allows the facility to be used by MVS/ESA being executed under VM/ESA[®], which in turn may be executed either under another VM/ESA or in a logical partition. (September, 1991)
- The *external-time-reference facility* provides a means to initiate and maintain the synchronization of TOD clocks to an external time reference (ETR). Synchronization tolerance of a few microseconds can be achieved, and the effect of leap seconds is taken into account. The facility consists of an ETR sending unit (Sysplex Timer[®]), which may be duplexed, two or more ETR receiving units, and optical-fiber cables. The cables are used to connect the ETR sending unit, which is an external device, to ETR receiving units of the configuration. CPU instructions are provided for setting the TOD clock to the value supplied by the ETR sending unit.
 - The ETR *automatic-propagation-delay-adjustment* function adjusts the time signals from the ETR to the attached processors to compensate for the propagation delay on the cables to the processors, thus allowing the

cables to be of different lengths. (September, 1991)

- The ETR *external-time-source* function synchronizes the ETR to a time signal received from a remote location by means of a telephone or radio. (September, 1991)
- *Extended sorting* provides instructions that improve the performance of the DB2® sorting function.
- Other *PER extensions*, besides those described beginning in the fourth edition of this publication, are an augmentation of PER 2 that provide additional PER function in the interpretive-execution mode.
- *Channel-subsystem call* provides various functions for use in the management of the I/O configuration. Some of the functions acquire information about the configuration from the accessible elements of the configuration, while others dynamically change the configuration.
- The *operational extensions* are a number of other improvements that result in increased availability and ease of use of the system, as follows:
 - *Automatic-reconfiguration* permits an operating system in an LPAR partition to declare itself willing to be terminated suddenly, usually to permit its storage and CPU resources to be acquired by an adjacent partition that is dynamically absorbing the work load of another system that has failed. Other functions deactivate and reset designated participating partitions.
 - A new *storage-reconfiguration* command decreases the time needed to reconfigure storage by allowing multiple requests for reconfiguration to be made by means of a single communication with the service processor.
 - *SCP-initiated reset* allows a system control program (SCP) to reset its I/O configuration prior to entering the disabled wait state following certain check conditions.
 - *Console integration* simplifies configuration requirements by reducing by one the number of consoles required by MVS.
 - The *processor-availability facility* enables a CPU experiencing an unrecoverable error

that will cause a check stop to save its state and alert the other CPUs in the configuration. This allows, in many cases, another CPU to continue execution of the program that was in execution on the failing CPU. The facility is applicable in both the ESA/390 mode and the LPAR mode. (April, 1991)

- *Extensions for virtual machines* are a number of improvements to the interpretive-execution facility, as follows:
 - The *VM-data-space facility* provides for making the ESA/390 access-register architecture more useful in virtual-machine applications. The facility improves the ability to address a larger amount of data and to share data. For information on how VM/ESA uses the VM-data-space facility, see the publication *VM/ESA CP Programming Services*, SC24-5520.
 - A new *storage-key function* improves performance by removing the need for the previously used RCP area.
 - Other improvements include an optional special-purpose lookaside for some of the guest-state information and greater freedom in certain implementation choices.
 - The *ESCON-multiple-image facility (EMIF)* allows multiple logical partitions to share ESCON channels (and FICON channels) and optionally to share any of the control units and associated I/O devices configured to these shared channels. This can reduce channel requirements, improve channel utilization, and improve I/O connectivity. (June, 1992)
 - *PR/SM LPAR* mode is enhanced to allow up to 10 logical partitions in a single-image configuration and 20 in a physically-partitioned configuration. The previous limits were seven and 14, respectively. (June, 1992)
- Coincident with z/Architecture, PR/SM LPAR mode allows 15 logical partitions, and physical partitioning is not supported.
- The *coupling facility* enables high-performance data sharing among MVS/ESA systems that are connected by means of the facility. The coupling facility provides storage that can be dynamically partitioned for caching data in shared buffers, maintaining work queues and status information

in shared lists, and locking data by means of shared lock controls. MVS/ESA services provide access to and manipulation of the coupling-facility contents. (April, 1994)

The ESA/370 and 370-XA Base

ESA/390 includes the complete set of facilities of ESA/370 as its base. This section briefly outlines most of the facilities that were additions in 370-XA as compared to System/370 and that were additions in ESA/370 as compared to 370-XA.

The CPU-related facilities that were new in 370-XA are as follows:

- *Bimodal addressing* provides two modes of operation: a 24-bit addressing mode for the execution of old programs and a 31-bit addressing mode.
- *31-bit logical addressing* extends the virtual address space from the 16M bytes addressable with 24-bit addresses to 2G bytes (2,147,483,648 bytes).
- *31-bit real and absolute addressing* provides addressability for up to 2G bytes of main storage.
- The 370-XA *protection* facilities include key-controlled protection on only 4K-byte blocks, page protection, and, as in System/370, low-address protection for addresses below 512. Fetch-protection override eliminates fetch protection for locations 0-2047.
- The *tracing* facility assists in the determination of system problems by providing an ongoing record in storage of significant events.
- The COMPARE AND FORM CODEWORD and UPDATE TREE instructions facilitate sorting applications.
- The *interpretive-execution facility* allows creation of virtual machines that may operate according to several architectures and whose performance is enhanced because many virtual-machine functions are directly interpreted by the machine rather than simulated by the program. This facility is described in the publication *IBM 370-XA Interpretive Execution*, SA22-7095.
- The *service-call-logical-processor (SCLP) facility* provides a means of communicating between the control program and the service processor for the purpose of describing and changing the configuration. This facility is not described.

The I/O-related differences between 370-XA and System/370 result from the 370-XA *channel subsystem*, which includes:

- *Path-independent addressing* of I/O devices, which permits the initiation of I/O operations without regard to which CPU is executing the I/O instruction or how the I/O device is attached to the channel subsystem. Any I/O interruption can be handled by any CPU enabled for it.
- *Path management*, whereby the channel subsystem determines which paths are available for selection, chooses a path, and manages any busy conditions encountered while attempting to initiate I/O processing with the associated devices.
- *Dynamic reconnection*, which permits any I/O device using this capability to reconnect to any available channel path to which it has access in order to continue execution of a chain of commands.
- *Programmable interruption subclasses*, which permit the programmed assignment of I/O-interruption requests from individual I/O devices to any one of eight maskable interruption queues.
- An *additional CCW format* for the direct use of 31-bit addresses in channel programs. The new CCW format, called format 1, is provided in addition to the System/370 CCW format, now called format 0.
- *Address-limit checking*, which provides an additional storage-protection facility to prevent data access to storage locations above or below a specified absolute address.
- *Monitoring facilities*, which can be invoked by the program to cause the channel subsystem to measure and accumulate key I/O-resource usage parameters.
- *Status-verification facility*, which reports inappropriate combinations of device-status bits presented by a device.
- A set of 13 *I/O instructions*, with associated control blocks, which are provided for the control of the channel subsystem.

The facilities that were new in ESA/370 are as follows:

- Sixteen *access registers* permit the program to have immediate access to storage operands in up to 16 2G-byte address spaces, including the address space in which the program resides. In a dynamic-address-translation mode named *access-register mode*, the instruction B field, or for certain instructions the R field, designates both a general register and an access register, and the contents of the access register, along with the contents of protected tables, specify the operand address space to be accessed. By changing the contents of the access registers, the program, under the control of an authorization mechanism, can have fast access to hundreds of different operand address spaces.
- A *linkage stack* is used in a functionally expanded mechanism for passing control between programs in either the same or different address spaces. This mechanism makes use also of the previously existing PROGRAM CALL instruction, an extended entry-table entry, and a new PROGRAM RETURN instruction. The mechanism saves various elements of status, including access-register and general-register contents, during a calling linkage, provides for changing the current status during the calling linkage, and restores the original status during the returning linkage. The linkage stack can also be used to save and restore access-register and general-register contents during a branch-type linkage performed by the new instruction BRANCH AND STACK.
- A translation mode named *home-space mode* provides an efficient means for the control program to obtain control in the address space, called the home address space, in which the principal control blocks for a dispatchable unit (a task or process) are kept.
- The semiprivileged MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY instructions allow bidirectional movement of data between storage areas having different storage keys, without the need to change the PSW key.
- The privileged LOAD USING REAL ADDRESS and STORE USING REAL ADDRESS instructions allow the control program to access data in real storage more efficiently.

- The *private-space facility* allows an address space not to contain any common segments and causes low-address protection and fetch-protection override not to apply to the address space.
- The unprivileged MOVE PAGE instruction allows the program to move a page of data between main and expanded storage, provided that the source and destination pages are both valid. Some details about the means for control-program support of MOVE PAGE are not provided. The ESA/370 version of MOVE PAGE is now called move-page facility 1.
- The *Processor Resource/Systems Manager™ (PR/SM™)* feature provides support for multiple preferred guests under VM/XA and provides the logically partitioned (LPAR) mode, with the latter providing flexible partitioning of processor resources among multiple logical partitions. Certain aspects of the LPAR use of PR/SM are described in the publication *IBM ES/3090 Processor Complex Processor Resource/Systems Manager Planning Guide*, GA22-7123.
- The COMPARE UNTIL SUBSTRING EQUAL instruction provides improved performance of the compression of IMS log data sets and can be useful in other programs also.

System Program

z/Architecture is designed to be used with a control program that coordinates the use of system resources and executes all I/O instructions, handles exceptional conditions, and supervises scheduling and execution of multiple programs.

Compatibility

Compatibility among z/Architecture Systems

Although systems operating as defined by z/Architecture may differ in implementation and physical capabilities, logically they are upward and downward compatible. Compatibility provides for simplicity in education, availability of system backup, and ease in system growth. Specifically, any program written for

z/Architecture gives identical results on any z/Architecture implementation, provided that the program:

1. Is not time-dependent.
2. Does not depend on system facilities (such as storage capacity, I/O equipment, or optional facilities) being present when the facilities are not included in the configuration.
3. Does not depend on system facilities being absent when the facilities are included in the configuration. For example, the program must not depend on interruptions caused by the use of operation codes or command codes that are not installed in some models. Also, it must not use or depend on fields associated with uninstalled facilities. For example, data should not be placed in an area used by another model for fixed-logout information. Similarly, the program must not use or depend on unassigned fields in machine formats (control registers, instruction formats, etc.) that are not explicitly made available for program use.
4. Does not depend on results or functions that are defined to be unpredictable or model-dependent or are identified as undefined. This includes the requirement that the program should not depend on the assignment of device numbers and CPU addresses.
5. Does not depend on results or functions that are defined in the functional-characteristics publication for a particular model to be deviations from the architecture.
6. Takes into account any changes made to the architecture that are identified as affecting compatibility.

Compatibility between z/Architecture and ESA/390

Control-Program Compatibility

Control programs written for ESA/390 cannot be directly transferred to systems operating as defined by z/Architecture. This is because the general-register and control-register sizes, PSW size, assigned storage locations, and dynamic address translation are changed.

Problem-State Compatibility

A high degree of compatibility exists at the problem-state level in going forward from ESA/390 to z/Architecture. Because the majority of a user's applications are written for the problem state, this problem-state compatibility is useful in many installations.

A problem-state program written for ESA/390 operates with z/Architecture, provided that the program:

1. Complies with the limitations described in "Compatibility among z/Architecture Systems".
2. Is not dependent on privileged facilities which are unavailable on the system.

Programming Note: This publication assigns meanings to various operation codes, to bit positions in instructions, channel-command words, registers, and table entries, and to fixed locations in the low 512 bytes and bytes 4096-8191 of storage. Unless specifically noted, the remaining operation codes, bit positions, and low-storage locations are reserved for future assignment to new facilities and other extensions of the architecture.

To ensure that existing programs operate if and when such new facilities are installed, programs should not depend on an indication of an exception as a result of invalid values that are currently defined as being checked. If a value must be placed in unassigned positions that are not checked, the program should enter zeros. When the machine provides a code or field, the program should take into account that new codes and bits may be assigned in the future. The program should not use unassigned low-storage locations for keeping information since these locations may be assigned in the future in such a way that the machine causes the contents of the locations to be changed.

Availability

Availability is the capability of a system to accept and successfully process an individual job. Systems operating in accordance with z/Architecture permit substantial availability by (1) allowing a large number and broad range of jobs to be processed concurrently, thus making the system readily accessible to any particular job, and (2) limiting the effect of an error and identifying more precisely its cause, with

the result that the number of jobs affected by errors is minimized and the correction of the errors facilitated.

Several design aspects make this possible.

- A program is checked for the correctness of instructions and data as the program is executed, and program errors are indicated separate from equipment errors. Such checking and reporting assists in locating failures and isolating effects.
- The protection facilities, in conjunction with dynamic address translation and the separation of programs and data in different address spaces, permit the protection of the contents of storage from destruction or misuse caused by erroneous or unauthorized storing or fetching by a program. This provides increased security for the user, thus permitting applications with different security requirements to be processed concurrently with other applications.
- Dynamic address translation allows isolation of one application from another, still permitting them to share common resources. Also, it permits the implementation of virtual machines, which may be used in the design and testing of new versions of operating systems along with the concurrent processing of application programs. Additionally, it provides for the concurrent operation of incompatible operating systems.
- The use of access registers allows programs, data, and different collections of data to reside in different address spaces, and this further reduces the likelihood that a store using an incorrect address will produce either erroneous results or a system-wide failure.
- Multiprocessing and the channel subsystem permit better use of storage and processing capabilities, more direct communication between CPUs, and duplication of resources, thus aiding in the continuation of system operation in the event of machine failures.
- MONITOR CALL, program-event recording, and the timing facilities permit the testing and debugging of programs without manual intervention and with little effect on the concurrent processing of other programs.
- On most models, error checking and correction (ECC) in main storage, CPU retry, and command retry provide for circumventing intermittent equipment malfunctions, thus reducing the number of equipment failures.
- An enhanced machine-check-handling mechanism provides model-independent fault isolation, which reduces the number of programs impacted by uncorrected errors. Additionally, it provides model-independent recording of machine-status information. This leads to greater machine-check-handling compatibility between models and improves the capability for loading and operating a program on a different model when a system failure occurs.
- A small number of manual controls are required for basic system operation, permitting most operator-system interaction to take place *via* a unit operating as an I/O device and thus reducing the possibility of operator errors.
- The logical partitions made available by the PR/SM feature allow continued reliable production operations in one or more partitions while new programming systems are tested in other partitions. This is an advancement in particular for non-VM installations.
- The operational extensions and channel-subsystem-call facility of ESA/390 and z/Architecture improve the ability to continue execution of application programs in the presence of system incidents and the ability to make configuration changes with less disruption to operations.

Chapter 2. Organization

Main Storage	2-2	Access Registers	2-5
Expanded Storage	2-2	Cryptographic Facility	2-5
CPU	2-2	External Time Reference	2-5
PSW	2-3	I/O	2-5
General Registers	2-3	Channel Subsystem	2-5
Floating-Point Registers	2-3	Channel Paths	2-6
Floating-Point-Control Register	2-3	I/O Devices and Control Units	2-6
Control Registers	2-5	Operator Facilities	2-6

Logically, a system consists of main storage, one or more central processing units (CPUs), operator facilities, a channel subsystem, and I/O devices. I/O devices are attached to the channel subsystem through control units. The connection between the channel subsystem and a control unit is called a channel path.

A channel path employs either a parallel-transmission protocol or a serial-transmission protocol and, accordingly, is called either a parallel or a serial channel path. A serial channel path may connect to a control unit through a dynamic switch that is capable of providing different internal connections between the ports of the switch.

Expanded storage may also be available in the system, a cryptographic unit may be included in a CPU, and an external time reference (ETR) may be connected to the system.

The physical identity of the above functions may vary among implementations, called “models”. Figure 2-1 depicts the logical structure of a two-CPU multiprocessing system that includes expanded storage and a cryptographic unit and that is connected to an ETR.

Specific processors may differ in their internal characteristics, the installed facilities, the number of subchannels, channel paths, and control units which can be attached to the channel subsystem, the size of main and expanded storage, and the representation of the operator facilities.

A system viewed without regard to its I/O devices is referred to as a configuration. All of the physical equipment, whether in the configuration or not, is referred to as the installation.

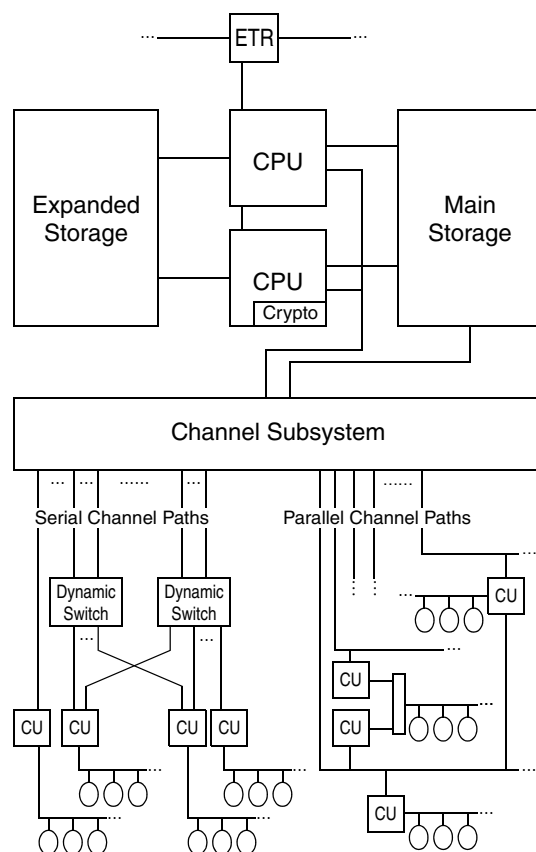


Figure 2-1. Logical Structure of a z/Architecture System with Two CPUs

Model-dependent reconfiguration controls may be provided to change the amount of main and expanded storage and the number of CPUs and channel paths in the configuration. In some instances, the reconfiguration controls may be used to partition a single configuration into multiple configurations. Each of the configurations so reconfigured has the same structure, that is, main and expanded storage, one or more CPUs, and one or more sub-

channels and channel paths in the channel subsystem.

Each configuration is isolated in that the main and expanded storage in one configuration is not directly addressable by the CPUs and the channel subsystem of another configuration. It is, however, possible for one configuration to communicate with another by means of shared I/O devices or a channel-to-channel adapter. At any one time, the storage, CPUs, subchannels, and channel paths connected together in a system are referred to as being in the configuration. Each CPU, subchannel, channel path, main-storage location, and expanded-storage location can be in only one configuration at a time.

Main Storage

Main storage, which is directly addressable, provides for high-speed processing of data by the CPUs and the channel subsystem. Both data and programs must be loaded into main storage from input devices before they can be processed. The amount of main storage available in the system depends on the model, and, depending on the model, the amount in the configuration may be under control of model-dependent configuration controls. The storage is available in multiples of 4K-byte blocks. At any instant, the channel subsystem and all CPUs in the configuration have access to the same blocks of storage and refer to a particular block of main-storage locations by using the same absolute address.

Main storage may include a faster-access buffer storage, sometimes called a cache. Each CPU may have an associated cache. The effects, except on performance, of the physical construction and the use of distinct storage media are not observable by the program.

Expanded Storage

Expanded storage may be available on some models. Expanded storage, when available, can be accessed by all CPUs in the configuration by means of instructions that transfer 4K-byte blocks of data from expanded storage to main storage or from main storage to expanded storage. These instructions are the PAGE IN and PAGE OUT instructions, described

in "PAGE IN" on page 10-64 and "PAGE OUT" on page 10-65.

Each 4K-byte block of expanded storage is addressed by means of a 32-bit unsigned binary integer called an expanded-storage block number.

CPU

The central processing unit (CPU) is the controlling center of the system. It contains the sequencing and processing facilities for instruction execution, interruption action, timing functions, initial program loading, and other machine-related functions.

The physical implementation of the CPU may differ among models, but the logical function remains the same. The result of executing an instruction is the same for each model, providing that the program complies with the compatibility rules.

The CPU, in executing instructions, can process binary integers and floating-point numbers (binary, decimal, and hexadecimal) of fixed length, decimal integers of variable length, and logical information of either fixed or variable length. Processing may be in parallel or in series; the width of the processing elements, the multiplicity of the shifting paths, and the degree of simultaneity in performing the different types of arithmetic differ from one model of CPU to another without affecting the logical results.

Instructions which the CPU executes fall into eight classes: general, decimal, floating-point-support (FPS), binary-floating-point (BFP), decimal-floating-point (DFP), hexadecimal-floating-point (HFP), control, and I/O instructions. The general instructions are used in performing binary-integer-arithmetic operations and logical, branching, and other non-arithmetic operations. The decimal instructions operate on data in the decimal format. The BFP, DFP, and HFP instructions operate on data in the BFP, DFP, and HFP formats, respectively, while the FPS instructions operate on floating-point data independent of the format or convert from one format to another. The privileged control instructions and the I/O instructions can be executed only when the CPU is in the supervisor state; the semiprivileged control instructions can be executed in the problem state, subject to the appropriate authorization mechanisms.

The CPU provides registers which are available to programs but do not have addressable representations in main storage. They include the current program-status word (PSW), the general registers, the floating-point registers and floating-point-control register, the control registers, the access registers, the prefix register, and the registers for the clock comparator and the CPU timer. Each CPU in an installation provides access to a time-of-day (TOD) clock, which is shared by all CPUs in the installation. The instruction operation code determines which type of register is to be used in an operation. See Figure 2-2 on page 2-4 for the format of the control, access, general, and floating-point registers.

PSW

The program-status word (PSW) includes the instruction address, condition code, and other information used to control instruction sequencing and to determine the state of the CPU. The active or controlling PSW is called the current PSW. It governs the program currently being executed.

The CPU has an interruption capability, which permits the CPU to switch rapidly to another program in response to exceptional conditions and external stimuli. When an interruption occurs, the CPU places the current PSW in an assigned storage location, called the old-PSW location, for the particular class of interruption. The CPU fetches a new PSW from a second assigned storage location. This new PSW determines the next program to be executed. When it has finished processing the interruption, the program handling the interruption may reload the old PSW, making it again the current PSW, so that the interrupted program can continue.

There are six classes of interruption: external, I/O, machine check, program, restart, and supervisor call. Each class has a distinct pair of old-PSW and new-PSW locations permanently assigned in real storage.

General Registers

Instructions may designate information in one or more of 16 general registers. The general registers may be used as base-address registers and index registers in address arithmetic and as accumulators in general arithmetic and logical operations. Each

register contains 64 bit positions. The general registers are identified by the numbers 0-15 and are designated by a four-bit R field in an instruction. Some instructions provide for addressing multiple general registers by having several R fields. For some instructions, the use of a specific general register is implied rather than explicitly designated by an R field of the instruction.

For some operations, either bits 32-63 or bits 0-63 of two adjacent general registers are coupled, providing a 64-bit or 128-bit format, respectively. In these operations, the program must designate an even-numbered register, which contains the leftmost (high-order) 32 or 64 bits. The next higher-numbered register contains the rightmost (low-order) 32 or 64 bits.

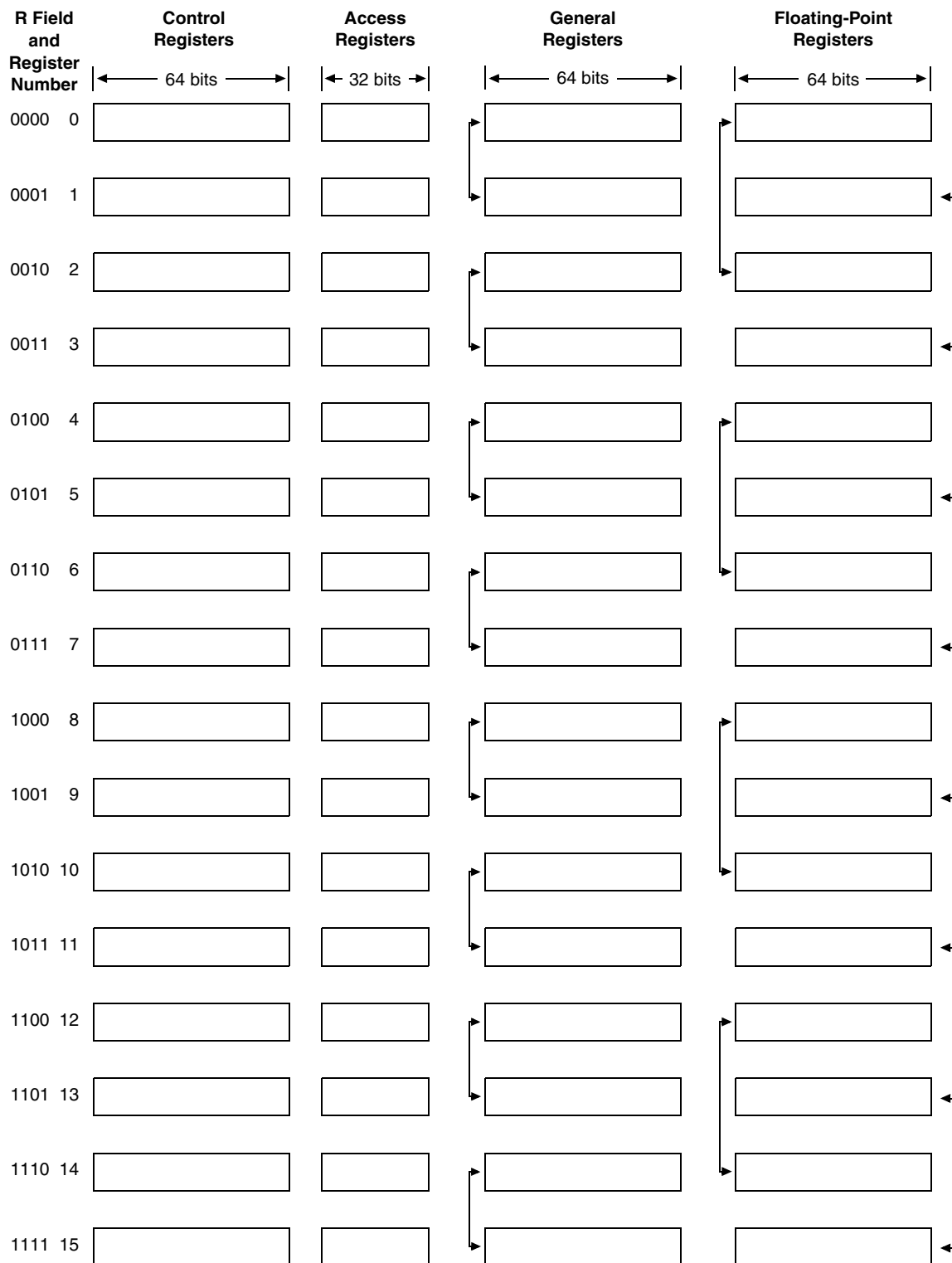
In addition to their use as accumulators in general arithmetic and logical operations, 15 of the 16 general registers are also used as base-address and index registers in address generation. In these cases, the registers are designated by a four-bit B field or X field in an instruction. A value of zero in the B or X field specifies that no base or index is to be applied, and, thus, general register 0 cannot be designated as containing a base address or index.

Floating-Point Registers

All floating-point instructions (FPS, BFP, DFP, and HFP) use the same set of floating-point registers. The CPU has 16 floating-point registers. The floating-point registers are identified by the numbers 0-15 and are designated by a four-bit R field in floating-point instructions. Each floating-point register is 64 bits long and can contain either a short (32-bit) or a long (64-bit) floating-point operand. As shown in Figure 2-2 on page 2-4, pairs of floating-point registers can be used for extended (128-bit) operands. Each of the eight pairs is referred to by the number of the lower-numbered register of the pair.

Floating-Point-Control Register

The floating-point-control (FPC) register is a 32-bit register that contains mask bits, flag bits, a data-exception code, and rounding-mode bits. The FPC register is described in the section "Floating-Point-Control (FPC) Register" on page 9-9.



Note: The arrows indicate that the two registers may be coupled as a double-register pair, designated by specifying the lower-numbered register in the R field. For example, the floating-point register pair 13 and 15 is designated by 1101 binary in the R field.

Figure 2-2. Control, Access, General, and Floating-Point Registers

Control Registers

The CPU has 16 control registers, each having 64 bit positions. The bit positions in the registers are assigned to particular facilities in the system, such as program-event recording, and are used either to specify that an operation can take place or to furnish special information required by the facility.

The control registers are identified by the numbers 0-15 and are designated by four-bit R fields in the instructions LOAD CONTROL and STORE CONTROL. Multiple control registers can be addressed by these instructions.

Access Registers

The CPU has 16 access registers numbered 0-15. An access register consists of 32 bit positions containing an indirect specification (not described here in detail) of an address-space-control element. An address-space-control element is a parameter used by the dynamic-address-translation (DAT) mechanism to translate references to a corresponding address space. When the CPU is in a mode called the access-register mode (controlled by bits in the PSW), an instruction B field, used to specify a logical address for a storage-operand reference, designates an access register, and the address-space-control element specified by the access register is used by DAT for the reference being made. For some instructions, an R field is used instead of a B field. Instructions are provided for loading and storing the contents of the access registers and for moving the contents of one access register to another.

Each of access registers 1-15 can designate any address space, including the current instruction space (the primary address space). Access register 0 always designates the current instruction space. When one of access registers 1-15 is used to designate an address space, the CPU determines which address space is designated by translating the contents of the access register. When access register 0 is used to designate an address space, the CPU treats the access register as designating the current instruction space, and it does not examine the actual contents of the access register. Therefore, the 16 access registers can designate, at any one time, the current instruction space and a maximum of 15 other spaces.

Cryptographic Facility

Depending on the model, an integrated cryptographic facility may be provided as an extension of the CPU. When the cryptographic facility is provided on a CPU, it functions as an integral part of that CPU. A summary of the benefits of the cryptographic facility is given on page 1-23; the facility is otherwise not described.

External Time Reference

Depending on the model, an external time reference (ETR) may be connected to the configuration. A summary of the benefits of the ETR is given on page 1-23; the facility is otherwise not described.

I/O

Input/output (I/O) operations involve the transfer of information between main storage and an I/O device. I/O devices and their control units attach to the channel subsystem, which controls this data transfer.

Channel Subsystem

The channel subsystem directs the flow of information between I/O devices and main storage. It relieves CPUs of the task of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O processing. The channel subsystem uses one or more channel paths as the communication link in managing the flow of information to or from I/O devices. As part of I/O processing, the channel subsystem also performs the path-management function of testing for channel-path availability, selecting an available channel path, and initiating execution of the operation with the I/O device. Within the channel subsystem are subchannels.

One subchannel is provided for and dedicated to each I/O device accessible to the channel subsystem. Each subchannel contains storage for information concerning the associated I/O device and its attachment to the channel subsystem. The subchannel also provides storage for information concerning I/O operations and other functions involving the asso-

ciated I/O device. Information contained in the sub-channel can be accessed by CPUs using I/O instructions as well as by the channel subsystem and serves as the means of communication between any CPU and the channel subsystem concerning the associated I/O device. The actual number of sub-channels provided depends on the model and the configuration; the maximum number of subchannels is 65,536.

Channel Paths

I/O devices are attached through control units to the channel subsystem via channel paths. Control units may be attached to the channel subsystem via more than one channel path, and an I/O device may be attached to more than one control unit. In all, an individual I/O device may be accessible to a channel subsystem by as many as eight different channel paths, depending on the model and the configuration. The total number of channel paths provided by a channel subsystem depends on the model and the configuration; the maximum number of channel paths is 256.

A channel path can use one of three types of communication links:

- System/360 and System/370 I/O interface, called the parallel-I/O interface; the channel path is called a parallel channel path
- ESCON I/O interface, called a serial-I/O interface; the channel path is called a serial channel path
- FICON I/O interface, also called a serial-I/O interface; the channel path again is called a serial channel path

Each parallel-I/O interface consists of a number of electrical signal lines between the channel subsystem and one or more control units. Eight control units can share a single parallel-I/O interface. Up to 256 I/O devices can be addressed on a single parallel-I/O interface. The parallel-I/O interface is described in the publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information*, GA22-6974.

Each serial-I/O interface consists of two optical-fiber conductors between any two of a channel subsystem, a dynamic switch, and a control unit. A dynamic switch can be connected by means of multiple serial-I/O interfaces to either the same or different channel subsystems and to multiple control units. The number of control units which can be connected on one channel path depends on the channel-subsystem and dynamic-switch capabilities. Up to 256 devices can be attached to each control unit that uses the serial-I/O interface, depending on the control unit. The ESCON I/O interface is described in the publication *ESA/390 ESCON I/O Interface*, SA22-7202. The FICON I/O interface is described in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

I/O Devices and Control Units

I/O devices include such equipment as printers, magnetic-tape units, direct-access-storage devices, displays, keyboards, communications controllers, teleprocessing devices, and sensor-based equipment. Many I/O devices function with an external medium, such as paper or magnetic tape. Other I/O devices handle only electrical signals, such as those found in displays and communications networks. In all cases, I/O-device operation is regulated by a control unit that provides the logical and buffering capabilities necessary to operate the associated I/O device. From the programming point of view, most control-unit functions merge with I/O-device functions. The control-unit function may be housed with the I/O device or in the CPU, or a separate control unit may be used.

Operator Facilities

The operator facilities provide the functions necessary for operator control of the machine. Associated with the operator facilities may be an operator-console device, which may also be used as an I/O device for communicating with the program.

The main functions provided by the operator facilities include resetting, clearing, initial program loading, start, stop, alter, and display.

Chapter 3. Storage

Storage Addressing	3-2	ASN-Second-Table Lookup	3-31
Information Formats	3-2	Recognition of Exceptions during ASN	
Integral Boundaries	3-3	Translation	3-31
Address Types and Formats	3-4	ASN Authorization	3-31
Address Types	3-4	ASN-Authorization Controls	3-31
Absolute Address	3-4	Control Register 4	3-31
Real Address	3-4	ASN-Second-Table Entry	3-32
Virtual Address	3-5	Authority-Table Entries	3-32
Primary Virtual Address	3-5	ASN-Authorization Process	3-32
Secondary Virtual Address	3-5	Authority-Table Lookup	3-32
AR-Specified Virtual Address	3-5	Recognition of Exceptions during ASN	
Home Virtual Address	3-5	Authorization	3-34
Logical Address	3-5	Dynamic Address Translation	3-34
Instruction Address	3-5	Translation Control	3-36
Effective Address	3-5	Translation Modes	3-36
Address Size and Wraparound	3-5	Control Register 0	3-37
Address Wraparound	3-6	Control Register 1	3-38
Storage Key	3-8	Control Register 7	3-39
Protection	3-10	Control Register 13	3-40
Key-Controlled Protection	3-10	Translation Tables	3-40
Storage-Protection-Override Control	3-11	Region-Table Entries	3-41
Fetch-Protection-Override Control	3-12	Segment-Table Entries	3-42
Access-List-Controlled Protection	3-12	Page-Table Entries	3-44
DAT Protection	3-12	Translation Process	3-45
Low-Address Protection	3-13	Inspection of Real-Space Control	3-46
Suppression on Protection	3-14	Inspection of Designation-Type Control	3-46
Enhanced Suppression on Protection	3-15	Lookup in a Table Designated by an	
Reference Recording	3-16	Address-Space-Control Element	3-48
Change Recording	3-16	Lookup in a Table Designated by a Region-	
Change-Recording Override	3-17	Table Entry	3-50
Prefixing	3-18	Page-Table Lookup	3-51
Address Spaces	3-18	Formation of the Real and Absolute	
Changing to Different Address Spaces	3-19	Addresses	3-52
Address-Space Number	3-20	Recognition of Exceptions during	
ASN-Second-Table-Entry Sequence Number	3-20	Translation	3-52
ASN-Second-Table-Entry Instance Number		Translation-Lookaside Buffer	3-52
and ASN Reuse	3-21	TLB Structure	3-53
ASN Translation	3-26	Formation of TLB Entries	3-53
ASN-Translation Controls	3-27	Use of TLB Entries	3-54
Control Register 14	3-27	Modification of Translation Tables	3-55
ASN-Translation Tables	3-27	Address Summary	3-58
ASN-First-Table Entries	3-27	Addresses Translated	3-58
ASN-Second-Table Entries	3-27	Handling of Addresses	3-59
ASN-Translation Process	3-29	Assigned Storage Locations	3-61
ASN-First-Table Lookup	3-30		

This chapter discusses the representation of information in main storage, as well as addressing, protec-

tion, and reference and change recording. The aspects of addressing which are covered include the

format of addresses, the concept of address spaces, the various types of addresses, and the manner in which one type of address is translated to another type of address.

A list of permanently assigned storage locations appears at the end of the chapter.

Main storage provides the system with directly addressable fast-access storage of data. Both data and programs must be loaded into main storage (from input devices) before they can be processed.

Main storage may include one or more smaller faster-access buffer storages, sometimes called caches. A cache is usually physically associated with a CPU or an I/O processor. The effects, except on performance, of the physical construction and use of distinct storage media are generally not observable by the program.

Separate caches may be maintained for instructions and for data operands. Information within a cache is maintained in contiguous bytes on an integral boundary called a cache block or cache line (or line, for short). A model may provide the EXTRACT CACHE ATTRIBUTE instruction which returns the size of a cache line in bytes. A model may also provide the PREFETCH DATA and PREFETCH DATA RELATIVE LONG instructions which effect the prefetching of storage into the data cache or the releasing of data from the cache.

Fetching and storing of data by a CPU are not affected by any concurrent channel-subsystem activity or by a concurrent reference to the same storage location by another CPU. When concurrent requests to a main-storage location occur, access normally is granted in a sequence determined by the system. If a reference changes the contents of the location, any subsequent storage fetches obtain the new contents.

Main storage may be volatile or nonvolatile. If it is volatile, the contents of main storage are not preserved when power is turned off. If it is nonvolatile, turning power off and then back on does not affect the contents of main storage, provided all CPUs are in the stopped state and no references are made to main storage when power is being turned off. In both types of main storage, the contents of storage keys are not necessarily preserved when the power for main storage is turned off.

Note: Because most references in this publication apply to virtual storage, the abbreviated term “storage” is often used in place of “virtual storage.” The term “storage” may also be used in place of “main storage,” “absolute storage,” or “real storage” when the meaning is clear. The terms “main storage” and “absolute storage” are used to describe storage which is addressable by means of an absolute address. The terms describe fast-access storage, as opposed to auxiliary storage, such as that provided by direct-access storage devices. “Real storage” is synonymous with “absolute storage” except for the effects of prefixing.

Programming Note: On models that implement separate caches for instructions and data operands, a significant delay may be experienced if the program stores into a cache line from which instructions are subsequently fetched, regardless of whether the store alters the instructions that are subsequently fetched.

Storage Addressing

Storage is viewed as a long horizontal string of bits. For most operations, accesses to storage proceed in a left-to-right sequence. The string of bits is subdivided into units of eight bits. An eight-bit unit is called a byte, which is the basic building block of all information formats.

Each byte location in storage is identified by a unique nonnegative integer, which is the address of that byte location or, simply, the byte address. Adjacent byte locations have consecutive addresses, starting with 0 on the left and proceeding in a left-to-right sequence. Addresses are unsigned binary integers and are 24, 31, or 64 bits. Addresses are described in “Address Size and Wraparound” on page 3-5.

Information Formats

Information is transmitted between storage and a CPU or the channel subsystem one byte, or a group of bytes, at a time. Unless otherwise specified, a group of bytes in storage is addressed by the left-most byte of the group. The number of bytes in the group is either implied or explicitly specified by the operation to be performed. When used in a CPU operation, a group of bytes is called a field.

Within each group of bytes, bits are numbered in a left-to-right sequence. The leftmost bits are sometimes referred to as the “high-order” bits and the rightmost bits as the “low-order” bits. Bit numbers are not storage addresses, however. Only bytes can be addressed. To operate on individual bits of a byte in storage, it is necessary to access the entire byte.

The bits in a byte are numbered 0 through 7, from left to right.

The bits in an address may be numbered 8-31 or 40-63 for 24-bit addresses or 1-31 or 33-63 for 31-bit addresses; they are numbered 0-63 for 64-bit addresses. Within any other fixed-length format of multiple bytes, the bits making up the format are consecutively numbered starting from 0.

For purposes of error detection, and in some models for correction, one or more check bits may be transmitted with each byte or with a group of bytes. Such check bits are generated automatically by the machine and cannot be directly controlled by the program. References in this publication to the length of data fields and registers exclude mention of the associated check bits. All storage capacities are expressed in number of bytes.

When the length of a storage-operand field is implied by the operation code of an instruction, the field is said to have a fixed length, which can be one, two, four, eight, or sixteen bytes. Larger fields may be implied for some instructions.

When the length of a storage-operand field is not implied but is stated explicitly, the field is said to have a variable length. Variable-length operands can vary in length by increments of one byte.

When information is placed in storage, the contents of only those byte locations are replaced that are included in the designated field, even though the width of the physical path to storage may be greater than the length of the field being stored.

Integral Boundaries

Certain units of information must be on an integral boundary in storage. A boundary is called integral for a unit of information when its storage address is a multiple of the length of the unit in bytes. Special names are given to fields of 2, 4, 8, and 16 bytes on an integral boundary. A halfword is a group of two consecutive bytes on a two-byte boundary and is the basic building block of instructions. A word is a group of four consecutive bytes on a four-byte boundary. A doubleword is a group of eight consecutive bytes on an eight-byte boundary. A quadword is a group of 16 consecutive bytes on a 16-byte boundary. (See Figure 3-1 on page 3-4.)

When storage addresses designate halfwords, words, doublewords, and quadwords, the binary representation of the address contains one, two, three, or four rightmost zero bits, respectively.

Instructions must be on two-byte integral boundaries, and CCWs, IDAWs, MIDAWs, and the storage operands of certain instructions must be on other integral boundaries. The storage operands of most instructions do not have boundary-alignment requirements.

Programming Note: For fixed-field-length operations with field lengths that are a power of 2, significant performance degradation is possible when storage operands are not positioned at addresses that are integral multiples of the operand length. To improve performance, frequently used storage operands should be aligned on integral boundaries.

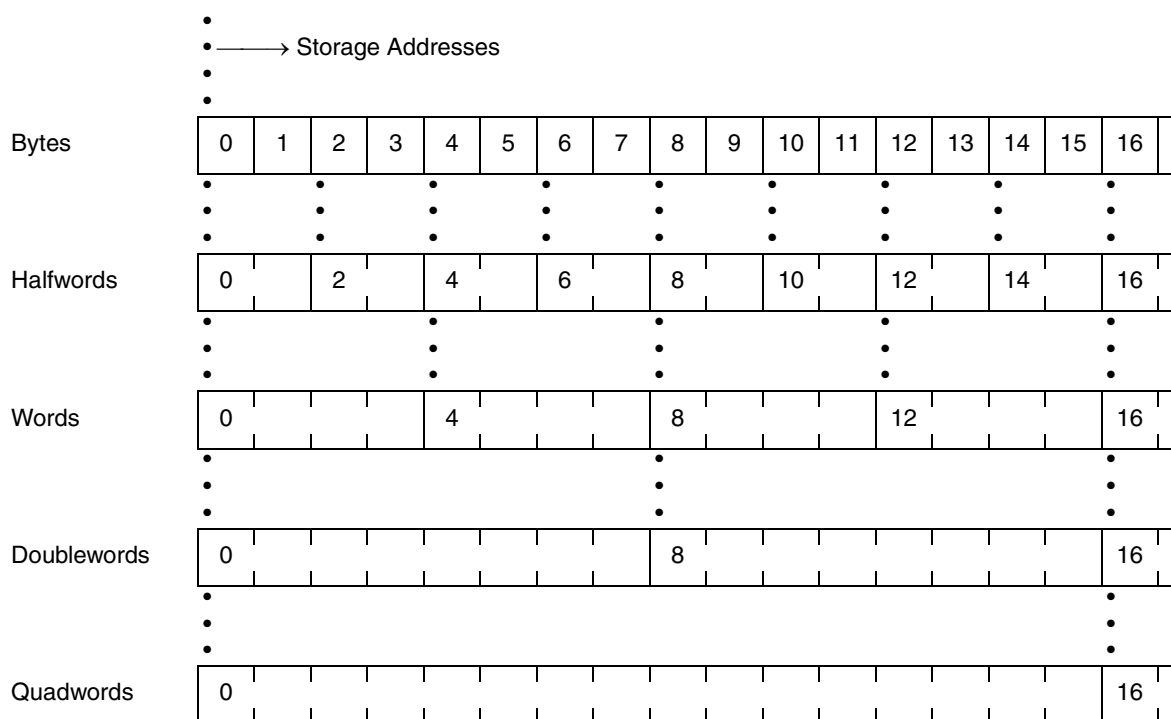


Figure 3-1. Integral Boundaries with Storage Addresses

Address Types and Formats

Address Types

For purposes of addressing main storage, three basic types of addresses are recognized: absolute, real, and virtual. The addresses are distinguished on the basis of the transformations that are applied to the address during a storage access. Address translation converts virtual to real, and prefixing converts real to absolute. In addition to the three basic address types, additional types are defined which are treated as one or another of the three basic types, depending on the instruction and the current mode.

Absolute Address

An absolute address is the address assigned to a main-storage location. An absolute address is used for a storage access without any transformations performed on it.

The channel subsystem and all CPUs in the configuration refer to a shared main-storage location by using the same absolute address. Available main

storage is usually assigned contiguous absolute addresses starting at 0, and the addresses are always assigned in complete 4K-byte blocks on integral boundaries. An exception is recognized when an attempt is made to use an absolute address in a block which has not been assigned to physical locations. On some models, storage-reconfiguration controls may be provided which permit the operator to change the correspondence between absolute addresses and physical locations. However, at any one time, a physical location is not associated with more than one absolute address.

Storage consisting of byte locations sequenced according to their absolute addresses is referred to as absolute storage.

Real Address

A real address identifies a location in real storage. When a real address is used for an access to main storage, it is converted, by means of prefixing, to an absolute address.

At any instant there is one real-address to absolute-address mapping for each CPU in the configuration. When a real address is used by a CPU to access

main storage, it is converted to an absolute address by prefixing. The particular transformation is defined by the value in the prefix register for the CPU.

Storage consisting of byte locations sequenced according to their real addresses is referred to as real storage.

Virtual Address

A virtual address identifies a location in virtual storage. When a virtual address is used for an access to main storage, it is translated by means of dynamic address translation, either (a) to a real address which is then further converted by prefixing to an absolute address, or (b) directly to an absolute address.

Primary Virtual Address

A primary virtual address is a virtual address which is to be translated by means of the primary address-space-control element. Logical addresses are treated as primary virtual addresses when in the primary-space mode. Instruction addresses are treated as primary virtual addresses when in the primary-space mode, secondary-space mode, or access-register mode. The first-operand address of MOVE TO PRIMARY and the second-operand address of MOVE TO SECONDARY are always treated as primary virtual addresses.

Secondary Virtual Address

A secondary virtual address is a virtual address which is to be translated by means of the secondary address-space-control element. Logical addresses are treated as secondary virtual addresses when in the secondary-space mode. The second-operand address of MOVE TO PRIMARY and the first-operand address of MOVE TO SECONDARY are always treated as secondary virtual addresses.

AR-Specified Virtual Address

An AR-specified virtual address is a virtual address which is to be translated by means of an access-register-specified address-space-control element. Logical addresses are treated as AR-specified addresses when in the access-register mode.

Home Virtual Address

A home virtual address is a virtual address which is to be translated by means of the home address-space-control element. Logical addresses and instruction addresses are treated as home virtual addresses when in the home-space mode.

Logical Address

Except where otherwise specified, the storage-operand addresses for most instructions are logical addresses. Logical addresses are treated as real addresses in the real mode, as primary virtual addresses in the primary-space mode, as secondary virtual addresses in the secondary-space mode, as AR-specified virtual addresses in the access-register mode, and as home virtual addresses in the home-space mode. Some instructions have storage-operand addresses or storage accesses associated with the instruction which do not follow the rules for logical addresses. In all such cases, the instruction definition contains a definition of the type of address.

Instruction Address

Addresses used to fetch instructions from storage are called instruction addresses. Instruction addresses are treated as real addresses in the real mode, as primary virtual addresses in the primary-space mode, secondary-space mode, or access-register mode, and as home virtual addresses in the home-space mode. The instruction address in the current PSW and the target address of execute-type instructions (EXECUTE and EXECUTE RELATIVE LONG) are instruction addresses.

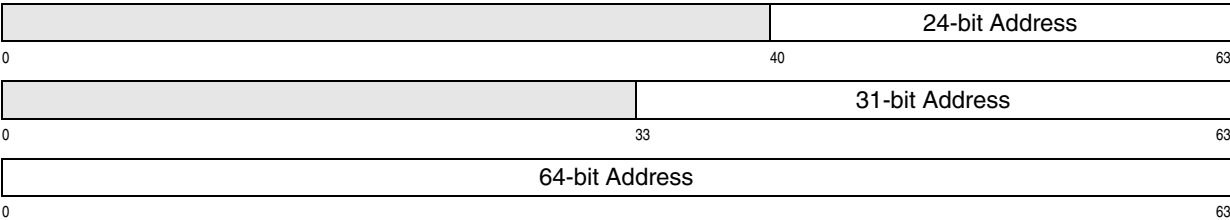
Effective Address

In some situations, it is convenient to use the term "effective address." An effective address is the address which exists before any transformation by dynamic address translation or prefixing is performed. An effective address may be specified directly in a register or may result from address arithmetic. Address arithmetic is the addition of the base and displacement or of the base, index, and displacement.

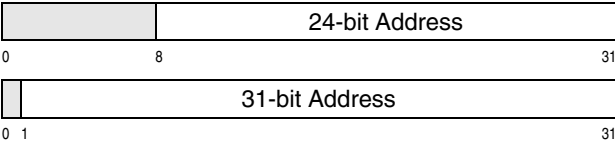
Address Size and Wraparound

An address size refers to the maximum number of significant bits that can represent an address. Three sizes of addresses are provided: 24-bit, 31-bit, and 64-bit. A 24-bit address can accommodate a maximum of 16,777,216 (16M) bytes; with a 31-bit address, 2,147,483,648 (2G) bytes can be addressed; and, with a 64-bit address, 18,446,744,073,709,551,616 (16E) bytes can be addressed. The bits of a 24-bit, 31-bit, or 64-bit address produced by address arithmetic under the control of the current addressing mode are numbered 40-63, 33-63, and 0-63, respectively, corre-

sponding to the numbering of base-address and index bits in a general register:



The bits of an address that is 31 bits regardless of the addressing mode are numbered 1-31, and, when a 24-bit or 31-bit address is contained in a four-byte field in storage, the bits are numbered 8-31 or 1-31, respectively:



A 24-bit or 31-bit virtual address is expanded to 64 bits by appending 40 or 33 zeros, respectively, on the left before it is translated by means of the DAT process, and a 24-bit or 31-bit real address is similarly expanded to 64 bits before it is transformed by prefixing. A 24-bit or 31-bit absolute address is expanded to 64 bits before main storage is accessed. Thus, the 24-bit address always designates a location in the first 16M-byte block of the 16E-byte storage addressable by a 64-bit address, and the 31-bit address always designates a location in the first 2G-byte block.

Unless specifically stated to the contrary, the following definition applies in this publication: whenever the machine generates and provides to the program a 24-bit or 31-bit address, the address is made available (placed in storage or loaded into a general register) by being imbedded in a 32-bit field, with the leftmost eight bits or one bit in the field, respectively, set to zeros. When the address is loaded into a general register, bits 0-31 of the register remain unchanged.

The size of effective addresses is controlled by bits 31 and 32 of the PSW, the extended-addressing-mode bit and the basic-addressing-mode bit, respectively. When bits 31 and 32 are both zero, the CPU is in the 24-bit addressing mode, and 24-bit operand and instruction effective addresses are specified.

When bit 31 is zero and bit 32 is one, the CPU is in the 31-bit addressing mode, and 31-bit operand and instruction effective addresses are specified. When bits 31 and 32 are both one, the CPU is in the 64-bit addressing mode, and 64-bit operand and instruction effective addresses are specified (see "Address Generation" on page 5-8).

The sizes of the real or absolute addresses used or yielded by the ASN-translation, ASN-authorization, PC-number-translation, and access-register-translation processes are always 31 bits regardless of the current addressing mode. Similarly, the sizes of the real or absolute addresses used or yielded by the DAT, stacking, unstacking, and tracing processes are always 64 bits.

The size of the data address in a CCW is under control of the CCW-format-control bit in the operation-request block (ORB) designated by a START SUB-CHANNEL instruction. The CCWs with 24-bit and 31-bit addresses are called format-0 and format-1 CCWs, respectively. Format-0 and format-1 CCWs are described in "Basic I/O Functions" on page 15-1. Similarly, the size of the data address in an IDAW is under control of the IDAW-format-control bit in the ORB. The IDAWs with 31-bit and 64-bit addresses are called format-1 and format-2 IDAWs, respectively. MIDAWs contain 64-bit data addresses. IDAWs and MIDAWs are described in Chapter 15, "Basic I/O Functions."

Address Wraparound

The CPU performs address generation when it forms an operand or instruction address or when it generates the address of a table entry from the appropriate table origin and index. It also performs address generation when it increments an address to access successive bytes of a field. Similarly, the channel subsystem performs address generation when it increments an address (1) to fetch a CCW, (2) to fetch an IDAW, (3) to fetch a MIDAW, (4) to transfer

data, or (5) to compute the address of an I/O measurement block.

When, during the generation of the address, an address is obtained that exceeds the value allowed for the address size ($2^{24} - 1$, $2^{31} - 1$, or $2^{64} - 1$), one of the following two actions is taken:

1. The carry out of the high-order bit position of the address is ignored. This handling of an address of excessive size is called *wraparound*.
2. An interruption condition is recognized.

The effect of wraparound is to make an address space appear circular; that is, address 0 appears to follow the maximum allowable address. Address arithmetic and wraparound occur before transformation, if any, of the address by DAT or prefixing.

Addresses generated by the CPU that may be virtual addresses always wrap. Wraparound also occurs when the linkage-stack-entry address in control register 15 is decremented below 0 by PROGRAM RETURN. For CPU table entries that are addressed by real or absolute addresses, it is unpredictable whether the address wraps or an addressing exception is recognized.

For channel-program execution, when the generated address exceeds the value for the address size (or, for the read-backward command is decremented below 0), an I/O program-check condition is recognized.

Figure 3-2 on page 3-7 identifies what limit values apply to the generation of different addresses and how addresses are handled when they exceed the allowed value.

Address Generation for	Address Type	Handling when Address Would Wrap
Instructions and operands when EAM and BAM are zero	L,I,R,V	W24
Successive bytes of instructions and operands when EAM and BAM are zero	I,L,V ¹	W24
Instructions and operands when EAM is zero and BAM is one	L,I,R,V	W31
Successive bytes of instructions and operands when EAM is zero and BAM is one	I,L,V ¹	W31
Instructions and operands when EAM and BAM are one	L,I,R,V	W64
Successive bytes of instructions and operands when EAM and BAM are one	I,L,V ¹	W64
DAT-table entries when used for implicit translation or for LPTEA, LRA, LRAg, or STRAG	A or R ²	X64
ASN-second-table, authority-table (during ASN authorization), linkage-table, linkage-first-table, linkage-second-table, and entry-table entries	R	X31
Authority-table (during access-register translation) and access-list entries	A or R ²	X31
Linkage-stack entry	V	W64
I/O measurement block	A	P31
For a channel program with format-0 CCWs:		
Successive CCWs	A	P24
Successive IDAWs	A	P24
Successive MIDAWs	A	P24
Successive bytes of I/O data (without IDAWs and MIDAWs)	A	P24
Successive bytes of I/O data (with format-1 IDAWs)	A	P31

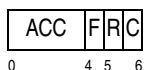
Figure 3-2. Address Wraparound (Part 1 of 2).

Address Generation for	Address Type	Handling when Address Would Wrap
Successive bytes of I/O data (with format-2 IDAWs)	A	P64
Successive bytes of I/O data (with MIDAWs)	A	P64
For a channel program with format-1 CCWs:		
Successive CCWs	A	P31
Successive IDAWs	A	P31
Successive MIDAWs	A	P31
Successive bytes of I/O data (without IDAWs and MIDAWs)	A	P31
Successive bytes of I/O data (with format-1 IDAWs)	A	P31
Successive bytes of I/O data (with format-2 IDAWs)	A	P64
Successive bytes of I/O data (with MIDAWs)	A	P64
Explanation: ¹ Real addresses do not apply in this case since the instructions which designate operands by means of real addresses cannot designate operands that cross boundary 2^{24} , 2^{31} , 2^{64} . ² It is unpredictable whether the address is absolute or real. A Absolute address. BAM Basic-addressing-mode bit in the PSW. EAM Extended-addressing-mode bit in the PSW. I Instruction address. L Logical address. P24 An I/O program-check condition is recognized when the address exceeds $2^{24} - 1$ or is decremented below zero. P31 An I/O program-check condition is recognized when the address exceeds $2^{31} - 1$ or is decremented below zero. P64 An I/O program-check condition is recognized when the address exceeds $2^{64} - 1$ or is decremented below zero. R Real address. V Virtual address. W24 Wrap to location 0 after location $2^{24} - 1$ and vice versa. W31 Wrap to location 0 after location $2^{31} - 1$ and vice versa. W64 Wrap to location 0 after location $2^{64} - 1$ and vice versa. X31 When the address exceeds $2^{31} - 1$, it is unpredictable whether the address wraps to location 0 after location $2^{31} - 1$ or whether an addressing exception is recognized. X64 When the address exceeds $2^{64} - 1$, it is unpredictable whether the address wraps to location 0 after location $2^{64} - 1$ or whether an addressing exception is recognized.		

Figure 3-2. Address Wraparound (Part 2 of 2).

Storage Key

A storage key is associated with each 4K-byte block of storage that is available in the configuration. The storage key has the following format:



The bit positions in the storage key are allocated as follows:

Access-Control Bits (ACC): If a reference is subject to key-controlled protection, the four access-control bits, bits 0-3, are matched with the four-bit access key when information is stored and when information is fetched from a location that is protected against fetching.

Fetch-Protection Bit (F): If a reference is subject to key-controlled protection, the fetch-protection bit, bit 4, controls whether key-controlled protection applies to fetch-type references: a zero indicates that only store-type references are monitored and that fetching with any access key is permitted; a one indicates that key-controlled protection applies to both fetching and storing. No distinction is made between the fetching of instructions and of operands.

Reference Bit (R): The reference bit, bit 5, normally is set to one each time a location in the corresponding storage block is referred to either for storing or for fetching of information.

Change Bit (C): The change bit, bit 6, is set to one each time information is stored at a location in the corresponding storage block.

Storage keys are not part of addressable storage. The entire storage key may be set by SET STORAGE KEY EXTENDED and inspected by INSERT STORAGE KEY EXTENDED. RESET REFERENCE BIT EXTENDED provides a means of inspecting the reference and change bits and of setting the reference bit to zero. RESET REFERENCE BITS MULTIPLE provides a means of inspecting the reference bits of 64 contiguous 4 K-byte blocks and setting the reference bits to zero. Bits 0-4 of the storage key are inspected by the INSERT VIRTUAL STORAGE KEY instruction. The contents of the storage key are unpredictable during and after the execution of the usability test of the TEST BLOCK instruction. When the conditional SSKE facility is installed, SET STORAGE KEY EXTENDED may be used to set all or portions of a storage key based on program-specified criteria. When the enhanced-DAT facility is installed, the SET STORAGE KEY EXTENDED or PERFORM FRAME MANAGEMENT FUNCTION instructions may be used to set all or portions of one or more storage keys based on program-specified criteria.

When enhanced DAT applies,¹ and the invalid bit of the segment-table entry used in the translation is zero, the following additional conditions are in effect:

- When the STE-format control (FC, bit 53 of the segment-table entry used during a translation) is zero, bit 55 of the page-table entry used during translation is the change-recording override (CO) for the page. When the CO bit in the page-table

entry is one, change recording is unpredictable for any store operations to the page.

- When the format-control bit (FC) in the segment-table entry is one, the following applies:
 - Bit position 47 of the segment-table entry contains the ACCF-validity control. The ACCF-validity control determines the validity of the access-control and fetch-protection bits in the STE. When the ACCF-validity control is zero, key-controlled protection uses the access-control and fetch protection bits in the storage key for the 4K-byte block corresponding to the address.
 - When the ACCF-validity control is one, bit positions 48-52 of the segment-table entry contain the access-control bits and the fetch-protection bit for the segment. When determining accessibility to a storage operand, it is unpredictable whether bits 48-52 of the STE or bits 0-4 of the individual storage keys for the 4K-byte blocks composing the segment are examined. See “Translation Process” on page 3-45 for further details.
 - Bit 55 of the segment-table entry is the change-recording override (CO) for the segment. When the CO bit in the segment-table entry is one, it is unpredictable whether the change bit is set for any store operations to the segment.

Programming Notes:

1. When enhanced DAT applies, and both the ACCF-validity control and the STE-format control (bits 47 and 53 of the segment-table entry, respectively) are one, bits 48-52 of the STE contain access-control and fetch protection bits which may be used in lieu of the corresponding bits in the storage keys. In this case, the program is responsible for ensuring that the access-control and fetch-protection bits in the STE are identical to the corresponding bits in each of the 256 storage keys for the segment. See “Modification of Translation Tables” on page 3-55 for restrictions on modifying the STE or the storage keys when the STE-format control is one.
2. When enhanced DAT applies and the STE format control is zero, the change-recording override in the PTE controls whether change recording for

1. See “Enhanced-DAT Terminology:” on page 3-37 for an explanation of the term “enhanced-DAT applies.”

the page's 4K-byte block may be bypassed. When enhanced DAT applies and the STE-format control is one, the change-recording override in the STE controls whether change recording for any of the segment's 256 4K-byte blocks may be bypassed. See "Modification of Translation Tables" on page 3-55 for restrictions on modifying the change-recording override in either the STE or the PTE.

Protection

Four protection facilities are provided to protect the contents of main storage from destruction or misuse by programs that contain errors or are unauthorized: key-controlled protection, access-list-controlled protection, DAT protection, and low-address protection. The protection facilities are applied independently; access to main storage is only permitted when none of the facilities prohibits the access.

Key-controlled protection affords protection against improper storing or against both improper storing and fetching, but not against improper fetching alone.

Key-Controlled Protection

When key-controlled protection applies to a storage access, a store is permitted only when the storage key matches the access key associated with the request for storage access; a fetch is permitted when the keys match or when the fetch-protection bit of the storage key is zero.

The keys are said to match when the four access-control bits of the storage key are equal to the access key, or when the access key is zero.

The protection action is summarized in Figure 3-3.

Conditions		Is Access to Storage Permitted	
Fetch-Protection Bit of Storage Key	Key Relation	Fetch	Store
0	Match	Yes	Yes
0	Mismatch	Yes	No
1	Match	Yes	Yes
1	Mismatch	No	No

Figure 3-3. Summary of Protection Action

Conditions		Is Access to Storage Permitted	
Fetch-Protection Bit of Storage Key	Key Relation	Fetch	Store
Explanation:			
Match	The four access-control bits of the storage key are equal to the access key, or the access key is zero.		
Yes	Access is permitted.		
No	Access is not permitted. On fetching, the information is not made available to the program; on storing, the contents of the storage location are not changed.		

Figure 3-3. Summary of Protection Action

When the access to storage is initiated by the CPU and key-controlled protection applies, the PSW key is the access key, except that the access key is specified in a general register for the first operand of MOVE TO SECONDARY and MOVE WITH DESTINATION KEY, for the second operand of MOVE TO PRIMARY, MOVE WITH KEY, and MOVE WITH SOURCE KEY, and for either the first or the second operand of MOVE PAGE. The access key may be the PSW key or the key specified in an operand-access control for either operand of MOVE WITH OPTIONAL SPECIFICATIONS. The PSW key occupies bit positions 8-11 of the current PSW.

When the access to storage is for the purpose of channel-program execution, the subchannel key associated with that channel program is the access key. The subchannel key for a channel program is specified in the operation-request block (ORB). When, for purposes of channel-subsystem monitoring, an access to the measurement block is made, the measurement-block key is the access key. The measurement-block key is specified by the SET CHANNEL MONITOR instruction. Even when the enhanced-DAT facility is installed, channel subsystem accesses continue to reference the storage keys for each 4K-byte block; because channel subsystem accesses are not subject to dynamic address translation, the access-control bits and fetch-protection bit in the segment-table entry are not used by the channel subsystem.

When a CPU access is prohibited because of key-controlled protection, the execution of the instruction is terminated, and a program interruption for a protection exception takes place. However, the unit of operation or the execution of the instruction may be suppressed, as described in the section "Suppression on Protection" on page 3-14. When a channel-

program access is prohibited, the start function is ended, and the protection-check condition is indicated in the associated interruption-response block (IRB). When a measurement-block access is prohibited, the I/O measurement-block protection-check condition is indicated.

When a store access is prohibited because of key-controlled protection, the contents of the protected location remain unchanged. When a fetch access is prohibited, the protected information is not loaded into a register, moved to another storage location, or provided to an I/O device. For a prohibited instruction fetch, the instruction is suppressed, and an arbitrary instruction-length code is indicated.

Key-controlled protection is independent of whether the CPU is in the problem or the supervisor state and, except as described below, does not depend on the type of CPU instruction or channel-command word being executed.

Except where otherwise specified, all accesses to storage locations that are explicitly designated by the program and that are used by the CPU to store or fetch information are subject to key-controlled protection.

Key-controlled protection does not apply when the storage-protection-override control is one and the value of the four access-control bits of the storage key is 9. Key-controlled protection for fetches may or may not apply when the fetch-protection-override control is one, depending on the effective address and the private-space control.

The storage-protection-override control and fetch-protection-override control do not affect storage references made by the channel subsystem.

Accesses to the second operand of TEST BLOCK are not subject to key-controlled protection.

All storage accesses by the channel subsystem to access the I/O measurement block, or by a channel program to fetch a CCW, IDAW, or MIDAW or to access a data area designated during the execution of a CCW, are subject to key-controlled protection. However, if a CCW, an IDAW, a MIDAW, or output data is prefetched, a protection check is not indicated until the CCW, IDAW, or MIDAW is due to take control or until the data is due to be written.

Key-controlled protection is not applied to accesses that are implicitly made for any of such sequences as:

- An interruption
- CPU logout
- Fetching of table entries for access-register translation, dynamic-address translation, PC-number translation, ASN translation, or ASN authorization
- Tracing
- A store-status function
- Storing in real locations 184-191 when TEST PENDING INTERRUPTION has an operand address of zero
- Initial program loading

Similarly, protection does not apply to accesses initiated via the operator facilities for altering or displaying information. However, when the program explicitly designates these locations, they are subject to protection.

Storage-Protection-Override Control

Bit 39 of control register 0 is the storage-protection-override control. When this bit is one, storage-protection override is active. When this bit is zero, storage-protection override is inactive. When storage-protection override is active, key-controlled storage protection is ignored for storage locations having an associated storage-key value of 9. When storage-protection override is inactive, no special action is taken for a storage-key value of 9.

Storage-protection override applies to instruction fetch and to the fetch and store accesses of instructions whose operand addresses are logical, virtual, or real. It does not apply to accesses made for the purpose of channel-program execution or for the purpose of channel-subsystem monitoring.

Storage-protection override has no effect on accesses which are not subject to key-controlled protection.

Programming Notes:

1. Storage-protection override can be used to improve reliability in the case when a possibly erroneous application program is executed in conjunction with a reliable subsystem, provided that the application program needs to access only a portion of the storage accessed by the

subsystem. The technique for doing this is as follows. The storage accessed by the application program is given storage key 9. The storage accessed by only the subsystem is given some other nonzero storage key, for example, key 8. The application is executed with PSW key 9. The subsystem is executed with PSW key 8 (in this example). As a result, the subsystem can access both the key-8 and the key-9 storage, while the application program can access only the key-9 storage.

2. Storage-protection override affects the accesses to storage made by the CPU and also affects the result set by TEST PROTECTION. However, those instructions which, in the problem state, test the PSW-key mask to determine if a particular key value may be used are not affected by whether storage-protection override is active. These instructions include, among others, MOVE WITH KEY and SET PSW KEY FROM ADDRESS. To permit these instructions to use an access key of 9 in the problem state, bit 9 of the PSW-key mask must be one.

Fetch-Protection-Override Control

Bit 38 of control register 0 is the fetch-protection-override control. When the bit is one, fetch protection is ignored for locations at effective addresses 0-2047. An effective address is the address which exists before any transformation by dynamic address translation or prefixing. However, fetch protection is not ignored if the effective address is subject to dynamic address translation and the private-space control, bit 55, is one in the address-space-control element used in the translation.

Fetch-protection override applies to instruction fetch and to the fetch accesses of instructions whose operand addresses are logical, virtual, or real. It does not apply to fetch accesses made for the purpose of channel-program execution or for the purpose of channel-subsystem monitoring. When this bit is set to zero, fetch protection of locations at effective addresses 0-2047 is determined by the state of the fetch-protection bit of the storage key associated with those locations.

Fetch-protection override has no effect on accesses which are not subject to key-controlled protection.

Programming Note: The fetch-protection-override control allows fetch protection of locations at addresses 2048-4095 along with no fetch protection of locations at addresses 0-2047.

Access-List-Controlled Protection

In the access-register mode, bit 6 of the access-list entry, the fetch-only bit, controls which types of operand references are permitted to the address space specified by the access-list entry. When the entry is used in the access-register-translation part of a reference and bit 6 is zero, both fetch-type and store-type references are permitted; when bit 6 is one, only fetch-type references are permitted, and an attempt to store causes a protection exception to be recognized and the execution of the instruction to be suppressed.

The fetch-only bit is included in the ALB access-list entry. A change to the fetch-only bit in an access-list entry in main storage does not necessarily have an immediate, if any, effect on whether a protection exception is recognized. However, this change to the bit will have an effect immediately after PURGE ALB or a COMPARE AND SWAP AND PURGE instruction that purges the ALB is executed.

TEST PROTECTION takes into consideration access-list-controlled protection when the CPU is in the access-register mode. A violation of access-list-controlled protection causes condition code 1 to be set, except that it does not prevent condition code 2 or 3 from being set when the conditions for those codes are satisfied.

Programming Note: A violation of access-list-controlled protection always causes suppression. A violation of any of the other protection types may cause termination.

DAT Protection

The DAT-protection function² controls access to virtual storage by using the DAT-protection bit in each page-table entry and segment-table entry, and, when the enhanced-DAT facility is installed, in each region-table entry. It provides protection against improper storing.

2. Prior to the enhanced-DAT facility, the DAT-protection function was known as the page-protection facility.

The DAT-protection bit, bit 54 of the page-table entry, controls whether storing is allowed into the corresponding 4K-byte page. When the bit is zero, both fetching and storing are permitted; when the bit is one, only fetching is permitted. When an attempt is made to store into a protected page, the contents of the page remain unchanged, the unit of operation or the execution of the instruction is suppressed, and a program interruption for protection takes place.

The DAT-protection bit, bit 54 of the segment-table entry, controls whether storing is allowed into the corresponding 1M-byte segment, as follows:

- When enhanced DAT does not apply, or when enhanced DAT applies and the STE-format control is zero, the DAT-protection bit of the segment-table entry is treated as being ORed into the DAT-protection-bit position of each entry in the page table designated by the segment-table entry. Thus, when the segment-table-entry DAT-protection bit is one, the effect is as if the DAT-protection bit were one in each entry in the designated page table.
- When enhanced DAT applies and the STE-format control is one, the DAT-protection bit of the segment-table entry controls whether storing is allowed into the corresponding 1M-byte segment. When the bit is zero, both fetching and storing are permitted; when the bit is one, only fetching is permitted. When an attempt is made to store into a protected segment, the contents of the segment remain unchanged, the unit of operation or the execution of the instruction is suppressed, and a program interruption for protection takes place

When enhanced DAT applies, the DAT-protection bit, bit 54 of the region-table entry, controls whether storing is allowed into the corresponding region(s). The DAT-protection bit in a region-table entry is treated as being ORed into the DAT-protection bit position of any subsequent region-table entry and segment-table entry that is used in the translation. When the STE-format control bit is zero, the DAT-protection bit is further propagated to the page-table entry, as described above.

DAT protection applies to all store-type references that use a virtual address.

Low-Address Protection

The low-address-protection facility provides protection against the destruction of main-storage information used by the CPU during interruption processing. This is accomplished by prohibiting instructions from storing with effective addresses in the ranges 0 through 511 and 4096 through 4607 (the first 512 bytes of each of the first and second 4K-byte effective-address blocks). The range criterion is applied before address transformation, if any, of the address by dynamic address translation or prefixing. However, the range criterion is not applied, with the result that low-address protection does not apply, if the effective address is subject to dynamic address translation and the private-space control, bit 55, is one in the address-space-control element used in the translation. Low-address protection does not apply if the address-space-control element to be used is not available due to another type of exception.

Low-address protection is under control of bit 35 of control register 0, the low-address-protection-control bit. When the bit is zero, low-address protection is off; when the bit is one, low-address protection is on.

If an access is prohibited because of low-address protection, the contents of the protected location remain unchanged, the execution of the instruction is terminated, and a program interruption for a protection exception takes place. However, the unit of operation or the execution of the instruction may be suppressed, as described in the section "Suppression on Protection" on page 3-14.

Any attempt by the program to store by using effective addresses in the range 0 through 511 or 4096 through 4607 is subject to low-address protection. Low-address protection is applied to the store accesses of instructions whose operand addresses are logical, virtual, real, or absolute. Low-address protection is also applied to the trace table.

Low-address protection is not applied to accesses made by the CPU or the channel subsystem for such sequences as interruptions, CPU logout, the storing of the I/O-interruption code in real locations 184-195 by TEST PENDING INTERRUPTION, the storing of the results of STORE FACILITY LIST in real locations 200-203, and the initial-program-loading and store-status functions, nor is it applied to data stores during I/O data transfer. When the enhanced-monitor facility is installed, low-address protection is not applied

when a monitor-event counting operation results in the updating the enhanced-monitor exception count in real locations 268-271. However, explicit stores by a program at any of these locations are subject to low-address protection.

Programming Notes:

1. Low-address protection and key-controlled protection apply to the same store accesses, except that:
 - a. Low-address protection does not apply to storing performed by the channel subsystem, whereas key-controlled protection does.
 - b. Key-controlled protection does not apply to tracing, the second operand of TEST BLOCK, or instructions that operate specifically on the linkage stack, whereas low-address protection does.
2. Because fetch-protection override and low-address protection do not apply to an address space for which the private-space control is one in the address-space-control element, locations 0-2047 and 4096-4607 in the address space are usable the same as the other locations in the space.

Suppression on Protection

Some instruction definitions specify that the operation is always suppressed if a protection exception due to any type of protection is recognized. When that specification is absent, the execution of an instruction is always suppressed if a protection exception due to access-list-controlled protection or DAT protection is recognized, and it may be either suppressed or terminated if a protection exception due to low-address protection or key-controlled protection is recognized.

The suppression-on-protection function allows the control program to locate the segment-table entry, page-table entry, and, when enhanced DAT applies, the region-table entry used in the translation of a virtual address that caused a protection exception, in order to determine if the exception was due to DAT protection.³ This is necessary for the implementation of the POSIX fork function (discussed in a program-

ming note). The function also allows the control program to avoid locating the segment-table and page-table entries if the address was not virtual or the exception was due to access-list-controlled protection; when enhanced DAT applies, the control program may also avoid locating the region-table entries if the address was not virtual or the exception was due to access-list-controlled protection.

During a program interruption due to a protection exception, either a one or a zero is stored in bit position 61 of real locations 168-175. The storing of a one in bit position 61 indicates that:

- The unit of operation or instruction execution during which the exception was recognized was suppressed.
- If dynamic address translation (DAT) was on, as indicated by the DAT-mode bit in the program old PSW, the effective address that caused the exception is one that was to be translated by DAT. (The effective address is the address which exists before any transformation by DAT or prefixing.) Bit 61 is set to zero if DAT was on but the effective address was not to be translated by DAT because it is a real address. If DAT was off, the protection exception cannot have been due to DAT protection or access-list-controlled protection.
- Bit positions 0-51 of real locations 168-175 contain bits 0-51 of the effective address that caused the exception. The contents of bit positions 52-59 of real locations 168-175 are always unpredictable.

If DAT was on, indicating that the effective address was to be translated by DAT, bit positions 62 and 63 of real locations 168-175, and real location 160, contain the same information as is stored during a program interruption due to a page-translation exception — this information identifies the address space containing the protected address. Also, bit 60 of real locations 168-175 is zero if the protection exception was not due to access-list-controlled protection or is one if the exception was due to access-list-controlled protection. A one in bit position 60 indicates that the exception was not due to DAT protection. If DAT was off, the contents of bit positions 60, 62, and 63 of real locations

3. The suppression-on-protection function originated as the ESA/390 suppression-on-protection facility. Suppression for page protection (now called DAT protection) was new as part of that facility.

168-175, and the contents of real location 160, are unpredictable.

Bit 61 being zero indicates that the operation was either suppressed or terminated and that the contents of the remainder of real locations 168-175, and of real location 160 are unpredictable.

Bit 61 is set to one if the protection exception was due to access-list-controlled protection or DAT protection. Bit 61 may be set to one if the protection exception was due to low-address protection or key-controlled protection.

If a protection-exception condition exists due to either access-list-controlled protection or DAT protection but also exists due to either low-address protection or key-controlled protection, it is unpredictable whether bit 61 is set to zero or one.

Programming Notes:

1. The suppression-on-protection function is useful in performing the POSIX fork function, which causes a duplicate address space to be created. The following discussion pertains to when enhanced DAT does not apply, or when enhanced DAT applies but the format-control (FC) bit in the segment-table entry is zero. When forking occurs, the control program causes the same page of different address spaces to map to a single page frame of real storage as long as a store in the page is not attempted. Then, when a store is attempted in a particular address space, the control program assigns a unique page frame to the page in that address space and copies the contents of the page to the new page frame. This last action is sometimes called the copy-on-write function. The control program sets the DAT-protection bit to one in the page-table entry for a page in order to detect an attempt to store in the page. The control program may initially set the DAT-protection bit to one in a segment-table entry to detect an attempt to store anywhere in the specified segment.

When enhanced DAT applies, and the format-control (FC) bit in the segment-table entry is one, a similar technique may be used to map a single segment frame of absolute storage.

2. Bit 61 being one in real locations 168-175 when DAT was on indicates that the address that caused a protection exception is virtual. This

indication allows programmed forms of access-register translation and dynamic address translation to be performed to determine whether the exception was due to DAT protection as opposed to low-address or key-controlled protection.

3. The results of suppression on protection are summarized in Figure 3-4.

Exception Conditions				Presented Fields		
LA or Key-Cont. Prot.	DAT	ALC or DAT Prot.	Eff. Addr.	Bit 61	If Bit 61 is One	
					Bits 62, 63 and Loc. 160	Bit 60
No	On	Yes	Log.	1	P	1A
Yes	On	Yes	Log.	U1	P	1A
Yes	Off	No	Log.	U2	U3	U3
Yes	Off	No	R/A	U2	U3	U3
Yes	On	No	Log.	U2	P	0
Yes	On	No	R/A	0R	—	—

Explanation:

— Immaterial or not applicable.

0R Zero because effective address is real.

1A One if bit 61 is set to one because of access-list-controlled protection; zero otherwise.

ALC Access-list-controlled.

LA Low-address.

Log. Logical.

P Predictable.

R/A Real or absolute

U1 Unpredictable because low-address or key-controlled protection may be recognized instead of access-list-controlled or DAT protection.

U2 Unpredictable because bit 61 is only required to be set to one for access-list-controlled or DAT protection.

U3 Unpredictable because DAT is off.

Figure 3-4. Suppression-on-Protection Results

Enhanced Suppression on Protection

When the enhanced suppression-on-protection function is installed, there are the following additional constraints on what may occur during a protection exception. These constraints take precedence over any constraints defined in the original suppression-on-protection function.

During a program interruption due to a protection exception, bit 61 of real locations 168-175 indicates the type of protection exception recognized. Bit 61 is set to one if the protection exception was due to access-list-controlled protection or DAT protection. Bit 61 is set to zero if the protection exception was due to low-address protection or key-controlled protection.

If a protection-exception condition exists due to either access-list-controlled protection or DAT protection but also exists due to either low-address protection or key-controlled protection, it is unpredictable which exception is recognized and whether bit 61 is set to zero or one. However, while it is unpredictable which exception is recognized, the recognized exception will produce consistent behavior as summarized in Figure 3-5.

Exception Type	DAT	Bit 61	Bit 60	Bits 62, 63 and Loc. 160	Bits 0-51
LAP	Any	0	—	—	—
KCP	Any	0	—	—	—
ALCP	On	1	1	AS	A
DATP	On	1	0	AS	A
Explanation:					
A	Bits 0-51 of the effective address that caused the exception.				
ALCP	Access-list-controlled protection.				
AS	Identifies the address space containing the effective address that caused the exception.				
DATP	DAT protection.				
KCP	Key-controlled protection.				
LAP	Low-address protection.				
—	Undefined.				

Figure 3-5. Enhanced Suppression-on-Protection Results.

When the access-exception-fetch/store-indication facility is installed, bits 52 and 53 of real locations 168-175 are handled in the same manner as described in “Access-Exception Fetch/Store Indication:” on page 3-63. When the access-exception-fetch/store-indication facility is not installed, the contents of bit positions 52 and 53 are unpredictable.

Reference Recording

Reference recording provides information for use in selecting pages for replacement. Reference recording uses the reference bit, bit 5 of the storage key.

The reference bit is set to one each time a location in the corresponding storage block is referred to either for fetching or for storing information, regardless of whether DAT is on or off.

Reference recording is always active and takes place for all storage accesses, including those made by any CPU, any operator facility, or the channel subsystem. It takes place for implicit accesses made by the machine, such as those which are part of interruptions and I/O-instruction execution.

Reference recording does not occur for operand accesses of the following instructions since they directly refer to a storage key without accessing a storage location:

- INSERT STORAGE KEY EXTENDED
- RESET REFERENCE BIT EXTENDED (reference bit is set to zero)
- RESET REFERENCE BITS MULTIPLE (reference bits of 64 consecutive 4 K-byte blocks are set to zero)
- PERFORM FRAME MANAGEMENT FUNCTION and SET STORAGE KEY EXTENDED (reference bit may be set to a specified value)

The record provided by the reference bit is substantially accurate. The reference bit may be set to one by fetching data or instructions that are neither designated nor used by the program, and, under certain conditions, a reference may be made without the reference bit being set to one. Under certain unusual circumstances, a reference bit may be set to zero by other than explicit program action.

Change Recording

Change recording provides information as to which pages have to be saved in auxiliary storage when they are replaced in main storage. Change recording uses the change bit, bit 6 of the storage key.

The change bit is set to one each time a store access causes the contents of the corresponding storage block to be changed, and either (a) enhanced DAT does not apply, or (b) enhanced DAT applies, and either of the following is true:

- The STE-format control in the segment-table entry used by DAT is zero, and the change-

recording override (CO) in the page-table entry used by DAT is zero.

- The STE-format control in the segment-table entry used by DAT is one, and the change-recording override (CO) in the segment-table entry used by DAT is zero.

The change bit is not set to one for an attempt to store if the access is prohibited. In particular:

1. For the CPU, a store access is prohibited whenever an access exception exists for that access, or whenever an exception exists which is of higher priority than the priority of an access exception for that access.
2. For the channel subsystem, a store access is prohibited whenever a key-controlled-protection violation exists for that access.

Change recording is always active and takes place for all store accesses to storage, including those made by any CPU (except when suppressed by the change-recording override, as described below), any operator facility, or the channel subsystem. It takes place for implicit references made by the machine, such as those which are part of interruptions.

Change recording does not take place for the operands of the following instructions since they directly modify a storage key without modifying a storage location:

- RESET REFERENCE BIT EXTENDED
- RESET REFERENCE BITS MULTIPLE
- SET STORAGE KEY EXTENDED (change bit may be set to a specified value)

Change bits which have been changed from zeros to ones are not necessarily restored to zeros on CPU retry (see “CPU Retry” on page 11-2). See “Exceptions to Nullification and Suppression” on page 5-23 for a description of the handling of the change bit in certain unusual situations.

When the condition specified by the M_3 field of a STORE ON CONDITION instruction is not met, it is model dependent whether the change bit is set for the storage location designated by the second operand.

Change-Recording Override

When enhanced DAT applies, and the virtual address is translated by means of DAT-table entries, a change-recording override (CO) is provided in bit 55 of both the segment-table entry and the page-table entry.

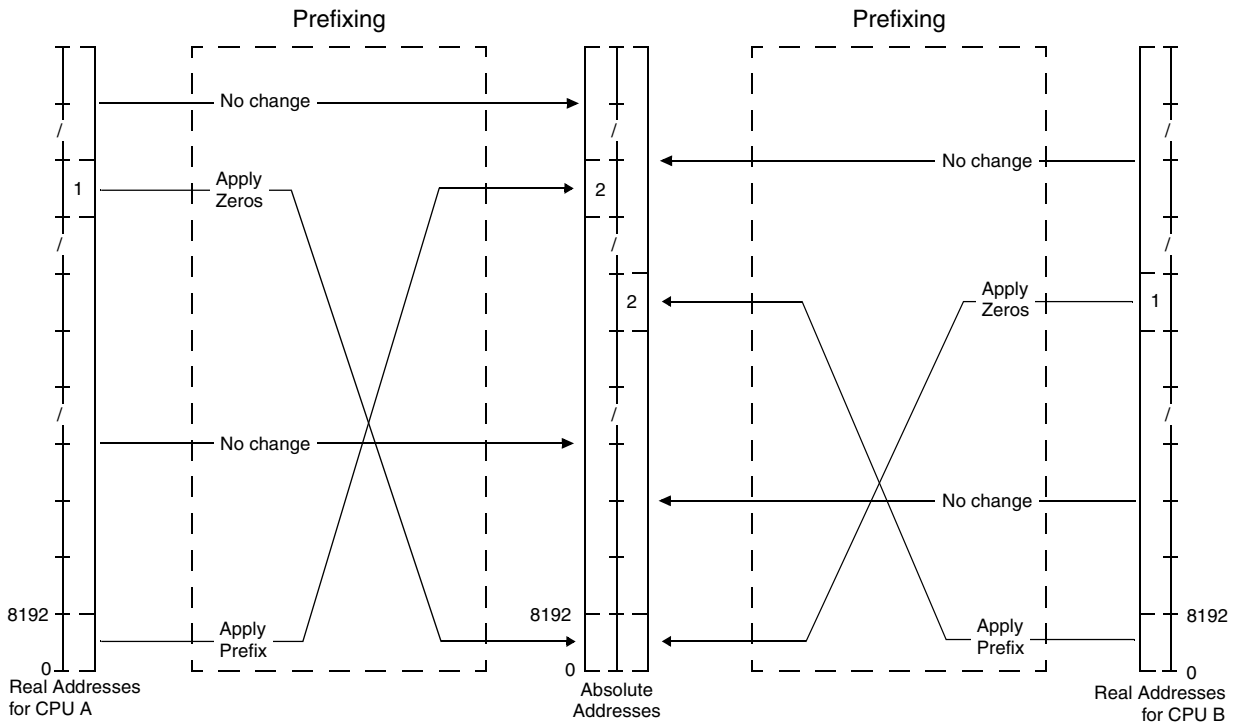
When the STE-format control (FC), bit 53 of the segment-table entry is zero, the change-recording override in the page-table entry applies. When the change-recording override in the PTE is zero, change recording occurs for store operations to the 4K-byte block. When the change-recording override is one, it is unpredictable whether change recording occurs for store operations to the 4K-byte block.

When the STE-format control is one, the change-recording override in the STE applies. When the change-recording override in the STE is zero, change recording occurs for store operations to any of the segment's 256 4K-byte blocks. When the change-recording override in the STE is one, it is unpredictable whether change recording occurs to any of the segment's 256 4K-byte blocks.

The change-recording override does not apply to real or absolute addresses, or to a virtual address that is translated by means of a real-space designation.

Programming Notes:

1. As stated in the section “Change Recording”, above, a fundamental use of the change bit is determining if the contents of a storage block needs to be migrated to auxiliary storage prior to replacing the block. If the program does not need to migrate certain blocks of storage, for example, blocks that are long-term “fixed” (also known as “pinned”), the program may not need to rely on the change bit for those blocks. In this case, either or both of the following techniques may improve CPU performance:
 - Setting the change-recording override to one to indicate that change recording is not required for the blocks
 - Presetting the change bit to one in the storage key
2. See “Modification of Translation Tables” on page 3-55 for restrictions on modifying the change-recording override in the STE or PTE.



- (1) Real addresses in which bits 0-50 are equal to bits 0-50 of the prefix for this CPU (A or B).
- (2) Absolute addresses of the block that contains for this CPU (A or B) the real locations 0-8191.

Figure 3-6. Relationship between Real and Absolute Addresses

DAT uses, at different times, the address-space-control elements in different control registers or specified by the access registers. The choice is determined by the translation mode specified in the current PSW. Four translation modes are available: primary-space mode, secondary-space mode, access-register mode, and home-space mode. Different address spaces are addressable depending on the translation mode.

At any instant when the CPU is in the primary-space mode or secondary-space mode, the CPU can translate virtual addresses belonging to two address spaces — the primary address space and the secondary address space. At any instant when the CPU is in the access-register mode, it can translate virtual addresses of up to 16 address spaces — the primary address space and up to 15 AR-specified address spaces. At any instant when the CPU is in the home-space mode, it can translate virtual addresses of the home address space.

The primary address space is identified as such because it consists of primary virtual addresses,

which are translated by means of the primary address-space-control element (ASCE). Similarly, the secondary address space consists of secondary virtual addresses translated by means of the secondary ASCE, the AR-specified address spaces consist of AR-specified virtual addresses translated by means of AR-specified ASCEs, and the home address space consists of home virtual addresses translated by means of the home ASCE. The primary and secondary ASCEs are in control registers 1 and 7, respectively. The AR-specified ASCEs are in control registers 1 and 7 and in table entries called ASN-second-table entries. The home ASCE is in control register 13.

Changing to Different Address Spaces

A program can cause different address spaces to be addressable by using the semiprivileged SET ADDRESS SPACE CONTROL or SET ADDRESS SPACE CONTROL FAST instruction to change the translation mode to the primary-space mode, secondary-space mode, access-register mode, or home-space mode. However, SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL

FAST can set the home-space mode only in the supervisor state. The program can cause still other address spaces to be addressable by using unprivileged instructions to change the contents of the access registers and by using semiprivileged or privileged instructions to change the address-space-control elements in control registers 1 and 7. The semiprivileged instructions are ones that cause linkage from one address space to another and are the `BRANCH IN SUBSPACE GROUP`, `PROGRAM CALL`, `PROGRAM RETURN`, `PROGRAM TRANSFER`, `PROGRAM TRANSFER WITH INSTANCE`, `SET SECONDARY ASN`, and `SET SECONDARY ASN WITH INSTANCE` instructions. The privileged instructions are the `LOAD ADDRESS SPACE PARAMETERS` and `LOAD CONTROL` instructions. Only `LOAD CONTROL` is available for changing the home address-space-control element in control register 13.

Address-Space Number

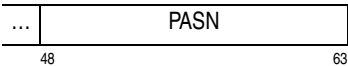
An address space may be assigned an address-space number (ASN) by the control program. The ASN designates, within a two-level table structure in main storage, an ASN-second-table entry containing information about the address space. If the ASN-second-table entry is marked as valid, it contains the address-space-control element that defines the address space.

Under certain circumstances, the semiprivileged instructions which place a new address-space-control element in control register 1 or 7 fetch this element from an ASN-second-table entry. Some of these instructions use an ASN-translation mechanism which, given an ASN, can locate the designated ASN-second-table entry.

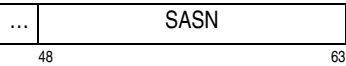
The 16-bit unsigned binary format of the ASN permits 64K unique ASNs.

The ASNs for the primary and secondary address spaces are assigned positions in control registers. The ASN for the primary address space, called the primary ASN, is assigned bits 48-63 in control register 4, and that for the secondary address space, called the secondary ASN, is assigned bits 48-63 in control register 3. Bits 48-63 of these registers have the following formats:

Control Register 4



Control Register 3



A semiprivileged instruction that loads the primary or secondary address-space-control element into the appropriate control register also loads the corresponding ASN into the appropriate control register.

The ASN for the home address space is not assigned a position in a control register.

An access register containing the value 0 or 1 specifies the primary or secondary address space, respectively; and the address-space-control element specified by the access register is in control register 1 or 7, respectively. An access register containing any other value designates an entry in a table called an access list. The designated access-list entry contains the real address of an ASN-second-table entry for the address space specified by the access register. The address-space-control element specified by the access register is in the ASN-second-table entry. Translating the contents of an access register to obtain an address-space-control element for use by DAT does not involve the use of an ASN.

Note: Virtual storage consisting of byte locations ordered according to their virtual addresses in an address space is usually referred to as “storage.”

Programming Note: Because an ASN-second-table entry is located from an access-list entry by means of its address instead of by means of its ASN, the ASN-second-table entries designated by access-list entries can be “pseudo” ASN-second-table entries, that is, entries which are not in the two-level structure able to be indexed by means of the ASN-translation process. The number of unique pseudo ASN-second-table entries can be greater than the number of unique ASNs and is limited only by the amount of storage available to be occupied by the ASN-second-table entries. Thus, in a sense, there is no limit on the number of possible address spaces.

ASN-Second-Table-Entry Sequence Number

The ASN-second-table entry contains an ASN-second-table-entry sequence number (ASTESN) that may be used to control storage references to the related address space by means of an access regis-

ter or through use of the SET SECONDARY ASN, SET SECONDARY ASN WITH INSTANCE, or LOAD ADDRESS SPACE PARAMETERS instruction and that may be used to control linkages to or back to the address space by means of the BRANCH IN SUBSPACE GROUP, PROGRAM CALL, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, and PROGRAM RETURN instructions. These uses of the ASTESN are described below.

When any instruction uses an access register to designate an access-list entry, which in turn designates an ASN-second-table entry, to perform a storage reference to the address space specified by the ASN-second-table entry, an ASTESN in the access-list entry is compared to the ASTESN in the ASN-second-table entry, and these ASTESNs must be equal; otherwise, an ASTE-sequence exception is recognized. This use of the ASTESN allows an access-list entry to be made unusable if some authorization policy is changed or the designated ASN-second-table entry is reassigned to specify a conceptually different address space. The entry can be made unusable by changing the ASTESN in the ASN-second-table entry. The use is further described in “Revoking Accessing Capability:” on page 5-48.

The ASTESN is used in connection with subspace groups as follows:

- When BRANCH IN SUBSPACE GROUP uses an access register to designate an access-list entry, which in turn designates an ASN-second-table entry, to transfer control to the subspace specified by the ASN-second-table entry, an ASTESN in the access-list entry is compared to the ASTESN in the ASN-second-table entry, and these ASTESNs must be equal.
- When BRANCH IN SUBSPACE GROUP uses, in an access register, an access-list-entry token with the value 00000001 hex to transfer control to the subspace specified by the subspace-ASN-second-table-entry origin in the current dispatchable-unit control table (designated by control register 2), a subspace-ASN-second-table-entry sequence number (SSASTESN) in the dispatchable-unit control table is compared to the ASTESN in the ASN-second-table entry for the subspace, and the SSASTESN and ASTESN must be equal.
- When LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, PROGRAM RETURN, PRO-

GRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, or SET SECONDARY ASN WITH INSTANCE sets the primary address-space-control element in control register 1 or the secondary address-space-control element in control register 7 with bits of the address-space-control element in the ASN-second-table entry for a subspace, the SSASTESN is compared to the ASTESN in the ASN-second-table entry for the subspace, and the SSASTESN and ASTESN must be equal.

Otherwise, in all of the above three cases, an ASTE-sequence exception is recognized, except that LOAD ADDRESS SPACE PARAMETERS sets condition code 1 or 2 depending on whether the subspace is the primary address space or the secondary address space, respectively. These uses of the ASTESN allow an access-list entry and the subspace-ASN-second-table-entry origin in the current dispatchable-unit control table to be made unusable when the ASN-second-table entry either of them designates is reassigned to specify a conceptually different address space. The access-list entry and subspace-ASN-second-table-entry origin can be made unusable by changing the ASTESN in the ASN-second-table entry. The uses are further described in “Subspace Groups” on page 5-62.

Programming Note: The above operations use an ASN-second-table-entry origin in either an access-list entry or the dispatchable-unit control table; they do not use an ASN. The designated ASN-second-table entry is normally a pseudo ASN-second-table entry (one not in the structure able to be indexed by means of an ASN).

ASN-Second-Table-Entry Instance Number and ASN Reuse

The ASN-and-LX-reuse facility may be installed on the model. If this facility is installed, the ASN-second-table entry contains an ASN-second-table-entry instance number (ASTEIN), and certain new definitions related to the ASTEIN and to a new linkage-second-table-entry sequence number (LSTESN) apply. The definitions related to the ASTEIN are summarized below and also given in the appropriate sections of this publication.

- The ASN-and-LX-reuse facility includes the following new instructions:

- EXTRACT PRIMARY ASN AND INSTANCE
- EXTRACT SECONDARY ASN AND INSTANCE
- PROGRAM TRANSFER WITH INSTANCE
- SET SECONDARY ASN WITH INSTANCE

- The facility also includes the following new control bits:
 - The ASN-and-LX-reuse control (R), bit 44 of control register 0
 - The controlled-ASN bit (CA), bit 30 of word 1 of the ASN-second-table entry
 - The reusable-ASN bit (RA), bit 31 of word 1 of the ASN-second-table entry

The three control bits are shown as follows:

Control Register 0

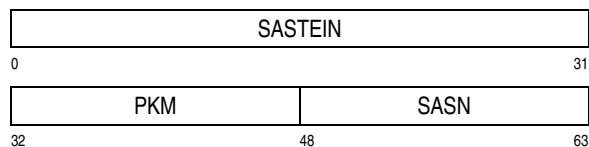


Word 1 of ASN-Second-Table Entry

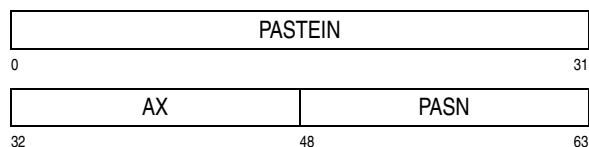


- The facility also includes the primary ASTEIN in bit positions 0-31 of control register 4 and the secondary ASTEIN in bit positions 0-31 of control register 3. The primary ASTEIN is a copy of the ASTEIN in the ASN-second-table entry for the current primary address space specified by the PASN in control register 4, and the secondary ASTEIN is a copy of the ASTEIN in the ASN-second-table entry for the current secondary address space specified by the SASN in control register 3. The complete formats of control registers 3 and 4 are as follows:

Control Register 3



Control Register 4



- The following operations are performed if the ASN-and-LX-reuse control, bit 44 of control register 0, is one. When LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, or SET SECONDARY ASN (1) sets the primary ASN in control register 4 or the secondary ASN in control register 3 equal to a specified ASN and (2) sets the primary address-space-control element in control register 1 or the secondary address-space-control element in control register 7, respectively, with the value of an address-space-control element obtained from an ASN-second-table entry located by means of ASN translation of the specified ASN (or located, by PROGRAM CALL only, by means of an ASTE address), the instruction also sets the primary ASTEIN in control register 4 or secondary ASTEIN in control register 3, respectively, with the value of the ASTEIN in the ASN-second-table entry. However, PROGRAM TRANSFER and SET SECONDARY ASN, which are performing their space-switching operations, recognize a special-operation exception if the reusable-ASN bit, bit 31 of word 1, in the ASN-second-table entry is one.

When the ASN-and-LX-reuse control is one and any of the instructions named above or BRANCH IN SUBSPACE GROUP sets the secondary ASN and secondary address-space-control element equal to the primary ASN (in the space-switching stacking PROGRAM CALL operation, this may be the old or the new primary ASN, as determined by a bit in the entry-table entry used) and primary address-space-control element, respectively, it also sets the secondary ASTEIN in control register 3 equal to the primary ASTEIN in control register 4 (again, in the space-switching stacking PROGRAM CALL case, this may be the old or the new primary ASTEIN). Since this function does not include the accessing of an ASN-second-table entry, it is not affected by a reusable-ASN bit.

PROGRAM TRANSFER WITH INSTANCE and SET SECONDARY ASN WITH INSTANCE operate as described above except (1) independent of the value of the ASN-and-LX-reuse control (they always update the primary ASTEIN and secondary ASTEIN as described above), (2) without recognizing an exception due to the reusable-ASN bit (they ignore the bit), and (3) in their space-switching forms and the problem

state at the beginning of the operation, by recognizing a special-operation exception if the controlled-ASN bit, bit 30 of word 1, is one in the ASN-second-table entry located by ASN translation. PROGRAM TRANSFER WITH INSTANCE and SET SECONDARY ASN WITH INSTANCE ignore the controlled-ASN bit if the CPU is in the supervisor state at the beginning of the operation. (PROGRAM TRANSFER WITH INSTANCE may switch the CPU from the supervisor state to the problem state.)

- PROGRAM TRANSFER WITH INSTANCE and SET SECONDARY ASN WITH INSTANCE operate the same as PROGRAM TRANSFER and SET SECONDARY ASN, respectively, except as described in the preceding item and except as follows.

Independent of the ASN-and-LX-reuse control and the reusable-ASN bit, but not of the controlled-ASN bit, each of PROGRAM TRANSFER WITH INSTANCE with space switching and SET SECONDARY ASN WITH INSTANCE with space switching, after it has used the ASN in the R_1 general register to locate an ASN-second-table entry, compares an ASTEIN in bit positions 0-31 of general register R_1 to the ASTEIN in the entry. The two ASTEINs must be equal; otherwise, an ASTE-instance exception is recognized. This comparison is performed in both the problem and the supervisor state.

- The following operations are performed if the ASN-and-LX-reuse control is one. Each of stacking PROGRAM CALL and BRANCH AND STACK places the current primary ASTEIN in bytes 180-183 of the linkage-stack state entry that it forms, and it places the current secondary ASTEIN, bits 0-31 of control register 3, in bytes 176-179 of the state entry.

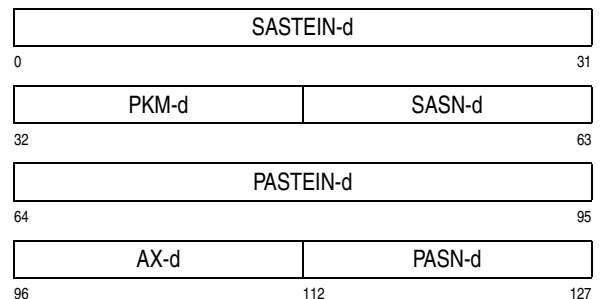
Note: There is not a test of an ASTEIN related to the ASN in the entry-table entry used by PROGRAM CALL. There may be a test of a linkage-second-table-entry sequence number related to the linkage index used to locate the entry-table entry, as summarized in “ASN-and-LX-Reuse Control:” on page 5-27.

- The following operations are performed if the ASN-and-LX-reuse control is one. PROGRAM RETURN with space switching, after it has used the PASN in bytes 134 and 135 of the linkage-stack program-call state entry to locate an ASN-

second-table entry (because that PASN is not equal to the current PASN in control register 4), compares the primary ASTEIN in bytes 180-183 of the state entry to the ASTEIN in the ASN-second-table entry. PROGRAM RETURN to current primary or with space switching, if it uses the SASN in bytes 130 and 131 of the state entry to locate an ASN-second-table entry (because that SASN is not equal to the new PASN), compares the secondary ASTEIN in bytes 176-179 of the state entry to the ASTEIN in the ASN-second-table entry. The one or two comparisons of ASTEINs must each give equal results; otherwise, an ASTE-instance exception is recognized. These operations occur independent of the controlled-ASN and reusable-ASN bits in the ASN-second-table entry. PROGRAM RETURN does not compare ASTEINs when it unstacks a branch state entry.

- The following format applies, and the operations are performed, if the ASN-and-LX-reuse control is one. The first operand of LOAD ADDRESS SPACE PARAMETERS is two consecutive doublewords having the following format:

First Operand of LOAD ADDRESS SPACE PARAMETERS



If PASN translation is performed to locate an ASN-second-table entry, PASTEIN-d is compared to the ASTEIN in the ASN-second-table entry, and they must be equal; otherwise, condition code 1 is set. If SASN translation is performed to locate an ASN-second-table entry, SASTEIN-d is compared to the ASTEIN in the ASN-second-table entry, and they must be equal; otherwise, condition code 2 is set. These operations are performed independent of the controlled-ASN and reusable-ASN bits in the ASN-second-table entries.

- Independent of the ASN-and-LX-reuse control, EXTRACT PRIMARY ASN AND INSTANCE places the current primary ASN, bits 48-63 of

control register 4, in bit positions 48-63 of general register R_1 ; places the current primary ASTEIN, bits 0-31 of control register 4, in bit positions 0-31 of the general register; and places zeros in bit positions 32-47 of the general register. EXTRACT SECONDARY ASN AND INSTANCE operates the same except that it obtains the secondary ASN and secondary ASTEIN in control register 3 for placement in the general register.

- Independent of the ASN-and-LX-reuse control, a new code, code 5, of the EXTRACT STACKED STATE instruction is valid. Code 5 causes the saved secondary ASTEIN, bytes 176-179 of the state entry, to be placed in bit positions 0-31 of the R_1 general register and the saved primary ASTEIN, bytes 180-183 of the state entry, to be placed in bit positions 0-31 of general register $R_1 + 1$. Bits 32-63 of the general registers remain unchanged.

These uses of the ASTEIN allow, in all cases except one, an ASN associated with a particular ASTEIN to be made unusable when the ASN and the ASN-second-table entry it designates are reassigned to specify a conceptually different address space. The ASN-and-ASTEIN combination can be made unusable by (1) setting to one the reusable-ASN bit in the ASN-second-table entry, which prevents the use of PROGRAM TRANSFER and SET SECONDARY ASN, and (2) changing the ASTEIN in the ASN-second-table entry, which prevents the use of LOAD ADDRESS SPACE PARAMETERS, PROGRAM TRANSFER WITH INSTANCE, PROGRAM RETURN, and SET SECONDARY ASN WITH INSTANCE. The uncovered case is that of an ASN (and the corresponding ASN-second-table-entry address) in an entry-table entry, and this case should be handled by means of deletion, by the control program, of the entry-table entry. However, if the control program then reforms the entry-table entry (forms an entry-table entry located by means of the same PC number, used by PROGRAM CALL, as was the deleted entry-table entry), a PC number that was used to link to the conceptually deleted address space will then be usable to link to a different (different ASN) or conceptually different (same ASN but different contents) address space, and this error should be avoided by using the linkage-second-table-entry sequence number as described in "ASN-and-LX-Reuse Control:" on page 5-27.

Programming Notes:

1. The reusable-ASN bit of the ASN-and-LX-reuse facility provides reliability, availability, and serviceability. Most importantly, it provides availability since it allows ASNs to be reused. However it does not provide system integrity since the ASTEIN in the general register used by PROGRAM TRANSFER WITH INSTANCE or SET SECONDARY ASN WITH INSTANCE is provided by the program. (The authorization index used by those instructions normally provides system integrity but may fail to do so if an ASN authorized by an unchanged authorization index is reused.) The controlled-ASN bit provides system integrity since, if the CPU is in the problem state at the beginning of the operation, PROGRAM TRANSFER WITH INSTANCE and SET SECONDARY ASN WITH INSTANCE are unable to proceed successfully after accessing an ASN-second-table entry in which the controlled-ASN bit is one. When the calling program and called program are both executed in the problem state, a program in an address space for which the controlled-ASN bit is one should perform linkages to programs in other address spaces only by means of stacking PROGRAM CALL. If that is done, the return by means of PROGRAM RETURN will always be successful, or will fail appropriately when the ASN of the first address space has been changed, because of the comparison to the saved primary ASTEIN in the linkage-stack state entry.
2. A program given control by a basic PROGRAM CALL operation can use EXTRACT SECONDARY ASN AND INSTANCE to obtain the ASTEIN to be used by PROGRAM TRANSFER WITH INSTANCE to return to the calling program or by SET SECONDARY ASN WITH INSTANCE to restore its secondary address space after a change of that space. This EXTRACT SECONDARY ASN AND INSTANCE instruction should be executed while the original secondary space remains continuously the secondary space; otherwise, depending on actions by the control program, EXTRACT SECONDARY ASN AND INSTANCE may return an ASTEIN that allows return to or use of a conceptually incorrect secondary space for which the ASTEIN has been changed.
3. A summary of the functions related to ASN reuse is given in Figure 3-7 on page 3-25.

4. There are certain programming errors, or situations that are not necessarily errors, that will not be detected. These cases are described in the definitions of the named instructions and are as follows:

- In LOAD ADDRESS SPACE PARAMETERS:
 - SASN translation is performed only when, but not necessarily when, SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d, SASCE-new and SAS-TEIN-new are set equal to PASCE-new and PASTEIN-new, respectively. In this case, there is not a test of whether SAS-TEIN-d is equal to PASTEIN-d; SAS-TEIN-d is ignored.
 - When SASN-d is not equal to PASN-d and is equal to SASN-old, bit 61 (force ASN translation) is zero, and bit 63 (skip SASN authorization) is one, SASN translation is not performed, and SASCE-old and SASTEIN-old become SASCE-new and SASTEIN-new, respectively. In this case, there is not a test of whether SAS-TEIN-d is equal to SASTEIN-old; SAS-TEIN-d is ignored.
- In PROGRAM RETURN, if the new SASN is equal to the new PASN, the SASCE in control register 7 is set equal to the new PASCE in control register 1. The SASTEIN, PKM, and SASN in control register 3 remain as restored from the state entry. In this case, there is not a test of whether the new SAS-TEIN is equal to the new PASTEIN. (There has already been a test of whether the PASTEIN saved in the entry, which becomes the new PASTEIN, equals the ASTEIN in the new PASTE.)
- In the PROGRAM TRANSFER WITH INSTANCE to-current-primary operation, there is not a test of whether the current PASTEIN equals the ASTEIN specified in bit positions 0-31 of general register R₁; the ASTEIN is ignored.
- In the SET SECONDARY ASN WITH INSTANCE to-current-primary operation, there is not a test of whether the current PASTEIN (to which the SASTEIN is set equal) equals the ASTEIN specified in bit positions 0-31 of general register R₁; the ASTEIN is ignored.

Function	Required Bit Value		
	R	CA	RA
LASP, PC, and PR update PASTEIN or SASTEIN in a control register after accessing an ASN-second-table entry	1	–	–
PT and SSAR update PASTEIN or SASTEIN in a control register after accessing an ASN-second-table entry	1	–	0 ¹
BSG, LASP, PC, PR, PT, and SSAR set SASTEIN equal to PASTEIN	1	–	–
PTI and SSAIR update PASTEIN or SASTEIN in a control register after accessing an ASN-second-table entry	–	0 ²	–
PTI and SSAIR set SASTEIN equal to PASTEIN	–	–	–
PTI and SSAIR after accessing an ASN-second-table entry, compare ASTEIN in general register R ₁ to ASTEIN in the entry	–	0 ²	–
Stacking PC and BAKR copy PASTEIN and SASTEIN from control registers to state entry	1	–	–
PR, after accessing an ASN-second-table entry, compares PASTEIN or SASTEIN in the state entry to ASTEIN in the ASN-second-table entry ³	1	–	–
LASP, after accessing an ASN-second-table entry, compares PASTEIN-d or SASTEIN-d to ASTEIN in the entry ⁴	1	–	–

Figure 3-7. Summary of Functions Related to ASN Reuse
(Part 1 of 2)

Function	Required Bit Value		
	R	CA	RA
EPAIR copies PASTEIN and PASN from control register 4 to general register R ₁	–	–	–
ESAIR copies SASTEIN and SASN from control register 3 to general register R ₁	–	–	–
ESTA code 5 copies PASTEIN and SASTEIN from state entry to general registers	–	–	–
Explanation: — Bit is ignored or not applicable to the operation. 1 A special-operation exception is recognized if the bit is one. 2 A special-operation exception is recognized if the bit is one and the CPU is in the problem state at the beginning of the operation. 3 An ASTE-instance exception is recognized if the ASTEINs are not equal. 4 Condition code 1 (if PASTEIN comparison) or 2 (if SASTEIN comparison) is set if the ASTEIN is not equal. CA Controlled-ASN bit, bit 30 of word 1 of ASN-second-table entry. R ASN-and-LX-reuse control, bit 44 of control register 0. RA Reusable-ASN bit, bit 31 of word 1 of ASN-second-table entry.			

Figure 3-7. Summary of Functions Related to ASN Reuse (Part 2 of 2)

ASN Translation

ASN translation is the process of translating a 16-bit ASN to locate the ASN-second-table entry designated by the ASN. ASN translation is performed as part of PROGRAM TRANSFER with space switching (PT-ss) PROGRAM TRANSFER WITH INSTANCE with space switching (PTI-ss), SET SECONDARY ASN with space switching (SSAR-ss), and SET SECONDARY ASN WITH INSTANCE with space switching (SSAIR-ss), and it may be performed as part of LOAD ADDRESS SPACE PARAMETERS. For PT-ss and PTI-ss the ASN which is translated replaces the primary ASN in control register 4. For SSAR-ss and SSAIR-ss, the ASN which is translated replaces the

secondary ASN in control register 3. These two translation processes are called primary ASN translation and secondary ASN translation, respectively, and both can occur for LOAD ADDRESS SPACE PARAMETERS. The ASN-translation process is the same for both primary and secondary ASN translation; only the uses of the results of the process are different.

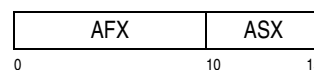
ASN translation may also be performed as part of PROGRAM RETURN. Primary ASN translation is performed as part of PROGRAM RETURN with space switching (PR-ss). Secondary ASN translation is performed if the secondary ASN restored by PROGRAM RETURN (PR-ss or PROGRAM RETURN to current primary) does not equal the primary ASN restored by PROGRAM RETURN.

PROGRAM CALL with space switching (PC-ss) performs the equivalent of primary ASN translation by obtaining a primary ASN and the address of the corresponding ASN-second-table entry from an entry-table entry.

The ASN-translation process uses two tables, the ASN first table and the ASN second table. They are used to locate the ASN-second-table entry and a third table, the authority table, which is used when ASN authorization is performed.

For the purposes of this translation, the 16-bit ASN is considered to consist of two parts: the ASN-first-table index (AFX) is the leftmost 10 bits of the ASN, and the ASN-second-table index (ASX) is the six rightmost bits. The ASN has the following format:

ASN



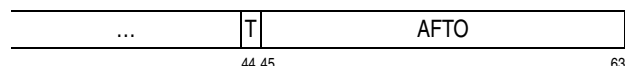
The AFX is used to select an entry from the ASN first table. The origin of the ASN first table is designated by the ASN-first-table origin in control register 14. The ASN-first-table entry contains the origin of the ASN second table. The ASX is used to select an entry from the ASN second table.

As a result of primary ASN translation and during the operation of PROGRAM CALL with space switching, the address of the located ASN-second-table entry (ASTE) is placed in control register 5 as the new primary-ASTE origin (PASTEO).

ASN-Translation Controls

ASN translation is controlled by the ASN-translation-control bit and the ASN-first-table origin, both of which reside in control register 14.

Control Register 14



ASN-Translation Control (T): Bit 44 of control register 14 is the ASN-translation-control bit. This bit provides a mechanism whereby the control program can indicate whether ASN translation can occur while a particular program is being executed, and also whether the execution of PROGRAM CALL with space switching is allowed. Bit 44 must be one to allow completion of these instructions:

- LOAD ADDRESS SPACE PARAMETERS
- PROGRAM CALL with space switching
- PROGRAM RETURN with space switching or when the restored SASN does not equal the restored PASN
- PROGRAM TRANSFER with space switching
- PROGRAM TRANSFER WITH INSTANCE with space switching
- SET SECONDARY ASN
- SET SECONDARY ASN WITH INSTANCE

Otherwise, a special-operation exception is recognized. The ASN-translation-control bit is examined in both the problem and the supervisor states.

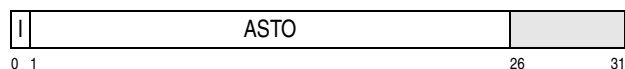
ASN-First-Table Origin (AFTO): Bits 45-63 of control register 14, with 12 zeros appended on the right, form a 31-bit real address that designates the beginning of the ASN first table.

ASN-Translation Tables

The ASN-translation process consists in a two-level lookup using two tables: an ASN first table and an ASN second table. These tables reside in real storage.

ASN-First-Table Entries

An entry in the ASN first table has the following format:



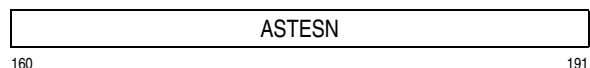
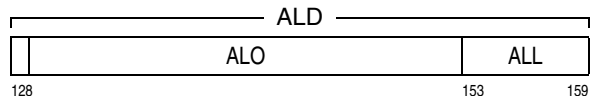
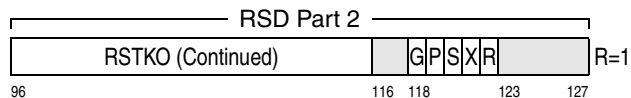
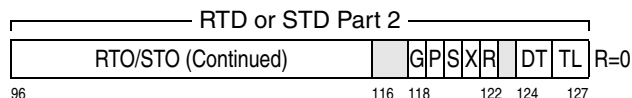
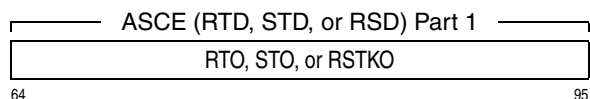
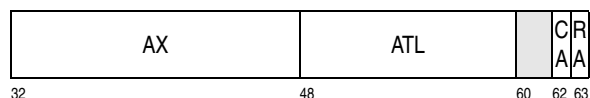
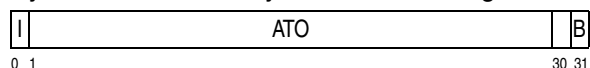
The fields in the entry are allocated as follows:

AFX-Invalid Bit (I): Bit 0 controls whether the ASN second table associated with the ASN-first-table entry is available. When bit 0 is zero, ASN translation proceeds by using the designated ASN second table. When the bit is one, the ASN translation cannot continue.

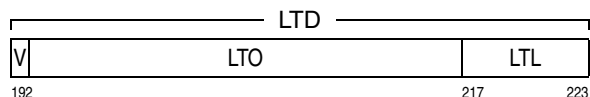
ASN-Second-Table Origin (ASTO): Bits 1-25, with six zeros appended on the right, are used to form a 31-bit real address that designates the beginning of the ASN second table.

ASN-Second-Table Entries

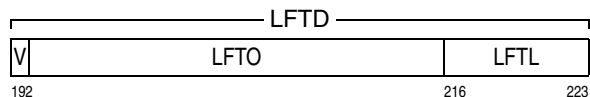
The ASN-second-table entry has a length of 64 bytes, with only the first 48 bytes currently in use. Bytes 0-47 of the entry have the following format:

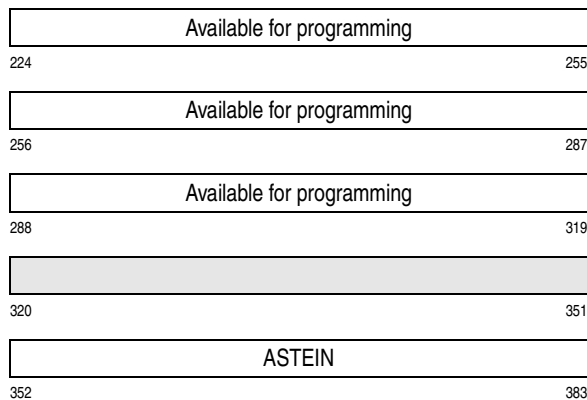


If ASN-and-LX Reuse Is Not Enabled



If ASN-and-LX Reuse Is Enabled





The fields in bytes 0-47 of the ASN-second-table entry are allocated as follows. Only the fields that are used in or as a result of ASN translation or PROGRAM CALL with space switching are described in detail.

ASX-Invalid Bit (I): Bit 0 controls whether the address space associated with the ASN-second-table entry is available. When bit 0 is zero, ASN translation proceeds. When the bit is one, the ASN translation cannot continue.

Authority-Table Origin (ATO): Bits 1-29, with two zeros appended on the right, are used to form a 31-bit real address that designates the beginning of the authority table.

Base-Space Bit (B): Bit 31 specifies, when one, that the address space associated with the ASN-second-table entry is the base space of a subspace group. Bit 31 is further described in "Subspace-Group ASN-Second-Table Entries" on page 5-64.

Authorization Index (AX): Bits 32-47 are used in ASN authorization as an index to locate the authority bits in the authority table. The AX field is used as a result of primary ASN translation by PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, and, possibly, LOAD ADDRESS SPACE PARAMETERS. It is also used by PROGRAM CALL with space switching. The AX field is ignored after secondary ASN translation.

Authority-Table Length (ATL): Bits 48-59 specify the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four bytes, is one more than the ATL value. The contents of the ATL field are used to establish

whether the entry designated by a particular AX falls within the authority table.

Controlled-ASN Bit (CA): PROGRAM TRANSFER WITH INSTANCE with space switching and SET SECONDARY ASN WITH INSTANCE with space switching recognize a special-operation exception if bit 62 is one and the CPU is in the problem state at the beginning of the operation. Bit 62 is ignored in the supervisor state.

Reusable-ASN Bit (RA): If the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, PROGRAM TRANSFER with space switching and SET SECONDARY ASN with space switching recognize a special-operation exception if bit 63 is one in the problem or the supervisor state.

Address-Space-Control Element (ASCE): Bits 64-127 are an eight-byte address-space-control element (ASCE) that may be a region-table designation (RTD), a segment-table designation (STD), or a real-space designation (RSD). (The term "region-table designation" is used to mean a region-first-table designation, region-second-table designation, or region-third-table designation.) The ASCE field is used as a result of ASN translation or in PROGRAM CALL with space switching to replace the primary ASCE (PASCE) or the secondary ASCE (SASCE). For PROGRAM CALL with space switching, the ASCE field replaces the PASCE, bits 0-63 of control register 1. For SET SECONDARY ASN, and SET SECONDARY ASN WITH INSTANCE, the ASCE field replaces the SASCE, bits 0-63 of control register 7. Each of these actions may occur independently for LOAD ADDRESS SPACE PARAMETERS. For PROGRAM TRANSFER, and PROGRAM TRANSFER WITH INSTANCE, the ASCE field replaces both the PASCE and the SASCE. For PROGRAM RETURN, as a result of primary ASN translation, the ASCE field replaces the PASCE, and, as a result of secondary ASN translation, the ASCE field replaces the SASCE. The contents of the entire ASCE field are placed in the appropriate control registers without being inspected for validity.

The subspace-group-control bit (G), bit 118 of the ASCE field, indicates, when one, that the ASCE specifies an address space that is the base space or a subspace of a subspace group. The bit is further described in "Subspace-Group ASN-Second-Table Entries" on page 5-64.

Bit 121 (X) of the ASCE field is the space-switch-event-control bit. When, in the space-switching operations of PROGRAM CALL, PROGRAM RETURN, and PROGRAM TRANSFER, and PROGRAM TRANSFER WITH INSTANCE, this bit is one in control register 1 either before or after the execution of the instruction, a program interruption for a space-switch event occurs after the execution of the instruction is completed. A space-switch-event program interruption also occurs after the completion of a SET ADDRESS SPACE CONTROL, SET ADDRESS SPACE CONTROL FAST, or RESUME PROGRAM instruction that changes the translation mode either to or from the home-space mode when this bit is one in either control register 1 or control register 13. When, in LOAD ADDRESS SPACE PARAMETERS, this bit is one during primary ASN translation, this fact is indicated by the condition code.

The real-space-control bit (R), bit 122 of the ASCE field, indicates, when zero, that the ASCE is a region-table or segment-table designation or, when one, that the ASCE is a real-space designation.

When bit 122 is zero, the designation-type-control bits (DT), bits 124 and 125 of the ASCE field, indicate the designation type of the ASCE. A value 11, 10, 01, or 00 binary of bits 124 and 125 indicates a region-first-table designation, region-second-table designation, region-third-table designation, or segment-table designation, respectively.

The other fields in the ASCE (RTO, STO, P, S, TL, and RSTKO) are described in “Control Register 1” on page 3-38.

Access-List Designation (ALD): The access-list-designation (ALD) field is described in “ASN-Second-Table Entries” on page 5-54.

ASN-Second-Table-Entry Sequence Number (ASTESN): The ASTE-sequence-number (ASTESN) field is described in “ASN-Second-Table Entries” on page 5-54.

Linkage-Table Designation (LTD) or Linkage-First-Table Designation (LFTD): The linkage-table-designation (LTD) or linkage-first-table designation (LFTD) field in the ASN-second-table entry is described in “PC-Number Translation Control” on page 5-31.

ASN-Second-Table-Entry Instance Number (ASTEIN): When the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control in control register 0, bits 352-383 are compared to an ASTEIN specified along with an ASN for use by PROGRAM RETURN or LOAD ADDRESS SPACE PARAMETERS. The ASTEINs must be equal; otherwise, an ASTE-instance exception is recognized by PROGRAM RETURN or condition code 1 or 2 is set by LOAD ADDRESS SPACE PARAMETERS. This comparison and the exception result also occur in the operations of PROGRAM TRANSFER WITH INSTANCE and SET SECONDARY ASN WITH INSTANCE except independent of the ASN-and-LX-reuse control.

Bits 224-319 in the ASN-second-table entry are available for use by programming.

Programming Note: All unused fields in the ASN-second-table entry, including the unused fields in bytes 0-31 and all of bytes 40-43 and 48-63 should be set to zeros. These fields are reserved for future extensions, and programs which place nonzero values in these fields may not operate compatibly on future machines.

ASN-Translation Process

This section describes the ASN-translation process as it is performed during the execution of the space-switching forms of PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, and SET SECONDARY ASN WITH INSTANCE, and also in PROGRAM RETURN when the restored secondary ASN does not equal the restored primary ASN. ASN translation for LOAD ADDRESS SPACE PARAMETERS is the same except that AFX-translation and ASX-translation exceptions do not occur; such conditions are instead indicated by the condition code. Translation of an ASN is performed by means of two tables, an ASN first table and an ASN second table, both of which reside in main storage.

The ASN first index is used to select an entry from the ASN first table. This entry designates the ASN second table to be used.

The ASN second index is used to select an entry from the ASN second table.

If the I bit is one in either the ASN-first-table entry or the ASN-second-table entry, the entry is invalid, and the ASN-translation process cannot be completed. An AFX-translation exception or ASX-translation exception is recognized.

Whenever access to main storage is made during the ASN-translation process for the purpose of fetching an entry from an ASN first table or ASN second table, key-controlled protection does not apply.

The ASN-translation process is shown in Figure 3-8 on page 3-30.

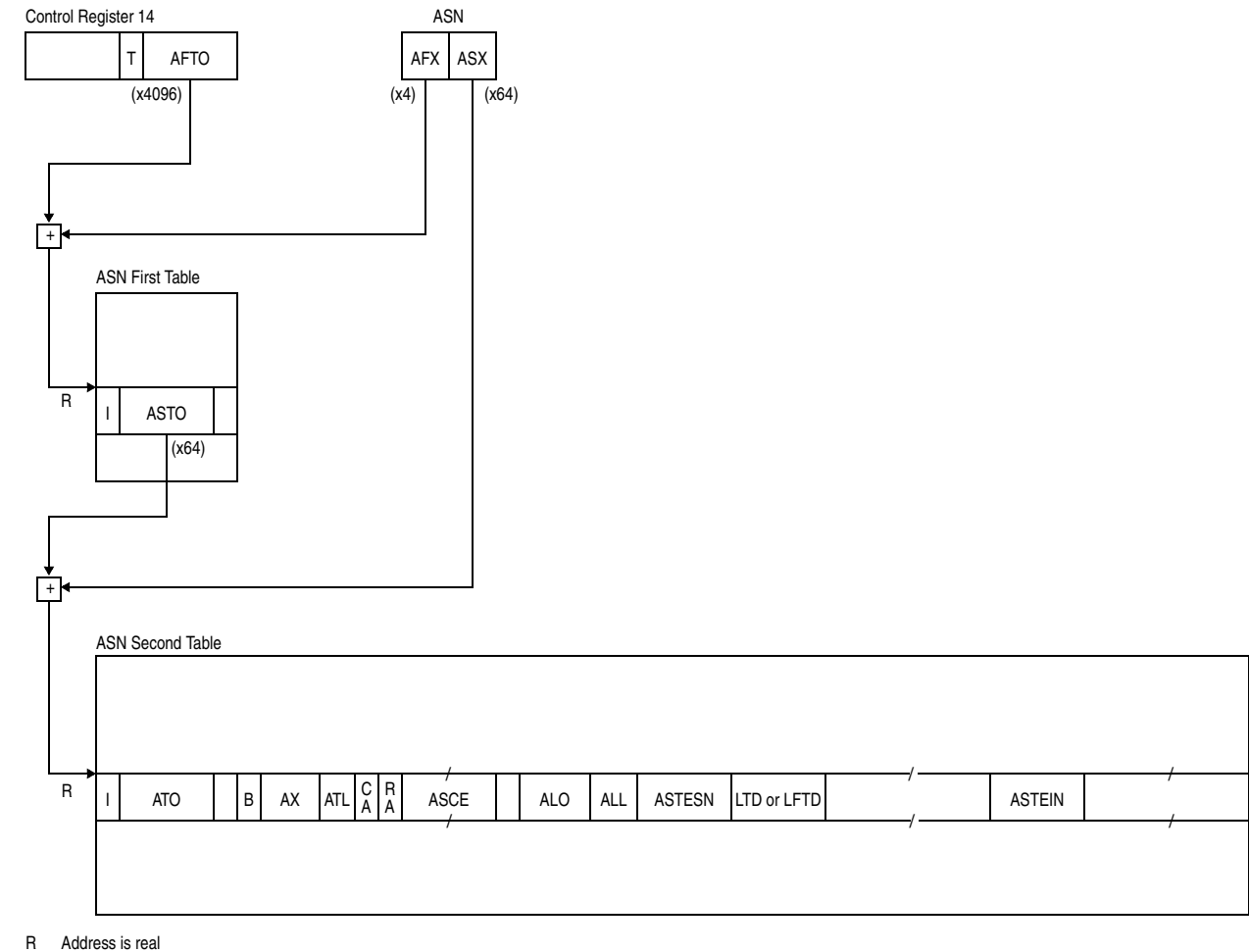


Figure 3-8. ASN Translation

ASN-First-Table Lookup

The AFX portion of the ASN, in conjunction with the ASN-first-table origin, is used to select an entry from the ASN first table.

The 31-bit real address of the ASN-first-table entry is obtained by appending 12 zeros on the right to the AFT origin contained in bit positions 45-63 of control register 14 and adding the AFX portion with two rightmost and 19 leftmost zeros appended. This addition cannot cause a carry into bit position 0. The

31-bit address is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

All four bytes of the ASN-first-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address which is generated for fetching the ASN-first-table entry designates a location which is not available in the configuration, an addressing

exception is recognized, and the operation is suppressed.

Bit 0 of the four-byte AFT entry specifies whether the corresponding AST is available. If this bit is one, an AFX-translation exception is recognized. The entry fetched from the AFT is used to access the AST.

ASN-Second-Table Lookup

The ASX portion of the ASN, in conjunction with the ASN-second-table origin contained in the ASN-first-table entry, is used to select an entry from the ASN second table.

The 31-bit real address of the ASN-second-table entry is obtained by appending six zeros on the right to bits 1-25 of the ASN-first-table entry and adding the ASX with six rightmost and 19 leftmost zeros appended. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31} - 1$ to zero. The 31-bit address is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

The fetch of the 64 bytes of the ASN-second-table entry appears to be word concurrent as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining 15 words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address which is generated for fetching the ASN-second-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the ASN-second-table entry specifies whether the address space is accessible. If this bit is one, an ASX-translation exception is recognized.

Recognition of Exceptions during ASN Translation

The exceptions which can be encountered during the ASN-translation process are collectively referred to as ASN-translation exceptions. A list of these exceptions and their priorities is given in Chapter 6, "Interruptions".

ASN Authorization

ASN authorization is the process of testing whether the program associated with the current authorization index is permitted to establish a particular address space. The ASN authorization is performed as part of PROGRAM TRANSFER with space switching (PT-ss) and SET SECONDARY ASN with space switching (SSAR-ss) and may be performed as part of LOAD ADDRESS SPACE PARAMETERS. ASN authorization is performed after the ASN-translation process for these instructions.

ASN authorization is also performed as part of PROGRAM RETURN when the restored secondary ASN does not equal the restored primary ASN. ASN authorization of the restored secondary ASN is performed after ASN translation of the restored secondary ASN.

When performed as part of PT-ss, the ASN authorization tests whether the ASN can be established as the primary ASN and is called primary-ASN authorization. When performed as part of LOAD ADDRESS SPACE PARAMETERS, PROGRAM RETURN, or SSAR-ss, the ASN authorization tests whether the ASN can be established as the secondary ASN and is called secondary-ASN authorization.

The ASN authorization is performed by means of an authority table in real storage which is designated by the authority-table-origin and authority-table-length fields in the ASN-second-table entry.

ASN-Authorization Controls

ASN authorization uses the authority-table origin and the authority-table length from the ASN-second-table entry, together with an authorization index.

Control Register 4

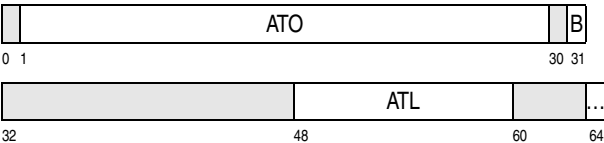
For PT-ss and SSAR-ss, the current contents of control register 4 include the authorization index. For LOAD ADDRESS SPACE PARAMETERS and PROGRAM RETURN, the value which will become the new contents of control register 4 is used. The register has the following format:

...	AX	...
32		48

Authorization Index (AX): Bits 32-47 of control register 4 are used as an index to locate the authority bits in the authority table.

ASN-Second-Table Entry

The ASN-second-table entry which is fetched as part of the ASN translation process contains information which is used to designate the authority table. An entry in the ASN second table has the following format:



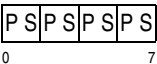
Authority-Table Origin (ATO): Bits 1-29, with two zeros appended on the right, are used to form a 31-bit real address that designates the beginning of the authority table.

Authority-Table Length (ATL): Bits 48-59 specify the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four bytes, is equal to one more than the ATL value. The contents of the length field are used to establish whether the entry designated by the authorization index falls within the authority table.

Authority-Table Entries

The authority table consists of entries of two bits each; accordingly, each byte of the authority table contains four entries in the following format:

The fields are allocated as follows:



Primary Authority (P): The left bit of an authority-table entry controls whether the program with the authorization index corresponding to the entry is permitted to establish the address space as a primary address space. If the P bit is one, the establishment is permitted. If the P bit is zero, the establishment is not permitted.

Secondary Authority (S): The right bit of an authority-table entry controls whether the program with the corresponding authorization index is permitted to establish the address space as a secondary address space. If the S bit is one, the establishment is permitted. If the S bit is zero, the establishment is not permitted.

ted to establish the address space as a secondary address space. If the S bit is one, the establishment is permitted. If the S bit is zero, the establishment is not permitted.

The authority table is also used in the extended-authorization process, as part of access-register translation. Extended authorization is described in “Authorizing the Use of the Access-List Entry” on page 5-59.

ASN-Authorization Process

This section describes the ASN-authorization process as it is performed during the execution of PROGRAM TRANSFER with space switching and SET SECONDARY ASN with space switching. For these two instructions, the ASN-authorization process is performed by using the authorization index currently in control register 4. Secondary authorization for PROGRAM RETURN, when the restored secondary ASN does not equal the restored primary ASN, and for LOAD ADDRESS SPACE PARAMETERS is the same, except that the value which will become the new contents of control register 4 is used for the authorization index. Also, for LOAD ADDRESS SPACE PARAMETERS, a secondary-authority exception does not occur. Instead, such a condition is indicated by the condition code.

The ASN-authorization process is performed by using the authorization index, in conjunction with the authority-table origin and length from the AST entry, to select an authority-table entry. The entry is fetched, and either the primary- or the secondary-authority bit is examined, depending on whether the primary- or secondary-ASN-authorization process is being performed. The ASN-authorization process is shown in Figure 3-9.

Authority-Table Lookup

The authorization index, in conjunction with the authority-table origin contained in the ASN-second-table entry, is used to select an entry from the authority table.

The authorization index is contained in bit positions 32-47 of control register 4.

Bit positions 1-29 of the AST entry contain the left-most 29 bits of the 31-bit real address of the authority table (ATO), and bit positions 48-59 contain the length of the authority table (ATL).

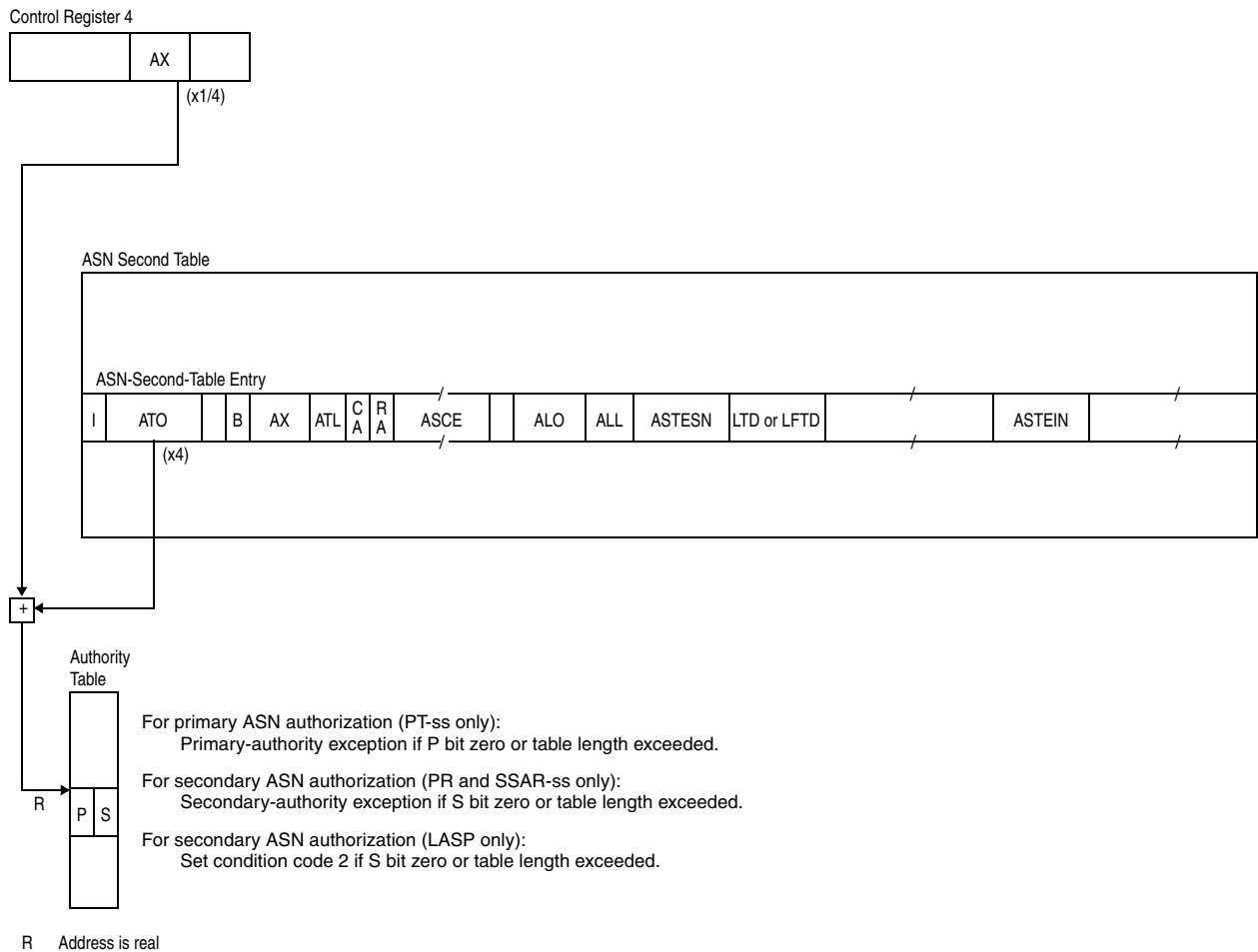


Figure 3-9. ASN Authorization

The 31-bit real address of a byte in the authority table is obtained by appending two zeros on the right to the authority-table origin and adding the 14 leftmost bits of the authorization index with 17 zeros appended on the left. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31} - 1$ to zero. The 31-bit address is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

As part of the authority-table-entry-lookup process, bits 0-11 of the authorization index are compared against the authority-table length. If the compared portion is greater than the authority-table length, a primary-authority exception or secondary-authority exception is recognized for PT-ss or SSAR-ss, respectively. For LOAD ADDRESS SPACE PARAMETER-

TERS, when the authority-table length is exceeded, condition code 2 is set.

The fetch access to the byte in the authority table is not subject to protection. When the storage address which is generated for fetching the byte designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

The byte contains four authority-table entries of two bits each. The rightmost two bits of the authorization index, bits 46 and 47 of control register 4, are used to select one of the four entries. The left or right bit of the entry is then tested, depending on whether the authorization test is for a primary ASN or a secondary ASN. The following table shows the bit which is selected from the byte as a function of bits 46 and 47 of the authorization index and the instruction PT-ss,

SSAR-ss, PROGRAM RETURN, or LOAD ADDRESS SPACE PARAMETERS.

Authorization-Index Bits		Bit Selected from Authority-Table Byte for Test	
46	47	P Bit (PT-ss)	S Bit (SSAR-ss, PR, or LASP)
0	0	0	1
0	1	2	3
1	0	4	5
1	1	6	7

If the selected bit is one, the ASN is authorized, and the appropriate fields in the AST entry are loaded into the appropriate control registers. If the selected bit is zero, the ASN is not authorized, and a primary-authority exception is recognized for PT-ss or a secondary-authority exception is recognized for SSAR-ss or PROGRAM RETURN. For LOAD ADDRESS SPACE PARAMETERS, when the ASN is not authorized, condition code 2 is set.

Recognition of Exceptions during ASN Authorization

The exceptions which can be encountered during the primary- and secondary-ASN-authorization processes and their priorities are described in the definitions of the instructions in which ASN authorization is performed.

Programming Note: The primary- and secondary-authority exceptions cause nullification in order to permit dynamic modification of the authority table. Thus, when an address space is created or “swapped in,” the authority table can first be set to all zeros and the appropriate authority bits set to one only when required.

Dynamic Address Translation

Dynamic address translation (DAT) provides the ability to interrupt the execution of a program at an arbitrary moment, record it and its data in auxiliary storage, such as a direct-access storage device, and at a later time return the program and the data to different main-storage locations for resumption of execution. The transfer of the program and its data between main and auxiliary storage may be performed piecemeal, and the return of the information to main storage may take place in response to an

attempt by the CPU to access it at the time it is needed for execution. These functions may be performed without change or inspection of the program and its data, do not require any explicit programming convention for the relocated program, and do not disturb the execution of the program except for the time delay involved.

With appropriate support by an operating system, the dynamic-address-translation facility may be used to provide to a user a system wherein storage appears to be larger than the main storage which is available in the configuration. This apparent main storage is referred to as virtual storage, and the addresses used to designate locations in the virtual storage are referred to as virtual addresses. The virtual storage of a user may far exceed the size of the main storage which is available in the configuration and normally is maintained in auxiliary storage. The virtual storage is considered to be composed of blocks of addresses, called pages. Only the most recently referred-to pages of the virtual storage are assigned to occupy blocks of physical main storage. As the user refers to pages of virtual storage that do not appear in main storage, they are brought in to replace pages in main storage that are less likely to be needed. The swapping of pages of storage may be performed by the operating system without the user’s knowledge.

The sequence of virtual addresses associated with a virtual storage is called an address space. With appropriate support by an operating system, the dynamic-address-translation facility may be used to provide a number of address spaces. These address spaces may be used to provide degrees of isolation between users. Such support can consist of a completely different address space for each user, thus providing complete isolation, or a shared area may be provided by mapping a portion of each address space to a single common storage area. Also, instructions are provided which permit a semiprivileged program to access more than one such address space. Dynamic address translation provides for the translation of virtual addresses from multiple different address spaces without requiring that the translation parameters in the control registers be changed. These address spaces are called the primary address space, secondary address space, and AR-specified address spaces. A privileged program can also cause the home address space to be accessed.

In the process of replacing blocks of main storage by new information from an external medium, it must be

determined which block to replace and whether the block being replaced should be recorded and preserved in auxiliary storage. To aid in this decision process, a reference bit and a change bit are associated with the storage key.

Dynamic address translation may be specified for instruction and data addresses generated by the CPU but is not available for the addressing of data and of CCWs, IDAWs, and MIDAWs in I/O operations. The CCW-indirect-data-addressing facility and modified-CCW-indirect-data-addressing facilities are provided to aid I/O operations in a virtual-storage environment.

Address computation can be carried out in the 24-bit, 31-bit, or 64-bit addressing mode. When address computation is performed in the 24-bit or 31-bit addressing mode, 40 or 33 zeros, respectively, are appended on the left to form a 64-bit address. Therefore, the resultant logical address is always 64 bits in length. The real or absolute address that is formed by dynamic address translation, and the absolute address that is then formed by prefixing, are always 64 bits in length.

Dynamic address translation is the process of translating a virtual address during a storage reference into the corresponding real or absolute address. The virtual address may be a primary virtual address, secondary virtual address, AR-specified virtual address, or home virtual address. These addresses are translated by means of the primary, the secondary, an AR-specified, or the home address-space-control element, respectively. After selection of the appropriate address-space-control element, the translation process is the same for all of the four types of virtual address. An address-space-control element may be a segment-table designation specifying a 2G-byte address space, a region-table designation specifying a 4T-byte, 8P-byte, or 16E-byte space, or a real-space designation specifying a 16E-byte space. (The letters K, M, G, T, P, and E represent kilo, 2^{10} , mega, 2^{20} , giga, 2^{30} , tera, 2^{40} , peta, 2^{50} , and exa, 2^{60} , respectively.) A segment-table designation or region-table designation causes translation to be performed by means of tables established by the operating system in real or absolute storage. A real-space designation causes the virtual address simply to be treated as a real address, without the use of tables in storage.

In the process of translation when using a segment-table designation or a region-table designation, three

types of units of information are recognized — regions, segments, and pages. A region is a block of sequential virtual addresses spanning 2G bytes and beginning at a 2G-byte boundary. A segment is a block of sequential virtual addresses spanning 1M bytes and beginning at a 1M-byte boundary. A page is a block of sequential virtual addresses spanning 4K bytes and beginning at a 4K-byte boundary.

The virtual address, accordingly, is divided into four principal fields. Bits 0-32 are called the region index (RX), bits 33-43 are called the segment index (SX), bits 44-51 are called the page index (PX), and bits 52-63 are called the byte index (BX). The virtual address has the following format:

RX											SX			PX		BX																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
0													33		44																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							

As determined by its address-space-control element, a virtual address space may be a 2G-byte space consisting of one region, or it may be up to a 16E-byte space consisting of up to 8G regions. The RX part of a virtual address applying to a 2G-byte address space must be all zeros; otherwise, an exception is recognized.

The RX part of a virtual address is itself divided into three fields. Bits 0-10 are called the region first index (RFX), bits 11-21 are called the region second index (RSX), and bits 22-32 are called the region third index (RTX). Bits 0-32 of the virtual address have the following format:

RFX											RSX											RTX																					
0											11											22											33										

A virtual address in which the RTX is the leftmost significant part (a 42-bit address) is capable of addressing 4T bytes (2K regions), one in which the RSX is the leftmost significant part (a 53-bit address) is capable of addressing 8P bytes (4M regions), and one in which the RFX is the leftmost significant part (a 64-bit address) is capable of addressing 16E bytes (8G regions).

A virtual address in which the RX is always zero can be translated into real addresses by means of one or two translation tables, as follows:

- When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control, bit 53 of the segment-table entry, is zero, the

virtual address can be translated by means of a segment table and a page table.

- When enhanced DAT applies and the STE-format control of the segment-table entry is one, the virtual address can be translated by means of a segment table only.

If the RX may be nonzero, from one to three additional translation tables are required, as follows. If the RFX may be nonzero, a region first table, region second table, and region third table are required. If the RFX is always zero but the RSX may be nonzero, a region second table and region third table are required. If the RFX and RSX are always zero but the RTX may be nonzero, a region third table is required. An exception is recognized if the address-space-control element for an address space does not designate the highest level of table (beginning with the region first table and continuing downward to the segment table) needed to translate a reference to the address space.

A region first table, region second table, or region third table is sometimes referred to simply as a region table. Similarly, a region-first-table designation, region-second-table designation, or region-third-table designation is sometimes referred to as a region-table designation.

The region, segment, and, when applicable, page tables reflect the current assignment of real or absolute storage.

When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control, bit 53 of the segment-table entry, is zero, the assignment of real storage occurs in units of pages, the real locations being assigned contiguously within a page. The pages need not be adjacent in real storage even though assigned to a set of sequential virtual addresses.

Similarly, when enhanced DAT applies and the STE-format control of the segment-table entry is one, the assignment of real storage occurs in units of segments, the absolute locations being assigned contiguously within a segment. The segments need not be adjacent in absolute storage even though assigned to a set of sequential virtual addresses.

To improve performance, translation normally is performed by means of table copies maintained in a

special buffer called the translation-lookaside buffer (TLB). The TLB may also contain entries that provide the virtual-equals-real translation specified by a real-space designation.

Translation Control

Address translation is controlled by three bits in the PSW and by a set of bits referred to as the translation parameters. The translation parameters are in control registers 0, 1, 7, and 13. Additional controls are located in the translation tables.

Additional controls are provided as described in Chapter 5, "Program Execution." These controls determine whether the contents of each access register can be used to obtain an address-space-control element for use by DAT.

Translation Modes

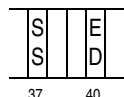
The three bits in the PSW that control dynamic address translation are bit 5, the DAT-mode bit, and bits 16 and 17, the address-space-control bits. When the DAT-mode bit is zero, then DAT is off, and the CPU is in the real mode. When the DAT-mode bit is one, then DAT is on, and the CPU is in the translation mode designated by the address-space-control bits: 00 designates the primary-space mode, 01 designates the access-register mode, 10 designates the secondary-space mode, and 11 designates the home-space mode. The various modes are shown in Figure 3-10, along with the handling of addresses in each mode.

PSW Bit				Mode	Handling of Addresses	
5	16	17	DAT		Instruction Addresses	Logical Addresses
0	0	0	Off	Real mode	Real	Real
0	0	1	Off	Real mode	Real	Real
0	1	0	Off	Real mode	Real	Real
0	1	1	Off	Real mode	Real	Real
1	0	0	On	Primary-space mode	Primary virtual	Primary virtual
1	0	1	On	Access-register mode	Primary virtual	AR-specified virtual
1	1	0	On	Secondary-space mode	Primary virtual	Secondary virtual
1	1	1	On	Home-space mode	Home virtual	Home virtual

Figure 3-10. Translation Modes

Control Register 0

Bit 37 is provided in control register 0 for use in controlling dynamic address translation. When the enhanced-DAT facility is installed, bit 40 is also provided for use in controlling dynamic-address translation. The bit assignments are as follows:



Secondary-Space Control (SS): Bit 37 of control register 0 is the secondary-space-control bit. When this bit is zero and execution of MOVE TO PRIMARY, MOVE TO SECONDARY, or SET ADDRESS SPACE CONTROL is attempted, a special-operation exception is recognized. When this bit is one, it indicates that the region table or segment table designated by the secondary address-space-control element is attached when the CPU is in the primary-space mode.

Enhanced-DAT-Enablement Control (ED): When the enhanced-DAT facility is installed, bit 40 of control register 0 is the enhanced-DAT-enablement control. When this bit is zero, dynamic address translation proceeds as though the enhanced-DAT facility was not installed. When the bit is one, the following conditions apply:

- The DAT-protection bit is defined in bit position 54 of each region-table entry.
- The STE-format control is defined in bit position 53 of the segment-table entry. When the STE-format control is zero, the following apply:
 - Bits 0-52 of the segment-table entry are used to locate the page table (as occurs when the enhanced-DAT facility is not installed or enabled)
 - Bit 53 of the page-table entry contains the change-recording override for the page.

When the STE-format control is one, the following apply:

- Bits 0-43 of the segment-table entry form the segment-frame absolute address. There is no designation of a page table, and no page-table entries are used.

- Bit 47 of the segment-table entry determines the validity of the access-control bits and fetch-protection bit (in bits 48-52 of the STE).
- Bits 48-52 of the segment-table entry contain access-control bits and a fetch-protection bit for the segment.
- Bit 55 the segment-table entry is the change-recording override for the segment.

When the enhanced-DAT facility is not installed, bit 40 of control register 0 is reserved and should contain zero; otherwise, the program may not operate compatibly in the future.

Enhanced-DAT Terminology: For the purpose of brevity, the following terms are used in conjunction with the enhanced-DAT facility:

- The term “*enhanced DAT applies*” refers to the case where all of the following are true:
 - The enhanced-DAT facility is installed.
 - The enhanced-DAT-enablement control, bit 40 of control register 0, is one.
 - The address is translated by means of DAT-table entries.
- The term “*enhanced-DAT does not apply*” refers to the case where any of the following are true:
 - The enhanced-DAT facility is not installed
 - The enhanced-DAT facility is installed, but the enhanced-DAT-enablement control is zero
 - The address is not translated by means of DAT-table entries (that is, DAT is off; DAT is on, but the ASCE designates a real space; or the instruction uses a real address such as LOAD USING REAL ADDRESS or STORE USING REAL ADDRESS).
 - A real address is implicitly used, for example in the handling of an interruption, CPU logout, or fetching of table entries for ART, ASN translation, ASN authorization, DAT, or PC-number translation.

Programming Notes:

1. If the program changes the state of the enhanced-DAT-enablement control, then it should also clear all entries from all TLBs in the configuration by executing any of the following instructions:

- PURGE TLB on all CPUs in the configuration while DAT is off
- COMPARE AND SWAP AND PURGE that purges the TLB
- SIGNAL PROCESSOR that sets the prefix of all CPUs in the configuration
- SIGNAL PROCESSOR that sets the architectural mode

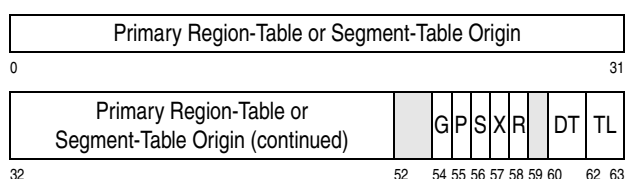
2. Before DAT is enabled or implicitly performed (such as in LOAD REAL ADDRESS), the program should ensure that the enhanced-DAT-enablement control is consistent with the enhanced-DAT-enablement control in any other CPU in the configuration that may be performing DAT.

Failure to follow these rules may result in unpredictable results, including the possibility of a delayed-access machine-check condition being recognized.

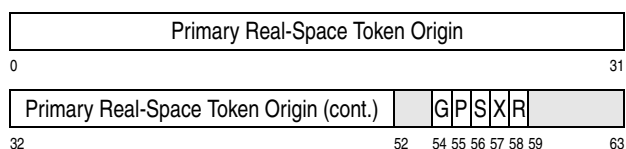
Control Register 1

Control register 1 contains the primary address-space-control element (PASCE). The register has one of the following two formats, depending on the real-space-control bit (R) in the register:

Primary Region-Table or Segment-Table Designation (R=0)



Primary Real-Space Designation (R=1)



The fields in the primary address-space-control element are allocated as follows:

Primary Region-Table or Segment-Table Origin: Bits 0-51 of the primary region-table or segment-table designation in control register 1, with 12 zeros appended on the right, form a 64-bit address that

designates the beginning of the primary region table or segment table. It is unpredictable whether the address is real or absolute. This table is called the primary region table or segment table since it is used to translate virtual addresses in the primary address space.

Primary Subspace-Group Control (G): Bit 54 of control register 1, when one, indicates that the address space specified by the PASCE is the base space or a subspace of a subspace group. When bit 54 is zero, the address space is not in a subspace group.

Primary Private-Space Control (P): If bit 55 of control register 1 is one, then (1) a one value of the common-segment bit in a translation-lookaside-buffer (TLB) representation of a segment-table entry prevents the entry and the TLB page-table copy it designates from being used when translating references to the primary address space, even with a match between the table or token origin in control register 1 and the table origin in the TLB entry, (2) low-address protection and fetch-protection override do not apply to the primary address space; and (3) a translation-specification exception is recognized if a reference to the primary address space is translated by means of a segment-table entry in storage and the common-segment bit is one in the entry. Item 2 in the above list applies even when the contents of control register 1 are a real-space designation.

Programming Note: With respect to item 1 in the above list, when the contents of control register 1 are a real-space designation, a one value of the common-segment bit in a TLB representation of a segment-table entry prevents the entry and the TLB page-table copy it designates from being used regardless of the value of the private-space control in the real-space designation.

Primary Storage-Alteration-Event Control (S): When the storage-alteration-space control in control register 9 is one, bit 56 of control register 1 specifies, when one, that the primary address space is one for which storage-alteration events can occur. Bit 56 is examined when the PASCE is used to perform dynamic-address translation for a storage-operand store reference. Bit 56 is ignored when the storage-alteration-space control is zero.

Primary Space-Switch-Event Control (X): When bit 57 of control register 1 is one:

- A space-switch-event program interruption occurs when execution of the space-switching form of PROGRAM CALL (PC-ss), PROGRAM RETURN (PR-ss), or PROGRAM TRANSFER (PT-ss) is completed. The interruption occurs if bit 57 is one either before or after the operation.
- A space-switch-event program interruption occurs upon completion of a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes the address space from which instructions are fetched either to or from the home address space; that is, when instructions are fetched from the home address space either before or after the operation but not both before and after the operation.
- Condition code 3 is set by LOAD ADDRESS SPACE PARAMETERS.

Primary Real-Space Control (R): If bit 58 of control register 1 is zero, the register contains a region-table or segment-table designation. If bit 58 is one, the register contains a real-space designation. When bit 58 is one, a one value of the common-segment bit in a translation-lookaside-buffer (TLB) representation of a segment-table entry prevents the entry and the TLB page-table copy it designates from being used when translating references to the primary address space, even with a match between the token origin in control register 1 and the table origin in the TLB entry.

Primary Designation-Type Control (DT): When R is zero, the type of table designation in control register 1 is specified by bits 60 and 61 in the register, as follows:

Bits 60 and 61	Designation Type
11	Region-first-table
10	Region-second-table
01	Region-third-table
00	Segment-table

When R is zero, bits 60 and 61 must be 11 binary when an attempt is made to use the PASCE to translate a virtual address in which the leftmost one bit is in bit positions 0-10 of the address. Similarly, bits 60 and 61 must be 11 or 10 binary when the leftmost one bit is in bit positions 11-21 of the address, and they must be 11, 10, or 01 binary when the leftmost

one bit is in bit positions 22-32 of the address. Otherwise, an ASCE-type exception is recognized.

Primary Region-Table or Segment-Table Length (TL): Bits 62 and 63 of the primary region-table designation or segment-table designation in control register 1 specify the length of the primary region table or segment table in units of 4,096 bytes, thus making the length of the region table or segment table variable in multiples of 512 entries. The length of the primary region table or segment table, in units of 4,096 bytes, is one more than the TL value. The contents of the length field are used to establish whether the portion of the virtual address (RFX, RSX, RTX, or SX) to be translated by means of the table designates an entry that falls within the table.

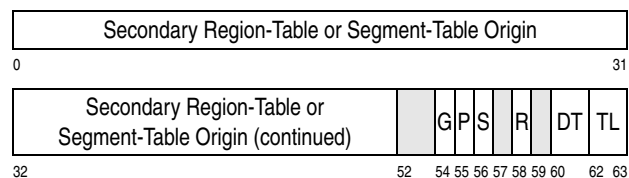
Primary Real-Space Token Origin: Bits 0-51 of the primary real-space designation in control register 1, with 12 zeros appended on the right, form a 64-bit address that may be used in forming and using TLB entries that provide a virtual-equals-real translation for references to the primary address space. Although this address is used only as a token and is not used to perform a storage reference, it still must be a valid address; otherwise, an incorrect TLB entry may be used when the contents of control register 1 are used.

The following bits of control register 1 are not assigned and are ignored: bits 52, 53, and 59 if the register contains a region-table designation or segment-table designation, and bits 52, 53 and 59-63 if the register contains a real-space designation.

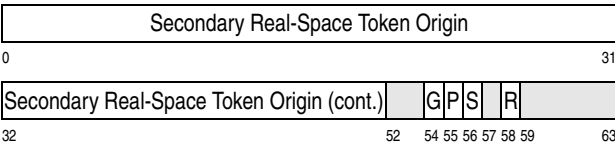
Control Register 7

Control register 7 contains the secondary address-space-control element (SASCE). The register has one of the following two formats, depending on the real-space-control bit (R) in the register:

Secondary Region-Table or Segment-Table Designation (R=0)



Secondary Real-Space Designation (R=1)



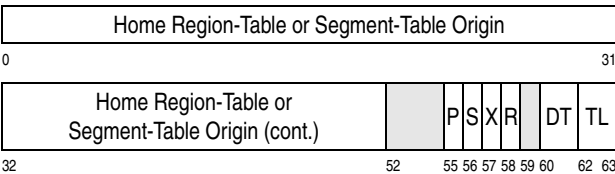
The secondary region-table origin, secondary segment-table origin, secondary subspace-group control (G), secondary private-space control (P), secondary storage-alteration-event control (S), secondary real-space control (R), secondary designation-type control (DT), secondary region-table or segment-table length (TL), and secondary real-space token origin in control register 7 are defined the same as the fields in the same bit positions in control register 1, except that control register 7 applies to the secondary address space.

The following bits of control register 7 are not assigned and are ignored: bits 52, 53, 57, and 59 if the register contains a region-table designation or segment-table designation, and bits 52, 53, 57, and 59-63 if the register contains a real-space designation.

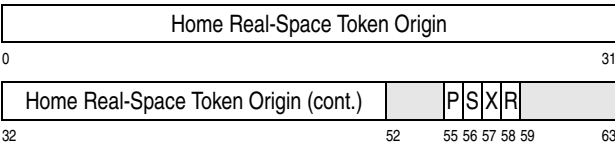
Control Register 13

Control register 13 contains the home address-space-control element (HASCE). The register has one of the following two formats, depending on the real-space-control bit (R) in the register:

Home Region-Table or Segment-Table Designation (R=0)



Home Real-Space Designation (R=1)



Home Space-Switch-Event Control (X): When bit 57 of control register 13 is one, a space-switch-event program interruption occurs upon completion of a

RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes the address space from which instructions are fetched either to or from the home address space; that is, when instructions are fetched from the home address space either before or after the operation but not both before and after the operation.

The home region-table origin, home segment-table origin, home private-space control (P), home storage-alteration-event control (S), home real-space control (R), home designation-type control (DT), home region-table or segment-table length (TL), and home real-space token origin in control register 13 are defined the same as the fields in the same bit positions in control register 1, except that control register 13 applies to the home address space.

The following bits of control register 13 are not assigned and are ignored: bits 52-54 and 59 if the register contains a region-table designation or segment-table designation, and bits 52-54 and 59-63 if the register contains a real-space designation.

Programming Notes:

1. The validity of the information loaded into a control register, including that pertaining to dynamic address translation, is not checked at the time the register is loaded. This information is checked and the program exception, if any, is indicated at the time the information is used.
2. The information pertaining to dynamic address translation is considered to be used when an instruction is executed with DAT on or when INVALIDATE DAT TABLE ENTRY, INVALIDATE PAGE TABLE ENTRY, LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, or STORE REAL ADDRESS is executed. The information is not considered to be used when the PSW specifies translation but an I/O, external, restart, or machine-check interruption occurs before an instruction is executed, or when the PSW specifies the wait state.

Translation Tables

When the address-space-control element (ASCE) used in a translation is a region-first-table designation, the translation process consists in a five-level lookup using five tables: a region first table, a region

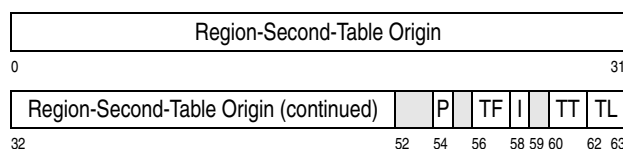
second table, a region third table a segment table, and a page table. These tables reside in real or absolute storage. When the ASCE is a region-second-table designation, region-third-table designation, or segment-table designation, the lookups in the levels of tables above the designated level are omitted, and the higher-level tables themselves are omitted.

Region-Table Entries

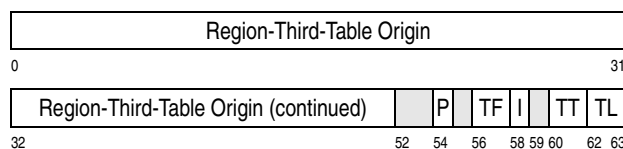
The term “region-table entry” means a region-first-table entry, region-second-table entry, or region-third-table entry.

The entries fetched from the region first table, region second table, and region third table have the following formats. The level (first, second, or third) of the table containing an entry is identified by the table-type (TT) bits in the entry.

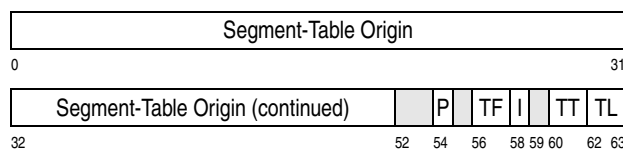
Region-First-Table Entry (TT=11)



Region-Second-Table Entry (TT=10)



Region-Third-Table Entry (TT=01)



The fields in the three levels of region-table entries are allocated as follows:

Region-Second-Table Origin, Region-Third-Table Origin, and Segment-Table Origin: A region-first-table entry contains a region-second-table origin. A region-second-table entry contains a region-third-table origin. A region-third-table entry contains a segment-table origin. The following description applies to

each of the three origins. Bits 0-51 of the entry, with 12 zeros appended on the right, form a 64-bit address that designates the beginning of the next-lower-level table. It is unpredictable whether the address is real or absolute.

DAT-Protection Bit (P): When enhanced DAT applies, bit 54 is treated as being ORed with the DAT-protection bit in each subsequent region-table entry, segment-table entry, and, when applicable, page-table entry used in the translation. Thus, when the bit is one, DAT protection applies to the entire region or regions specified by the region-table entry.

When the enhanced-DAT facility is not installed, or when the facility is installed but the enhanced-DAT-enablement control is zero, bit 54 of the region-table entry is ignored.

Region-Second-Table Offset, Region-Third-Table Offset, and Segment-Table Offset (TF): A region-first-table entry contains a region-second-table offset. A region-second-table entry contains a region-third-table offset. A region-third-table entry contains a segment-table offset. The following description applies to each of the three offsets. Bits 56 and 57 of the entry specify the length of a portion of the next-lower-level table that is missing at the beginning of the table, that is, the bits specify the location of the first entry actually existing in the next-lower-level table. The bits specify the length of the missing portion in units of 4,096 bytes, thus making the length of the missing portion variable in multiples of 512 entries. The length of the missing portion, in units of 4,096 bytes, is equal to the TF value. The contents of the offset field, in conjunction with the length field, bits 62 and 63, are used to establish whether the portion of the virtual address (RSX, RTX, or SX) to be translated by means of the next-lower-level table designates an entry that actually exists in the table.

Region-Invalid Bit (I): Bit 58 in a region-first-table entry or region-second-table entry controls whether the set of regions associated with the entry is available. Bit 58 in a region-third-table entry controls whether the single region associated with the entry is available. When bit 58 is zero, address translation proceeds by using the region-table entry. When the bit is one, the entry cannot be used for translation.

Table-Type Bits (TT): Bits 60 and 61 of the region-first-table entry, region-second-table entry, and

region-third-table entry identify the level of the table containing the entry, as follows:

Bits 60 and 61	Region-Table Level
11	First
10	Second
01	Third

Bits 60 and 61 must identify the correct table level, considering the type of table designation that is the address-space-control element being used in the translation and the number of table levels that have so far been used; otherwise, a translation-specification exception is recognized.

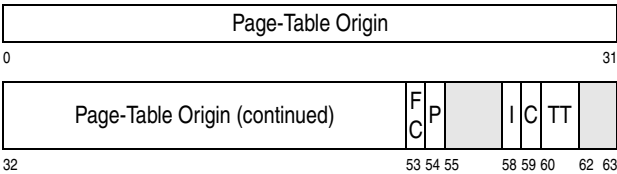
Region-Second-Table Length, Region-Third-Table Length, and Segment-Table Length (TL): A region-first-table entry contains a region-second-table length. A region-second-table entry contains a region-third-table length. A region-third-table entry contains a segment-table length. The following description applies to each of the three lengths. Bits 62 and 63 of the entry specify the length of the next-lower-level table in units of 4,096 bytes, thus making the length of the table variable in multiples of 512 entries. The length of the next-lower-level table, in units of 4,096 bytes, is one more than the TL value. The contents of the length field, in conjunction with the offset field, bits 56 and 57, are used to establish whether the portion of the virtual address (RSX, RTX, or SX) to be translated by means of the next-lower-level table designates an entry that actually exists in the table.

All other bit positions of the region-table entry are reserved for possible future extensions and should contain zeros; otherwise, the program may not operate compatibly in the future. When enhanced DAT applies, the reserved bit positions of the region-table entry should contain zeros even if the table entry is invalid.

Segment-Table Entries

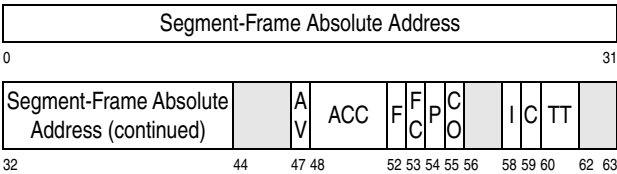
When enhanced DAT does not apply, or when enhanced DAT applies and the STE-format control, bit 53 of the segment-table entry is zero, the entry fetched from the segment table has the following format:

Segment-Table Entry (TT=00, FC=0)



When enhanced DAT applies and the STE-format control is one, the entry fetched from the segment table has the following format:

Segment-Table Entry (TT=00, FC=1)



The fields in the segment-table entry are allocated as follows:

Page-Table Origin: When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control, bit 53 of the segment-table entry, is zero, bits 0-52, with 11 zeros appended on the right, form a 64-bit address that designates the beginning of a page table. It is unpredictable whether the address is real or absolute.

Segment-Frame Absolute Address (SFAA): When enhanced DAT applies and the STE-format control is one, bits 0-43 of the entry, with 20 zeros appended on the right, form the 64-bit absolute address of the segment.

ACCF-Validity Control (AV): When enhanced DAT applies and the STE-format control is one, bit 47 is the access-control-bits and fetch-protection bit (ACCF) validity control. When the AV control is zero, bits 48-52 of the segment-table entry are ignored. When the AV control is one, bits 48-52 are used as described below.

Access-Control Bits (ACC): When enhanced DAT applies, the STE-format control is one, and the AV control is one, bits 48-51 of the segment-table entry contain the access-control bits that may be used for any key-controlled access checking that applies to the address. It is unpredictable whether the CPU uses these bits or the access-control bits in the stor-

age key of the 4K-byte block corresponding to the address.

Fetch-Protection Bit (F): When enhanced DAT applies, the STE-format control is one, and the AV control is one, bit 52 of the segment-table entry contains the fetch-protection bit that may be used for any key-controlled access checking that applies to the address. It is unpredictable whether the CPU uses this bit or the fetch-protection bit in the storage key of the 4K-byte block corresponding to the address.

STE-Format Control (FC): When enhanced DAT applies, bit 53 is the format control for the segment-table entry, as follows:

- When the FC bit is zero, bits 0-52 of the entry form the page-table origin, and bit 55 is reserved.
- When the FC bit is one, bits 0-43 of the entry form the segment-frame absolute address, bit 47 is the ACCF-validity control, bits 48-51 are the access-control bits, bit 52 is the fetch-protection bit, and bit 55 is the change-recording override.

When enhanced DAT does not apply, bit 53 is ignored.

DAT-Protection Bit (P): Bit 54, when one, indicates that DAT protection applies to the entire segment.

When enhanced DAT does not apply, bit 54 is treated as being ORed with the DAT-protection bit in the page-table entry used in the translation.

When enhanced DAT applies, the DAT-protection bit in any and all region-table entries used in the translation are treated as being ORed with the DAT-protection bit in the segment-table entry; when the STE-format control is zero, the DAT-protection bit in the STE is further treated as being ORed with the DAT-protection bit in the page-table entry.

Change-Recording Override (CO): When enhanced DAT applies, and the STE-format control is one, bit 55 of the segment-table entry is the change-recording override for the segment. See “Change-Recording Override” on page 3-17 for a discussion of change-recording-override processing.

When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control is zero, bit 55 of the segment-table entry is ignored.

Segment-Invalid Bit (I): Bit 58 controls whether the segment associated with the segment-table entry is available. When the bit is zero, address translation proceeds by using the segment-table entry. When the bit is one, the segment-table entry cannot be used for translation.

Common-Segment Bit (C): Bit 59 controls the use of the translation-lookaside-buffer (TLB) copies of the segment-table entry. When enhanced DAT does not apply or when enhanced DAT applies but the format control is zero, bit 59 also controls the use of the TLB copies of the page table designated by the segment-table entry. A zero identifies a private segment; in this case, the segment-table entry and the page table it designates may be used only in association with the segment-table origin that designates the segment table in which the segment-table entry resides. A one identifies a common segment; in this case, the segment-table entry and any page table it designates may continue to be used for translating addresses corresponding to the segment index, even though a different segment table is specified. However, TLB copies of the segment-table entry and any page table for a common segment are not usable if the private-space control, bit 55, is one in the address-space-control element used in the translation or if that address-space-control element is a real-space designation. The common-segment bit must be zero if the segment-table entry is fetched from storage during a translation when the private-space control is one in the address-space-control element being used; otherwise, a translation-specification exception is recognized.

Table-Type Bits (TT): Bits 60 and 61 of the segment-table entry are 00 binary to identify the level of the table containing the entry. The meanings of all possible values of bits 60 and 61 in a region-table entry or segment-table entry are as follows:

Bits 60 and 61	Table Level
11	Region-first
10	Region-second
01	Region-third
00	Segment

Bits 60 and 61 must identify the correct table level, considering the type of table designation that is the address-space-control element being used in the translation and the number of table levels that have so far been used; otherwise, a translation-specification exception is recognized.

All other bit positions of the segment-table entry are reserved for possible future extensions and should contain zeros; otherwise, the program may not operate compatibly in the future. When enhanced DAT applies, the reserved bit positions of the segment-table entry should contain zeros even if the table entry is invalid.

Programming Note: When the common-segment (C) bit is one in a valid segment-table entry fetched from storage during dynamic-address translation for a non-private address space, that is, for an address space in which the private-space control is zero in the address-space-control element being used, then the following conditions apply:

1. The segment-table entries corresponding to the same virtual address in all other non-private address spaces must be identical.

If the program alters such a segment-table entry in one non-private address space, then it must also (a) identically alter all other segment-table entries corresponding to the same virtual address in all other non-private address spaces, and (b) ensure that the affected entries are cleared from the TLBs of all CPUs in the configuration. Further information on clearing TLB entries may be found in “Modification of Translation Tables” on page 3-55.

2. The program must ensure that segment-table entries in which the C bit is set to one, and any region-table entries used by the DAT process to locate such segment-table entries, are consistent across all address spaces. That is, if the DAT process can successfully locate a segment-table entry in which the C bit is one in one non-private address space, then there must be no exception condition that prevents DAT from locating the segment-table entry corresponding to the same virtual address in any other non-private address spaces.

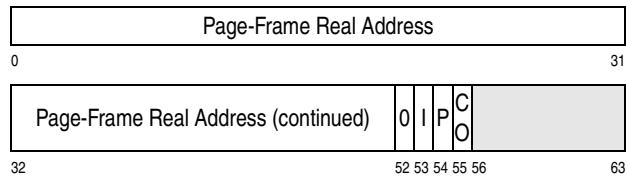
If the program alters such a segment-table entry or region-table entry in one non-private address space, then it must also (a) perform consistent alteration to all corresponding table entries in all other non-private address spaces, and (b) ensure that the affected entries are cleared from the TLBs of all CPUs in the configuration.

3. If the program fails to maintain consistent DAT table entries as described above, results are unpredictable and may include the presentation

of a delayed-access-exception machine check, as described in “Delayed Access Exception” on page 11-16.

Page-Table Entries

The entry fetched from the page table entry has the following format:



The fields in the page-table entry are allocated as follows:

Page-Frame Real Address (PFRA): Bits 0-51 provide the leftmost bits of a real storage address. When these bits are concatenated with the 12-bit byte-index field of the virtual address on the right, a 64-bit real address is obtained.

Page-Invalid Bit (I): Bit 53 controls whether the page associated with the page-table entry is available. When the bit is zero, address translation proceeds by using the page-table entry. When the bit is one, the page-table entry cannot be used for translation.

DAT-Protection Bit (P): Bit 54 controls whether store accesses can be made in the page. This protection mechanism is in addition to the key-controlled-protection and low-address-protection mechanisms. The bit has no effect on fetch accesses. If the bit is zero, stores are permitted to the page, subject to the following additional constraints:

- The DAT-protection bit being zero in the segment-table entry used in the translation,
- When enhanced DAT applies, the DAT-protection bit being zero in all region-table entries used in the translation,
- Other protection mechanisms

If the bit is one, stores are disallowed. When no higher priority exception conditions exist, an attempt to store when the DAT-protection bit is one causes a protection exception to be recognized. The DAT-protection bit in the segment-table entry is treated as being ORed with bit 54 when determining whether DAT protection applies to the page. When enhanced

DAT applies, the DAT-protection bits in any region-table entries used in translation are also treated as being ORed with bit 54 when determining whether DAT-protection applies.

Change-Recording Override (CO): When enhanced DAT does not apply, bit 55 of the page-table entry must contain zero; otherwise, a translation-specification exception is recognized as part of the execution of an instruction using that entry for address translation. When enhanced DAT applies and the STE-format control is zero, bit 55 of the page-table entry is the change-recording override for the page. See “Change-Recording Override” on page 3-17 for a discussion of change-recording-override processing.

Bit position 52 of the entry must contain zero; otherwise, a translation-specification exception is recognized as part of the execution of an instruction using that entry for address translation.

Bit positions 56-63 are not assigned and are ignored.

Translation Process

This section describes the translation process as it is performed implicitly before a virtual address is used to access main storage. Explicit translation, which is the process of translating the operand address of LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, STORE REAL ADDRESS, and TEST PROTECTION, is the same, except that, for LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS and TEST PROTECTION, region-first-translation, region-second-translation, region-third-translation, and segment-translation exceptions are not recognized, and for LOAD REAL ADDRESS and TEST PROTECTION, page-translation exceptions are not recognized; such conditions are instead indicated by the condition code. Translation of the operand address of LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS and STORE REAL ADDRESS also differs in that the CPU may be in the real mode.

Translation of a virtual address is controlled by the DAT-mode bit and address-space-control bits in the PSW and by the address-space-control elements (ASCEs) in control registers 1, 7, and 13 and as specified by the access registers. When the ASCE used in a translation is a region-first-table designa-

tion, the translation is performed by means of a region first table, region second table, region third table, segment table, and page table, all of which reside in real or absolute storage. When the ASCE is a lower-level type of table designation (region-second-table designation, region-third-table designation, or segment-table designation) the translation is performed by means of only the table levels beginning with the designated level, and the virtual-address bits that would, if nonzero, require use of a higher level or levels of table must be all zeros; otherwise, an ASCE-type exception is recognized. When the ASCE is a real-space designation, the virtual address is treated as a real address, and table entries in real or absolute storage are not used.

The address-space-control element (ASCE) used for a particular address translation is called the effective ASCE. Accordingly, when a primary virtual address is translated, the contents of control register 1 are used as the effective ASCE. Similarly, for a secondary virtual address, the contents of control register 7 are used; for an AR-specified virtual address, the ASCE specified by the access register is used; and for a home virtual address, the contents of control register 13 are used.

When the real-space control in the effective ASCE is zero, the designation-type control in the ASCE specifies the table-designation type of the ASCE: region-first-table designation, region-second-table designation, region-third-table designation, or segment-table designation. The corresponding portion of the virtual address (region first index, region second index, region third index, or segment index) is checked against the table-length field in the designation, and it is added to the origin in the designation to select an entry in the designated table. If the selected entry is outside its table, as determined by the table-length field in the designation, or if the I bit is one in the selected entry, a region-first-translation, region-second-translation, region-third-translation, or segment-translation exception is recognized, depending on the table level specified by the designation. If the table-type bits in the selected entry do not indicate the expected table level, a translation-specification exception is recognized.

The table entry selected by means of the effective ASCE designates the next-lower-level table to be used. If the current table is a region first table, region second table, or region third table, the next portion of the virtual address (region second index, region third index, or segment index, respectively) is checked

against the table-offset and table-length fields in the current table entry, and it is added to the origin in the entry to select an entry in the next-lower-level table. If the selected entry in the next table is outside its table, as determined by the table-offset and table-length fields in the current table entry, or if the I bit is one in the selected entry, a region-second-translation, region-third-translation, or segment-translation exception is recognized, depending on the level of the next table. If the table-type bits in the selected entry do not indicate the expected table level, a translation-specification exception is recognized.

Processing of portions of the virtual address by means of successive table levels continues until a segment-table entry has been selected. When enhanced DAT applies, the DAT-protection bit in any and all region-table entries used during the translation are treated as being ORed with the respective bit in segment-table entry.

When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control is zero, the following conditions are in effect:

- The segment-table entry contains a DAT-protection bit that applies to all pages in the specified segment; the segment-table entry also contains a common-segment bit that controls the use of the TLB copies of the page table designated by the segment-table entry.
- The segment-table entry designates the page table to be used.
- The page-index portion of the virtual address is added to the page-table origin in the segment-table entry to select an entry in the page table. If the I bit is one in the page-table entry, a page-translation exception is recognized. The page-table entry contains the leftmost bits of the real address that represents the translation of the virtual address, and it contains a DAT-protection bit that applies only to the page specified by the page-table entry; when enhanced DAT applies and the STE-format control is zero, the page-table entry also contains a change-recording override that applies only to the page.
- The byte-index field of the virtual address is used unchanged as the rightmost bit positions of the real address.

When enhanced DAT applies and the STE-format control is one, the following conditions are in effect:

- The segment-table entry contains the leftmost bits of the absolute address that represents the translation of the virtual address.
- The page-index and byte-index fields of the virtual address are used unchanged as the rightmost bit positions of the real address.
- The segment-table entry also contains the ACCF-validity control, the access-control bits, the fetch-protection bit, the DAT-protection bit, the change-recording override, and common-segment bit that apply to the segment.

In order to eliminate the delay associated with references to translation tables in real or absolute storage, the information fetched from the tables normally is also placed in a special buffer, the translation-lookaside buffer (TLB), and subsequent translations involving the same table entries may be performed by using the information recorded in the TLB. The TLB may also record virtual-equals-real translations related to a real-space designation. The operation of the TLB is described in “Translation-Lookaside Buffer” on page 3-52.

Whenever access to real or absolute storage is made during the address-translation process for the purpose of fetching an entry from a region table, segment table, or page table, key-controlled protection does not apply.

The translation process, including the effect of the TLB, is shown graphically in Figure 3-11 on page 3-47.

Inspection of Real-Space Control

When the effective address-space-control element (ASCE) contains a real-space control, bit 58, having the value zero, the ASCE is a region-table or segment-table designation. When the real-space control is one, the ASCE is a real-space designation.

Inspection of Designation-Type Control

When the real-space control is zero, the designation-type control, bits 60 and 61 of the effective address-space-control element (ASCE), specifies the table-designation type of the ASCE. Depending on the type, some number of leftmost bits of the virtual address being translated must be zeros; otherwise, an ASCE-type exception is recognized. For each possible value of bits 60 and 61, the table-designa-

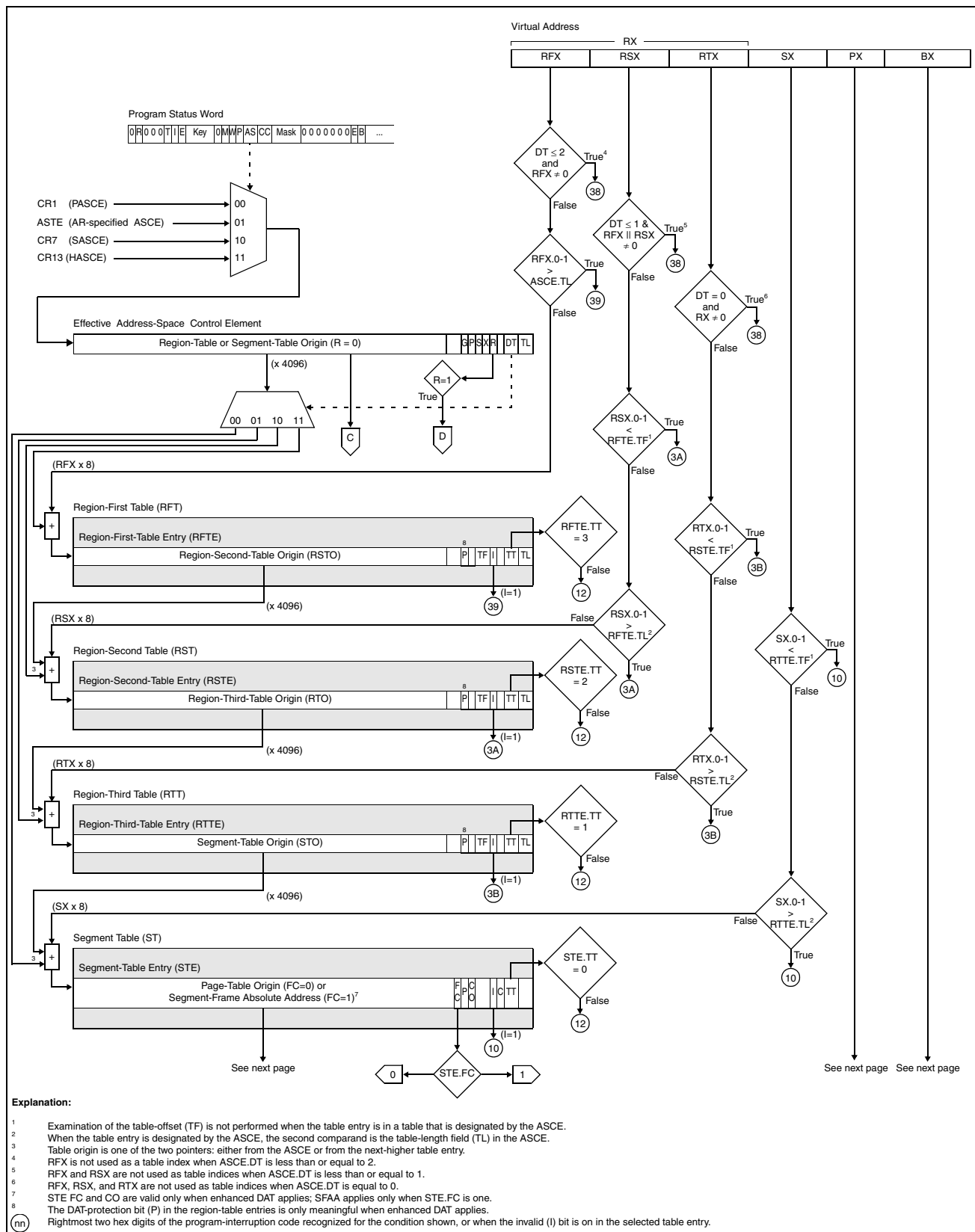


Figure 3-11. Translation Process

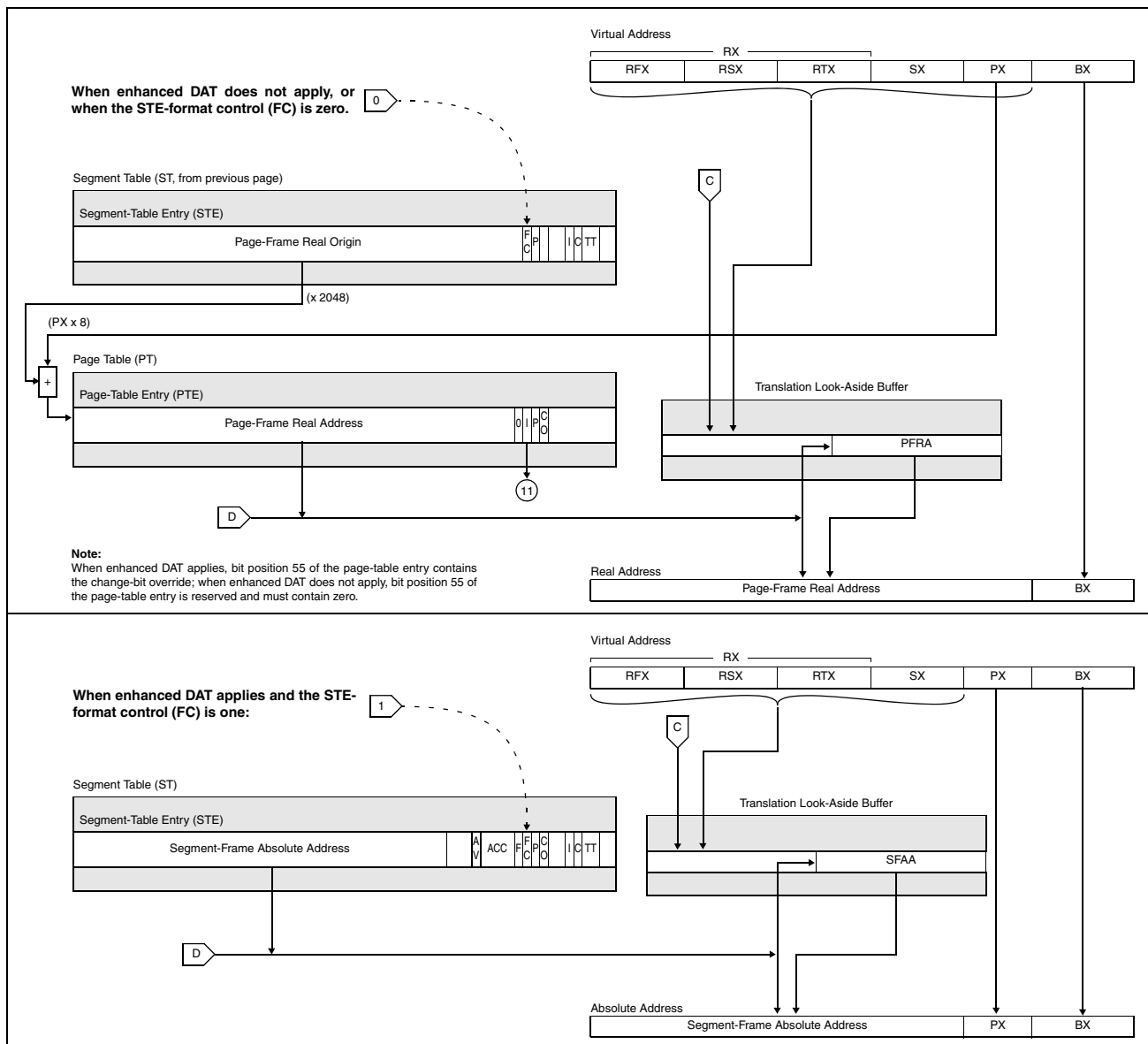


Figure 3-11. Translation Process (Continued)

tion type and the virtual-address bits required to be zeros are as follows:

Bits 60 and 61	Designation Type	Virtual-Address Bits Required to be Zeros
11	Region-first-table	None
10	Region-second-table	0-10
01	Region-third-table	0-21
00	Segment-table	0-32

Lookup in a Table Designated by an Address-Space-Control Element

The designation-type control, bits 60 and 61 of the effective address-space-control element (ASCE), specifies both the table-designation type of the ASCE and the portion of the virtual address that is to be

translated by means of the designated table, as follows:

Bits 60 and 61	Designation Type	Virtual-Address Portion Translated by the Table
11	Region-first-table	Region first index (bits 0-10)
10	Region-second-table	Region second index (bits 11-21)
01	Region-third-table	Region third index (bits 22-32)
00	Segment-table	Segment index (bits 33-43)

When bits 60 and 61 have the value 11 binary, the region-first-index portion of the virtual address, in conjunction with the region-first-table origin contained in the ASCE, is used to select an entry from the region first table.

The 64-bit address of the region-first-table entry in real or absolute storage is obtained by appending 12 zeros to the right of bits 0-51 of the region-first-table designation and adding the region first index with three rightmost and 50 leftmost zeros appended. When a carry out of bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{64} - 1$ to zero. All 64 bits of the address are used, regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode. When forming the address of a region-first-, region-second-, region-third-, or segment-table entry, it is unpredictable whether prefixing, if any, is applied to the respective table origin contained in the ASCE before the addition of the table index value, or prefixing is applied to the table-entry address that is formed by the addition of the table origin and table index value.

As part of the region-first-table-lookup process, bits 0 and 1 of the virtual address (which are bits 0 and 1 of the region first index) are compared against the table length, bits 62 and 63 of the region-first-table designation, to establish whether the addressed entry is within the region first table. If the value in the table-length field is less than the value in the corresponding bit positions of the virtual address, a region-first-translation exception is recognized. The comparison against the table length may be omitted if the equivalent of a region-first-table entry in the translation-lookaside buffer is used in the translation.

All eight bytes of the region-first-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address generated for fetching the region-first-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the unit of operation is suppressed.

Bit 58 of the entry fetched from the region first table specifies whether the corresponding set of regions is available. This bit is inspected, and, if it is one, a region-first-translation exception is recognized.

A translation-specification exception is recognized if the table-type bits, bits 60 and 61, in the region-first-table entry do not have the same value as bits 60 and 61 of the ASCE.

When no exceptions are recognized in the process of region-first-table lookup, the entry fetched from the region first table designates the beginning and specifies the offset and length of the corresponding region second table.

When bits 60 and 61 of the ASCE have the value 10 binary, the region-second-index portion of the virtual address, in conjunction with the region-second-table origin contained in the ASCE, is used to select an entry from the region second table. Bits 11 and 12 of the virtual address (which are bits 0 and 1 of the region second index) are compared against the table length in the ASCE. If the value in the table-length field is less than the value in the corresponding bit positions of the virtual address, a region-second-translation exception is recognized. The comparison against the table length may be omitted if the equivalent of a region-second-table entry in the translation-lookaside buffer is used in the translation. The region-second-table-lookup process is otherwise the same as the region-first-table-lookup process, except that a region-second-translation exception is recognized if bit 58 is one in the region-second-table entry. When no exceptions are recognized, the entry fetched from the region second table designates the beginning and specifies the offset and length of the corresponding region third table.

When bits 60 and 61 of the ASCE have the value 01 binary, the region-third-index portion of the virtual address, in conjunction with the region-third-table origin contained in the ASCE, is used to select an entry from the region third table. Bits 22 and 23 of the virtual address (which are bits 0 and 1 of the region

third index) are compared against the table length in the ASCE. If the value in the table-length field is less than the value in the corresponding bit positions of the virtual address, a region-third-translation exception is recognized. The comparison against the table length may be omitted if the equivalent of a region-third-table entry in the translation-lookaside buffer is used in the translation. The region-third-table-lookup process is otherwise the same as the region-first-table-lookup process, including the checking of the table-type bits in the region-third-table entry, except that a region-third-translation exception is recognized if bit 58 is one in the region-third-table entry. When no exceptions are recognized, the entry fetched from the region third table designates the beginning and specifies the offset and length of the corresponding segment table.

When bits 60 and 61 of the ASCE have the value 00 binary, the segment-index portion of the virtual address, in conjunction with the segment-table origin contained in the ASCE, is used to select an entry from the segment table. Bits 33 and 34 of the virtual address (which are bits 0 and 1 of the segment index) are compared against the table length in the ASCE. If the value in the table-length field is less than the value in the corresponding bit positions of the virtual address, a segment-translation exception is recognized. The comparison against the table length may be omitted if the equivalent of a segment-table entry in the translation-lookaside buffer is used in the translation. A translation-specification exception is recognized if (1) the private-space control, bit 55, in the ASCE is one and (2) the common-segment bit, bit 59, in the entry fetched from the segment table is one. The segment-table-lookup process is otherwise the same as the region-first-table-lookup process, including the checking of the table-type bits in the segment-table entry, except that a segment-translation exception is recognized if bit 58 is one in the segment-table entry. When no exceptions are recognized, processing is as follows:

- When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control is zero, the entry fetched from the segment table designates the beginning of the corresponding page table, and processing continues as described in “Page-Table Lookup”, below.
- When enhanced DAT applies and the STE-format control is one, the entry fetched from the segment table contains the leftmost bits of the segment-frame absolute address. If the DAT-pro-

tection bit, bit 54, is one either in any region-table entry used in the translation or in the segment-table entry, and the storage reference for which the translation is being performed is a store, a protection exception is recognized.

Lookup in a Table Designated by a Region-Table Entry

When the effective address-space-control element (ASCE) is a region-table designation, a region-table entry is selected as described in the preceding section. Then the contents of the selected entry and the next index portion of the virtual address are used to select an entry in the next-lower-level table, which may be another region table or a segment table.

When the table entry selected by means of the ASCE is a region-first-table entry, the region-second-index portion of the virtual address, in conjunction with the region-second-table origin contained in the region-first-table entry, is used to select an entry from the region second table.

The 64-bit address of the region-second-table entry in real or absolute storage is obtained by appending 12 zeros to the right of bits 0-51 of the region-first-table entry and adding the region second index with three rightmost and 50 leftmost zeros appended. When a carry out of bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{64} - 1$ to zero. All 64 bits of the address are used, regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode. When forming the address of a region-second-, region-third-, or segment-table entry, it is unpredictable whether prefixing, if any, is applied to the respective table origin contained in the higher-level table entry before the addition of the table index value, or prefixing is applied to the table-entry address that is formed by the addition of the table origin and table index value.

As part of the region-second-table-lookup process, bits 11 and 12 of the virtual address (which are bits 0 and 1 of the region second index) are compared against the table offset, bits 56 and 57 of the region-first-table entry, and against the table length, bits 62 and 63 of the region-first-table entry, to establish whether the addressed entry is within the region second table. If the value in the table-offset field is greater than the value in the corresponding bit positions of the virtual address, or if the value in the table-

length field is less than the value in the corresponding bit positions of the virtual address, a region-second-translation exception is recognized.

All eight bytes of the region-second-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address generated for fetching the region-second-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the unit of operation is suppressed.

Bit 58 of the entry fetched from the region second table specifies whether the corresponding set of regions is available. This bit is inspected, and, if it is one, a region-second-translation exception is recognized.

A translation-specification exception is recognized if the table-type bits, bits 60 and 61, in the region-second-table entry do not have a value that is one less than the value of those bits in the next-higher-level table.

When no exceptions are recognized in the process of region-second-table lookup, the entry fetched from the region second table designates the beginning and specifies the offset and length of the corresponding region third table.

When the table entry selected by means of the ASCE is a region-second-table entry, or if a region-second-table entry has been selected by means of the contents of a region-first-table entry, the region-third-index portion of the virtual address, in conjunction with the region-third-table origin contained in the region-second-table entry, is used to select an entry from the region third table. Bits 22 and 23 of the virtual address (which are bits 0 and 1 of the region third index) are compared against the table offset and table length in the region-second-table entry. A region-third-translation exception is recognized if the table offset is greater than bits 22 and 23 or if the table length is less than bits 22 and 23. The region-third-table-lookup process is otherwise the same as the region-second-table-lookup process, including the checking of the table-type bits in the region-third-table entry, except that a region-third-translation exception is recognized if bit 58 is one in the region-third-table entry. When no exceptions are recognized, the entry fetched from the region third table designates the beginning and specifies the offset and length of the corresponding segment table.

When the table entry selected by means of the ASCE is a region-third-table entry, or if a region-third-table entry has been selected by means of the contents of a region-second-table entry, the segment-index portion of the virtual address, in conjunction with the segment-table origin contained in the region-third-table entry, is used to select an entry from the segment table. Bits 33 and 34 of the virtual address (which are bits 0 and 1 of the segment index) are compared against the table offset and table length in the region-third-table entry. A segment-translation exception is recognized if the table offset is greater than bits 33 and 34 or if the table length is less than bits 33 and 34. A translation-specification exception is recognized if (1) the private-space control, bit 55, in the ASCE is one and (2) the common-segment bit, bit 59, in the entry fetched from the segment table is one. The segment-table-lookup process is otherwise the same as the region-second-table-lookup process, including the checking of the table-type bits in the segment-table entry, except that a segment-translation exception is recognized if bit 58 is one in the segment-table entry. When no exceptions are recognized, processing is as follows:

- When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control is zero, the entry fetched from the segment table designates the beginning of the corresponding page table, and processing continues as described in “Page-Table Lookup”, below.
- When enhanced DAT applies and the STE-format control is one, the entry fetched from the segment table contains the leftmost bits of the segment-frame absolute address. If the DAT-protection bit, bit 54, is one either in any region-table entry used in the translation or in the segment-table entry, and the storage reference for which the translation is being performed is a store, a protection exception is recognized.

Page-Table Lookup

When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control is zero, the page-index portion of the virtual address, in conjunction with the page-table origin contained in the segment-table entry, is used to select an entry from the page table.

The 64-bit address of the page-table entry in real or absolute storage is obtained by appending 11 zeros to the right of the page-table origin and adding the page index, with three rightmost and 53 leftmost

zeros appended. A carry out of bit position 0 cannot occur. All 64 bits of the address are used, regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

All eight bytes of the page-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address generated for fetching the page-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the unit of operation is suppressed.

The entry fetched from the page table indicates the availability of the page and contains the leftmost bits of the page-frame real address. The page-invalid bit, bit 53, is inspected to establish whether the corresponding page is available. If this bit is one, a page-translation exception is recognized. If bit position 52 contains a one, a translation-specification exception is recognized. When enhanced DAT does not apply, or enhanced DAT applies and the STE-format control is zero, a translation-specification exception is also recognized if bit position 55 contains a one. If the DAT-protection bit, bit 54, is one either in the segment-table entry used in the translation, in the page-table entry, or, when enhanced DAT applies, in any region-table entry used during the translation, and the storage reference for which the translation is being performed is a store, a protection exception is recognized.

Formation of the Real and Absolute Addresses

When the effective address-space-control element (ASCE) is a real-space designation, bits 0-63 of the virtual address are used directly as the real storage address. The real address is then subjected to prefixing to form an absolute address.

When the effective ASCE is not a real-space designation and no exceptions in the translation process are encountered, the following conditions apply:

- When the enhanced DAT does not apply, or when enhanced DAT applies but the STE format control is zero, the page-frame real address is obtained from the page-table entry. The page-frame real address and the byte-index portion of the virtual address are concatenated, with the page-frame real address forming the leftmost part. The result is the real storage address which

corresponds to the virtual address. The real address is then subjected to prefixing to form an absolute address.

- When enhanced DAT applies and the STE format control is one, the segment-frame absolute address and the page-index and byte-index portions of the virtual address are concatenated, left to right, respectively, to form the absolute address which corresponds to the virtual address.

All 64 bits of the real and absolute addresses are used, regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

Recognition of Exceptions during Translation

Invalid addresses and invalid formats can cause exceptions to be recognized during the translation process. Exceptions are recognized when information contained in table entries is used for translation and is found to be incorrect.

The information pertaining to DAT is considered to be used when an instruction is executed with DAT on or when INVALIDATE DAT TABLE ENTRY, INVALIDATE PAGE TABLE ENTRY, LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, or STORE REAL ADDRESS is executed. The information is not considered to be used when the PSW specifies DAT on but an I/O, external, restart, or machine-check interruption occurs before an instruction is executed, or when the PSW specifies the wait state. Only that information required in order to translate a virtual address is considered to be in use during the translation of that address, and, in particular, addressing exceptions that would be caused by the use of an address-space-control element are not recognized when that address-space-control element is not the one actually used in the translation.

A list of translation exceptions, with the action taken for each exception and the priority in which the exceptions are recognized when more than one is applicable, is provided in "Recognition of Access Exceptions" on page 6-40.

Translation-Lookaside Buffer

To enhance performance, the dynamic-address-translation mechanism normally is implemented such that some of the information specified in the region

tables, segment tables, and page tables is maintained in a special buffer, referred to as the translation-lookaside buffer (TLB). The CPU necessarily refers to a DAT-table entry in real or absolute storage only for the initial access to that entry. This information may be placed in the TLB, and subsequent translations may be performed by using the information in the TLB. For consistency of operation, the virtual-equals-real translation specified by a real-space designation also may be performed by using information in the TLB. The presence of the TLB affects the translation process to the extent that (1) a modification of the contents of a table entry in real or absolute storage does not necessarily have an immediate effect, if any, on the translation, (2) a region-first-table origin, region-second-table origin, region-third-table origin, segment-table origin, or real-space token origin in an address-space-control element (ASCE) may select a TLB entry that was formed by means of an ASCE containing an origin of the same value even when the two origins are of different types, and (3) the comparison against the table length in an address-space-control element may be omitted if a TLB equivalent of the designated table entry is used. In a multiple-CPU configuration, each CPU has its own TLB.

Entries within the TLB are not explicitly addressable by the program.

Information is not necessarily retained in the TLB under all conditions for which such retention is permissible. Furthermore, information in the TLB may be cleared under conditions additional to those for which clearing is mandatory.

TLB Structure

The description of the logical structure of the TLB covers the implementation by all systems operating as defined by z/Architecture. The TLB entries are considered as being of three types: TLB combined region-and-segment-table entries, TLB page-table entries, and TLB real-space entries. A TLB combined region-and-segment-table entry or TLB page-table entry is considered as containing within it both the information obtained from the table entry or entries in real or absolute storage and the attributes used to fetch this information from storage. A TLB real-space entry is considered as containing a page-frame real address and the real-space token origin and region, segment, and page indexes used to form the entry.

The token origin in a TLB real-space entry is indistinguishable from the table origin in a TLB combined region-and-segment-table entry.

Note: The following sections describe the conditions under which information may be placed in the TLB, the conditions under which information from the TLB may be used for address translation, and how changes to the translation tables affect the translation process.

Formation of TLB Entries

The formation of TLB combined region-and-segment-table entries and TLB page-table entries from table entries in real or absolute storage, and the effect of any manipulation of the contents of table entries in storage by the program, depend on whether the entries in storage are attached to a particular CPU and on whether the entries are valid.

The *attached* state of a table entry denotes that the CPU to which it is attached can attempt to use the table entry for implicit address translation, except that a table entry for the primary or home address space may be attached even when the CPU does not fetch from either of those spaces. A table entry may be attached to more than one CPU at a time.

The *valid* state of a table entry denotes that the region set, region, segment, or page associated with the table entry is available. An entry is valid when the region-invalid, segment-invalid, or page-invalid bit in the entry is zero.

The region-table entries, if any, and the segment-table entry used to form a TLB combined region-and-segment-table entry are called a *translation path*. A translation path may be placed in the TLB as a combined region-and-segment-table entry whenever all entries in the path are attached and valid and would not cause a translation-specification exception if used for translation. Similarly, a page-table entry may be placed in the TLB whenever the entry is attached and valid and would not cause a translation-specification exception if used for translation.

The highest-level table entry in a translation path is attached when it is within a table designated by an attaching address-space-control element (ASCE). “Within a table” means as determined by the origin and length fields in the ASCE. An ASCE is an attaching ASCE when all of the following conditions are met:

1. The current PSW specifies DAT on.
2. The current PSW contains no errors that would cause an early specification exception to be recognized.
3. The ASCE meets the requirements in a, b, c, d, or e below.
 - a. The ASCE is the primary ASCE in control register 1.
 - b. The ASCE is the secondary ASCE in control register 7, and either of the following requirements is met:
 - The CPU is in the secondary-space mode or access-register mode.
 - The CPU is in the primary-space mode, and the secondary-space control, bit 37 of control register 0, is one.
 - c. The ASCE is in either an attached and valid ASN-second-table entry (ASTE) or a usable ALB ASTE, and the CPU is in the access-register mode. See “ART-Lookaside Buffer” on page 5-60 for the meaning of the terminology used here.
 - d. The ASCE is the home ASCE in control register 13.
 - e. For the R_2 operand of the LOAD PAGE-TABLE-ENTRY ADDRESS instruction, the ASCE is that specified by the M_4 field.

When the enhanced-monitor facility is installed, the home ASCE is also considered to be an attaching ASCE when a monitor-event counting operation occurs, regardless of whether DAT is on.

Each of the remaining table entries in a translation path is attached when the next-higher-level entry is attached and valid and would not cause a translation-specification exception if used for translation and the subject entry is within the table designated by the next-higher-level entry. “Within the table” means as determined by the origin, offset, and length fields in the next-higher-level entry.

A page-table entry is attached when it is within the page table designated by either an attached and valid segment-table entry that would not cause a translation-specification exception if used for translation or a usable TLB combined region-and-seg-

ment-table entry. A usable TLB combined region-and-segment-table entry is explained in the next section.

A region-table entry or segment-table entry causes a translation-specification exception if the table-type bits, bits 60 and 61, in the entry are inconsistent with the level at which the entry would be encountered when using the translation path in the translation process. A segment-table entry also causes a translation-specification exception if the private-space-control bit is one in the address-space-control element used to select it and the common-segment bit is one in the entry. A page-table entry causes a translation-specification exception if bit 52 in the entry is one. When enhanced DAT does not apply, a page-table entry also causes a translation-specification exception if bit 55 in the entry is one.

A TLB real-space entry may be formed whenever an attaching real-space designation exists. The entry is formed using the real-space token origin in the designation and any value of bits 0-51 of a virtual address.

Use of TLB Entries

The *usable* state of a TLB entry denotes that the CPU can attempt to use the TLB entry for implicit address translation. A usable TLB entry attaches the next-lower-level table, if any, and may be usable for a particular instance of implicit address translation.

A TLB combined region-and-segment-table entry is in the usable state when all of the following conditions are met:

1. The current PSW specifies DAT on.
2. The current PSW contains no errors that would cause an early specification exception to be recognized.
3. The TLB combined region-and-segment-table entry meets either of the following requirements:
 - a. The common-segment bit is one in the TLB entry.
 - b. The table-origin (TO) field in the TLB entry matches the table- or token-origin field in an attaching address-space-control element.

A TLB combined region-and-segment-table entry may be used for a particular instance of implicit address translation only when the entry is in the usable state, either the common-segment bit is one in the TLB entry or the table-origin (TO) field in the

TLB entry matches the table- or token-origin field in the address-space-control element being used in the translation, and the region-index and segment-index fields in the TLB entry match those of the virtual address being translated. However, the TLB combined region-and-segment-table entry is not used if the common-segment bit is one in the entry and either the private-space-control bit is one in the address-space-control element being used in the translation or that address-space-control element is a real-space designation. In both these cases, the TLB entry is not used even if the table-origin field in the entry and the table- or token-origin field in the address-space-control element match.

A TLB page-table entry may be used for a particular instance of implicit address translation only when the page-table-origin field in the entry matches the page-table-origin field in the segment-table entry or TLB combined region-and-segment-table entry being used in the translation and the page-index field in the TLB page-table entry matches the page index of the virtual address being translated.

A TLB real-space entry may be used for implicit address translation only when the token-origin field in the TLB entry matches the table- or token-origin field in the address-space-control element being used in the translation and the region-index, segment-index, and page-index fields in the TLB entry match those of the virtual address being translated

The operand address of LOAD PAGE-TABLE-ENTRY ADDRESS and LOAD REAL ADDRESS may be translated with the use of the TLB contents whether DAT is on or off, but TLB entries still are formed only if DAT is on.

Programming Notes:

1. Although contents of a table entry may be copied into the TLB only when the table entry is both attached and valid, the copy may remain in the TLB even when the table entry itself is no longer attached or valid.
2. No contents can be copied into the TLB when DAT is off because the table entries at this time are not attached. In particular, translation of the operand address of LOAD PAGE-TABLE-ENTRY ADDRESS and LOAD REAL ADDRESS with DAT off does not cause entries to be placed in the TLB.

Conversely, when DAT is on, information may be copied into the TLB from all translation-table entries that could be used for address translation, given the current translation parameters, the setting of the address-space-control bits, and the contents of the access registers. The loading of the TLB does not depend on whether the entry is used for translation as part of the execution of the current instruction, and such loading can occur when the CPU is in the wait state.

3. More than one copy of contents of a table entry may exist in the TLB. For example, some implementations may cause a copy of contents of a valid table entry to be placed in the TLB for the table origin in each address-space-control element by which the entry becomes attached.

Modification of Translation Tables

When an attached and invalid table entry is made valid and no entry usable for translation of the associated virtual address is in the TLB, the change takes effect no later than the end of the current unit of operation. Similarly, when an unattached and valid table entry is made attached and no usable entry for the associated virtual address is in the TLB, the change takes effect no later than the end of the current unit of operation.

When a valid and attached table entry is changed, and when, before the TLB is cleared of entries that qualify for substitution for that entry, an attempt is made to refer to storage by using a virtual address requiring that entry for translation, unpredictable results may occur, to the following extent. The use of the new value may begin between instructions or during the execution of an instruction, including the instruction that caused the change. Moreover, until the TLB is cleared of entries that qualify for substitution for that entry, the TLB may contain both the old and the new values, and it is unpredictable whether the old or new value is selected for a particular access. If both old and new values of a translation path are present in the TLB, a page-table entry may be fetched by using one value and placed in the TLB associated with the other value. If the new value of the path is a value that would cause an exception, the exception may or may not cause an interruption to occur. If an interruption does occur, the result fields of the instruction may be changed even though the exception would normally cause suppression or nullification.

Entries are cleared from the TLB in accordance with the following rules:

1. All entries are cleared from the TLB by the execution of PURGE TLB or SET PREFIX and by CPU reset.
2. All entries may be cleared from all TLBs in the configuration by the execution of COMPARE AND SWAP AND PURGE by any of the CPUs in the configuration, depending on a bit in a general register used by the instruction.
3. Selected entries are cleared from all TLBs in the configuration by the execution of INVALIDATE DAT TABLE ENTRY or INVALIDATE PAGE TABLE ENTRY by any of the CPUs in the configuration.
4. Some or all TLB entries may be cleared at times other than those required by the preceding rules.

Programming Notes:

1. Entries in the TLB may continue to be used for translation after the table entries from which they have been formed have become unattached or invalid. These TLB entries are not necessarily removed unless explicitly cleared from the TLB.

A change made to an attached and valid entry or a change made to a table entry that causes the entry to become attached and valid is reflected in the translation process for the next instruction, or earlier than the next instruction, unless a TLB entry qualifies for substitution for that table entry. However, a change made to a table entry that causes the entry to become unattached or invalid is not necessarily reflected in the translation process until the TLB is cleared of entries that qualify for substitution for that table entry.

2. Exceptions associated with dynamic address translation may be established by a pretest for operand accessibility that is performed as part of the initiation of instruction execution. Consequently, a region-first-translation, region-second-translation, region-third-translation, segment-translation, or page-translation exception may be indicated when a table entry is invalid at the start of execution even if the instruction would have validated the table entry it uses and the table entry would have appeared valid if the instruction was considered to process the operands one byte at a time.

3. A change made to an attached table entry, except to set the I bit to zero, to set the CO bit of a segment- or page-table entry to one, or to alter the rightmost byte of a page-table entry, may produce unpredictable results if that entry is used for translation before the TLB is cleared of all copies of contents of that entry. The use of the new value may begin between instructions or during the execution of an instruction, including the instruction that caused the change. When an instruction, such as MOVE (MVC), makes a change to an attached table entry, including a change that makes the entry invalid, and subsequently uses the entry for translation, a changed entry is being used without a prior clearing of the entry from the TLB, and the associated unpredictability of result values and of exception recognition applies.

Manipulation of attached table entries may cause spurious table-entry values to be recorded in a TLB. For example, if changes are made piecemeal, modification of a valid attached entry may cause a partially updated entry to be recorded, or, if an intermediate value is introduced in the process of the change, a supposedly invalid entry may temporarily appear valid and may be recorded in the TLB. Such an intermediate value may be introduced if the change is made by an I/O operation that is retried, or if an intermediate value is introduced during the execution of a single instruction.

As another example, if a segment-table entry is changed to designate a different page table and used without clearing the TLB, the new page-table entries may be fetched and associated with the old page-table origin. In such a case, execution of INVALIDATE PAGE TABLE ENTRY designating the new page-table origin will not necessarily clear the page-table entries fetched from the new page table.

4. To facilitate the manipulation of page tables, the INVALIDATE PAGE TABLE ENTRY instruction is provided. This instruction sets the I bit in one or more page-table entries to one and clears all TLBs in the configuration of entries formed from those table entries.

INVALIDATE PAGE TABLE ENTRY is useful for setting the I bit to one in one or more page-table entries and causing TLB copies of the entries to be cleared from the TLB of each CPU in the configuration. The following aspects of the TLB oper-

ation should be considered when using INVALIDATE PAGE TABLE ENTRY. (See also the programming notes for INVALIDATE PAGE TABLE ENTRY.)

- a. INVALIDATE PAGE TABLE ENTRY should be executed before making any change to a page-table entry other than changing the rightmost byte; otherwise, the selective-clearing portion of INVALIDATE PAGE TABLE ENTRY may not clear the TLB copies of the entry.
 - b. Invalidation of all the page-table entries within a page table by means of INVALIDATE PAGE TABLE ENTRY does not necessarily clear the TLB of any combined region-and-segment-table entry designating the page table. When it is desired to invalidate and clear the TLB of a combined region-and-segment-table entry, the rules in note 5 below must be followed.
 - c. When a large number of page-table entries are to be invalidated at a single time, the overhead involved in using COMPARE AND SWAP AND PURGE (one that purges the TLB), INVALIDATE DAT TABLE ENTRY, or PURGE TLB and in following the rules in note 5 below may be less than in issuing INVALIDATE PAGE TABLE ENTRY once for each individual page-table entry. When the IPTE-range facility is installed, improved performance may be realized by invalidating multiple page-table entries with a single execution of INVALIDATE PAGE TABLE ENTRY.
5. Manipulation of table entries should be in accordance with the following rules. If these rules are complied with, translation is performed as if the table entries from real or absolute storage were always used in the translation process.
- a. A valid table entry must not be changed while it is attached to any CPU and may be used for translation by that CPU except to (1) invalidate the entry by using INVALIDATE PAGE TABLE ENTRY or INVALIDATE DAT TABLE ENTRY, (2) alter bits 56-63 of a page-table entry, or (3) make a change by means of a COMPARE AND SWAP AND PURGE instruction that purges the TLB.
 - b. When any change is made to an attached and valid or unattached or invalid table entry

other than a change to bits 56-63 of a page-table entry, each CPU which may have a TLB entry formed from that entry must be caused to purge its TLB after the change occurs and prior to the use of that entry for implicit translation by that CPU. (Note that a separate purge is unnecessary if the change was made by using INVALIDATE DAT TABLE ENTRY, INVALIDATE PAGE TABLE ENTRY, or a COMPARE AND SWAP AND PURGE instruction that purges the TLB.) In the case when the table entry is attached and valid, this rule applies when it is known that a program is not being executed that may require the entry for translation.

- c. When any change is made to an invalid table entry in such a way as to allow intermediate valid values to appear in the entry, each CPU to which the entry is attached must be caused to purge its TLB after the change occurs and prior to the use of the entry for implicit address translation by that CPU.
- d. When a change is made to an offset or length specified for a table, each CPU which may have a TLB entry formed from a table entry that no longer lies within its table must be caused to purge its TLB after the change occurs and prior to the use of the table for implicit translation by that CPU.

Note that when an invalid page-table entry is made valid without introducing intermediate valid values, the TLB need not be cleared in a CPU which does not have any TLB entries formed from that entry. Similarly, when an invalid region-table or segment-table entry is made valid without introducing intermediate valid values, the TLB need not be cleared in a CPU which does not have any TLB entries formed from that validated entry and which does not have any TLB entries formed from entries in a page table attached by means of that validated entry.

The execution of PURGE TLB, COMPARE AND SWAP AND PURGE, or SET PREFIX may have an adverse effect on the performance of some models. Use of these instructions should, therefore, be minimized in conformance with the above rules.

6. In addition to the constraints described in programming note 3 on page 3-56, the following

considerations are in effect when enhanced DAT applies:

- a. When the STE-format and ACCF-validity controls are both one, it is unpredictable whether the CPU inspects the access-control bits and the fetch-protection bit in the segment-table entry or in the storage key of the corresponding 4K-byte block for any given key-controlled-protection check. Therefore, the program should ensure that the access-control bits and fetch-protection bit in the segment-table entry are identical to the respective fields in all 256 storage keys for the constituent 4K-byte blocks of the segment, before setting the invalid bit in the STE to zero.

Prior to changing the ACCF-validity control, the access-control bits, or the fetch protection bit in the segment-table entry, and prior to changing the access-control bits or fetch-protection bit in any of the segment's 256 storage keys, the program should first set the invalid bit to one in the segment-table entry and clear all entries in all TLBs in the configuration, as described previously in this section.

- b. Prior to setting the change-recording override to zero in a segment-table entry, the program should set the invalid bit to one in the segment-table entry, and clear all entries in all TLBs in the configuration (for example, by using INVALIDATE DAT TABLE ENTRY).
- c. Prior to setting the change-recording override to zero in page-table entry, the program should set the invalid bit to one in the page-table entry, and clear all entries in all TLBs in the configuration (for example, by using INVALIDATE PAGE TABLE ENTRY).

Failure to observe these procedures may lead to unpredictable results, possibly including a delayed-access-exception machine-check or failure to record a change.

Address Summary

Addresses Translated

Most addresses that are explicitly specified by the program and are used by the CPU to refer to storage are instruction or logical addresses and are subject to implicit translation when DAT is on. Analogously, the corresponding addresses indicated to the program on an interruption or as the result of executing an instruction are instruction or logical addresses. The operand address of LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, and STORE REAL ADDRESS is explicitly translated, regardless of whether the PSW specifies DAT on or off.

Translation is not applied to quantities that are formed from the values specified in the B and D fields of an instruction but that are not used to address storage. This includes operand addresses in EXTRACT CACHE ATTRIBUTE, LOAD ADDRESS, LOAD ADDRESS EXTENDED, MONITOR CALL, the second operand address in LOAD ADDRESS SPACE PARAMETERS and SHIFT AND ROUND PACKED, and the shifting instructions. This also includes the addresses in control registers 10 and 11 designating the starting and ending locations for PER.

Except as noted below, the addresses explicitly designating storage keys (operand addresses in INSERT STORAGE KEY EXTENDED, PERFORM FRAME MANAGEMENT FUNCTION, RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED) are real addresses.

- For INSERT VIRTUAL STORAGE KEY, the address designating the storage key is virtual.
- For PERFORM FRAME MANAGEMENT FUNCTION when the set-key control and frame-size code are both 1, for RESET REFERENCE BITS MULTIPLE, and for SET STORAGE KEY EXTENDED when the multiple-block control is one, the address designating the storage keys is absolute.
- For TEST PROTECTION, the address designating the block to be tested is logical.

The addresses implicitly used by the CPU for such sequences as interruptions are real addresses.

The addresses used by channel programs to transfer data and to refer to CCWs, IDAWs, or MIDAWs are absolute addresses.

The handling of storage addresses associated with DIAGNOSE is model-dependent.

The processing of addresses, including dynamic address translation and prefixing, is discussed in “Address Types” on page 3-4. Prefixing, when provided, is applied after the address has been trans-

lated by means of the dynamic-address-translation facility. For a description of prefixing, see “Prefixing” on page 3-18.

Handling of Addresses

The handling of addresses is summarized in Figure 3-12. This figure lists all addresses that are encountered by the program and specifies the address type.

Virtual Addresses

- Address of storage operand for INSERT VIRTUAL STORAGE KEY
- Operand address in LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, and STORE REAL ADDRESS
- Addresses of storage operands for MOVE TO PRIMARY and MOVE TO SECONDARY
- Address stored in the doubleword at real location 168 on a program interruption for ASCE-type, region-first-translation, region-second-translation, region-third-translation, segment-translation, or page-translation exception
- Linkage-stack-entry address in control register 15
- Backward stack-entry address in linkage-stack header entry
- Forward-section-header address in linkage-stack trailer entry
- Trap-control-block address in dispatchable-unit-control table
- Trap-save-area address and trap-program address in trap control block
- Address of the enhanced-monitor counter array in the doubleword at real location 256

Instruction Addresses

- Instruction address in PSW
- Branch address
- Target of EXECUTE and EXECUTE RELATIVE LONG
- Address stored in the doubleword at real location 152 on a program interruption for PER
- Address placed in general register by BRANCH AND LINK, BRANCH AND SAVE, BRANCH AND SAVE AND SET MODE, BRANCH AND STACK, BRANCH IN SUBSPACE GROUP, BRANCH RELATIVE AND SAVE, BRANCH RELATIVE AND SAVE LONG, and PROGRAM CALL
- Address used in general register by BRANCH AND STACK
- Address placed in general register by BRANCH AND SET AUTHORITY executed in reduced-authority state

Logical Addresses

- Addresses of storage operands for instructions not otherwise specified
- Address placed in general register 1 by EDIT AND MARK and TRANSLATE AND TEST
- Addresses in general registers updated by MOVE LONG, MOVE LONG EXTENDED, COMPARE LOGICAL LONG, and COMPARE LOGICAL LONG EXTENDED
- Addresses in general registers updated by CHECKSUM, COMPARE AND FORM CODEWORD, and UPDATE TREE
- Address for TEST PENDING INTERRUPTION when the second-operand address is nonzero
- Address of parameter list of RESUME PROGRAM

Figure 3-12. Handling of Addresses (Part 1 of 3).

Real Addresses

- Address of storage key for INSERT STORAGE KEY EXTENDED, and RESET REFERENCE BIT EXTENDED
- Address of storage key for SET STORAGE KEY EXTENDED (when the enhanced-DAT facility is not installed, or when the enhanced-DAT facility is installed but the multiple-block control is zero)
- Address of second operand for PERFORM FRAME MANAGEMENT FUNCTION when frame-size code is 0.
- Address of storage operand for LOAD USING REAL ADDRESS, STORE USING REAL ADDRESS, and TEST BLOCK
- The translated address generated by LOAD REAL ADDRESS and STORE REAL ADDRESS
- Page-frame real address in page-table entry
- Trace-entry address in control register 12
- ASN-first-table origin in control register 14
- ASN-second-table origin in ASN-first-table entry
- Authority-table origin in ASN-second-table entry, except when used by access-register translation
- Linkage-table origin in primary ASN-second-table entry
- Entry-table origin in linkage-table entry
- Dispatchable-unit-control-table origin in control register 2
- Primary-ASN-second-table-entry origin in control register 5
- Base-ASN-second-table-entry origin and subspace-ASN-second-table-entry origin in dispatchable-unit control table
- ASN-second-table-entry address in entry-table entry and access-list entry

Permanently Assigned Real Addresses

- Address of the doubleword into which TEST PENDING INTERRUPTION stores when the second-operand address is zero
- Addresses of PSWs, interruption codes, and the associated information used during interruption
- Addresses used for machine-check logout and save areas
- Address of STORE FACILITY LIST operand

Addresses which Are Unpredictably Real or Absolute

- Region-first-table origin, region-second-table origin, region-third-table origin, or segment-table origin in control registers 1, 7, and 13, in access-register-specified address-space-control element, and in region-first-table entry, region-second-table entry, or region-third-table entry
- Page-table origin in segment-table entry and in INVALIDATE PAGE TABLE ENTRY
- Address of segment-table entry or page-table entry provided by LOAD REAL ADDRESS
- Address of region-first-table entry, region-second-table entry, region-third-table entry, segment-table entry, or page-table entry provided by LOAD PAGE-TABLE-ENTRY ADDRESS
- The dispatchable-unit or primary-space access-list origin and the authority-table origin (in the ASTE designated by the ALE used) used by access-register translation

Figure 3-12. Handling of Addresses (Part 2 of 3).

Absolute Addresses

- Prefix value
- Channel-program address in ORB
- Data address in CCW
- IDAW address in a CCW specifying indirect data addressing
- MIDAW address in a CCW specifying modified indirect data addressing
- CCW address in a CCW specifying transfer in channel
- Data address in IDAW
- Data address in MIDAW
- Measurement-block origin specified by SET CHANNEL MONITOR
- Address limit specified by SET ADDRESS LIMIT
- Addresses used by the store-status-at-address SIGNAL PROCESSOR order
- Address of storage key for RESET REFERENCE BITS MULTIPLE
- Address of storage key for SET STORAGE KEY EXTENDED (when the enhanced-DAT facility is installed and the multiple-block control is one)
- Address of second operand for PERFORM FRAME MANAGEMENT FUNCTION when frame-size code is 1.
- Failing-storage address stored in the doubleword at real location 248
- CCW address in SCSW

Permanently Assigned Absolute Addresses

- Addresses used for the store-status function
- Addresses of PSW and first two CCWs used for initial program loading

Addresses Not Used to Reference Storage

- PER starting address in control register 10
- PER ending address in control register 11
- Address stored in the doubleword at real location 176 for a monitor event
- Address in shift instructions and other instructions specified not to use the address to reference storage
- Real-space token origin in real-space designation

Figure 3-12. Handling of Addresses (Part 3 of 3).

Assigned Storage Locations

Figure 3-13 on page 3-68 shows the format and extent of the assigned locations in storage. The locations are used as follows.

128-131 (80-83 hex) Real Address

External-Interruption Parameter: During an external interruption due to service signal or timing alert, the parameter associated with the interruption is stored at locations 128-131.

132-133 (84-85 hex) Real Address

CPU Address: During an external interruption due to malfunction alert, emergency signal, or external call, the CPU address associated with

the source of the interruption is stored at locations 132-133. For all other external-interruption conditions, zeros are stored at locations 132-133.

134-135 (86-87 hex) Real Address

External-Interruption Code: During an external interruption, the interruption code is stored at locations 134-135.

136-139 (88-8B hex) Real Address

Supervisor-Call-Interruption Identification: During a supervisor-call interruption, the instruction-length code is stored in bit positions 5 and 6 of location 137, and the interruption code is stored at locations 138-139. Zeros are stored at location 136 and in the remaining bit positions of location 137.

140-143 (8C-8F hex) Real Address

Program-Interrupt Identification: During a program interruption, the instruction-length code is stored in bit positions 5 and 6 of location 141, and the interruption code is stored at locations 142-143. Zeros are stored at location 140 and in the remaining bit positions of location 141.

144-147 (90-93 hex). Real Address

Data-Exception Code (DXC): During a program interruption due to a data exception, the data-exception code is stored at location 147, and zeros are stored at locations 144-146. The DXC is described in "Data-Exception Code (DXC)" on page 6-14.

148-149 (94-95 hex). Real Address

Monitor-Class Number: During a program interruption due to a monitor event, the monitor-class number is stored at location 149, and zeros are stored at location 148.

150-151 (96-97 hex). Real Address

PER Code: During a program interruption due to a PER event the PER code is stored in bit positions 0-7 of locations 150-151, and other information is or may be stored as described in "Identification of Cause" on page 4-26.

152-159 (98-9F hex) Real Address

PER Address: During a program interruption due to a PER event, the PER address is stored at locations 152-159.

160 (A0 hex). Real Address

Exception Access Identification: During a program interruption due to an ASCE-type, region-first-translation, region-second-translation, region-third-translation, segment-translation, or page-translation exception, an indication of the address space to which the exception applies may be stored at location 160. If the CPU was in the access-register mode and either (a) the access was an instruction fetch, including a fetch of the target of an execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG), or (b) the access was to the second operand of the relative-long form of COMPARE, COMPARE HALFWORD, COMPARE LOGICAL, COMPARE LOGICAL HALFWORD, LOAD, LOAD HALFWORD, LOAD LOGICAL, LOAD LOGICAL

HALFWORD, STORE or STORE HALFWORD, then zeros are stored at location 160. If the access was a storage-operand reference that used an AR-specified address-space-control element, the number of the access register used is stored in bit positions 4-7 of location 160, and zeros are stored in bit positions 0-3. (In either of the two cases described so far, storing at location 160 occurs regardless of the value stored in bit positions 62 and 63 of real locations 168-175.) The contents of location 160 are unpredictable if the access did not use an AR-specified address-space-control element.

During a program interruption due to an ALEN-translation, ALE-sequence, ASTE-validity, ASTE-sequence, or extended-authority exception recognized during access-register translation, the number of the access register used is stored in bit positions 4-7 of location 160, and zeros are stored in bit positions 0-3. During a program interruption due to an ASTE-validity or ASTE-sequence exception recognized during a sub-space-replacement operation, all zeros are stored at location 160.

During a program interruption due to an ASTE-instance exception recognized due to use of the ASN-and-LX-reuse facility, (1) a one is stored in bit position 2, and zeros are stored in bit positions 0, 1, and 3-7, if the exception was recognized after primary ASN translation in PROGRAM TRANSFER WITH INSTANCE or PROGRAM RETURN, or (2) a one is stored in bit position 3, and zeros are stored in bit positions 0-2 and 4-7, if the exception was recognized after secondary ASN translation in SET SECONDARY ASN WITH INSTANCE or PROGRAM RETURN.

During a program interruption due to a protection exception, information is stored at location 160 as described in "Suppression on Protection" on page 3-14.

161 (A1 hex) Real Address

PER Access Identification: During a program interruption due to a PER storage-alteration event, an indication of the address space to which the event applies may be stored at location 161. If the access used an AR-specified address-space-control element, the number of the access register used is stored in bit positions 4-7 of location 161, and zeros are stored in bit positions

0-3. The contents of location 161 are unpredictable if the access did not use an AR-specified address-space-control element.

Programming Note: The PER ASCE identification may be inspected to determine whether the PER storage-alteration event used an AR-specified ASCE. See “PER ASCE Identification (AI)” on page 4-27 for further details.

162 (A2 hex) Real Address

Operand Access Identification: When enhanced DAT does not apply, and a program interruption due to a page-translation exception is recognized by the MOVE PAGE instruction, the contents of the R₁ field of the instruction are stored in bit positions 0-3 of location 162, and the contents of the R₂ field are stored in bit positions 4-7. If the page-translation exception was recognized during the execution of an instruction other than MOVE PAGE, or if an ASCE-type, region-first-translation, region-second-translation, region-third-translation, or segment-translation exception was recognized, the contents of location 162 are unpredictable.

When enhanced DAT applies, and a program interruption due to a region-first-translation, region-second-translation, region-third-translation, segment-translation, or page-translation exception is recognized by the MOVE PAGE instruction, the contents of the R₁ and R₂ fields are stored in location 162 as described above. If any of the exceptions listed in the preceding sentence was recognized during the execution of an instruction other than MOVE PAGE, or if an ASCE-type exception was recognized, the contents of location 162 are unpredictable.

163 (A3 hex) Absolute Address

Store-Status Architectural-Mode Identification: During the execution of the store-status operation, zeros are stored in bit positions 0-6 of location 163, and a one is stored in bit position 7. A zero stored in bit position 7 indicates the ESA/390 architectural mode, and a one indicates the z/Architecture architectural mode.

163 (A3 hex) Real Address

Machine-Check Architectural-Mode Identification: During a machine-check interruption, zeros are stored in bit positions 0-6 of location 163, and a one is stored in bit position 7. A zero stored in

bit position 7 indicates the ESA/390 architectural mode, and a one indicates the z/Architecture architectural mode.

168-175 (A8-AF hex) Real Address

Translation-Exception Identification for DAT

Exceptions: During a program interruption due to an ASCE-type, region-first-translation, region-second-translation, region-third-translation, segment-translation, or page-translation exception, bits 0-51 of the virtual address causing the exception are stored in bit positions 0-51 of locations 168-175. This address is sometimes referred to as the translation-exception address.

When the access-exception-fetch/store-indication facility is installed, bits 52 and 53 of locations 168-175 are set as described below.

Access-Exception Fetch/Store Indication: Bit positions 52-53 of locations 168-175 contain an indication of whether the exception was due to a fetch or a store operation, as follows:

Bit	Bit	Meaning
52	53	
0	0	Could not determine whether the exception was due to a fetch or store
0	1	Exception was due to a store operation
1	0	Exception was due to a fetch operation
1	1	Reserved

When the access-exception-fetch/store-indication facility is not installed, bits 52 and 53 of locations 168-175 are unpredictable.

The access-exception fetch/store indication is stored for ASCE-type, page-translation, protection, region-first-translation, region-second-translation, region-third translation, and segment-translation exceptions.

For a storage-operand update reference, it is unpredictable whether a fetch or store access is indicated. When multiple access-exception conditions exist — for one or more operands — the condition recognized is determined based on the discussion of “Multiple Program-Interruption Conditions” on page 6-44.

The enhanced suppression on protection facility is a prerequisite for the access-exception-fetch/store-indication facility.

Bits 54-60 are unpredictable.

MOVE PAGE Indication: When enhanced DAT does not apply, and the exception was a page-translation exception that was recognized during the execution of MOVE PAGE, bit 61 of locations 168-175 is set to one. If the exception was a page-translation exception recognized during the execution of an instruction other than MOVE PAGE, bit 61 is set to zero. If the exception was an ASCE-type, region-first-translation, region-second-translation, region-third-translation, or segment-translation exception, bit 61 of locations 168-175 is unpredictable. See the definition of real location 162 for related information.

When enhanced DAT applies, and the exception was a region-first-, region-second, region-third-, segment-, or page-translation exception that was recognized during the execution of MOVE PAGE, bit 61 of locations 168-175 is set to one. If the exception was a region-first-, region-second, region-third-, segment-, or page-translation exception recognized during the execution of an instruction other than MOVE PAGE, bit 61 is set to zero. If the exception was an ASCE-type exception, bit 61 of locations 168-175 is unpredictable. See the definition of real location 162 for related information.

ASCE Identifier: Bits 62 and 63 of locations 168-175 are set to identify the address-space-control element (ASCE) used in the translation, as follows:

Bit Bit

62 63 Meaning

- | | | |
|---|---|---|
| 0 | 0 | Primary ASCE was used. |
| 0 | 1 | CPU was in the access-register mode, and either the access was an instruction fetch or it was a storage-operand reference that used an AR-specified ASCE (the access was not an implicit reference to the linkage stack). The exception access id, real location 160, can be examined to determine the ASCE used. However, if the primary, secondary, or home ASCE was used, bits 62 and 63 may be set to 00, 10, or 11, respectively, instead of 01. |
| 1 | 0 | Secondary ASCE was used. |
| 1 | 1 | Home ASCE was used (includes the case of an implicit reference to the linkage stack). |

The CPU may avoid setting bits 62 and 63 to 01 by recognizing that the access was an instruction fetch, that access-list-entry token 00000000 or 00000001 hex was used, or that the access-list-entry token designated, through an access-list

entry, an ASN-second-table entry containing an ASCE equal to the primary ASCE, secondary ASCE, or home ASCE.

Translation-Exception Identification for ASN-Translation Exceptions: During a program interruption due to an AFX-translation, ASX-translation, primary-authority, or secondary-authority exception, the ASN being translated is stored at locations 174 and 175, zeros are stored at locations 172 and 173, and the contents of locations 168-171 remain unchanged.

Translation-Exception Identification for Space-Switch Events: During a program interruption due to a space-switch event, an identification of the old instruction space is stored at locations 174 and 175, the old instruction-space space-switch-event-control bit is placed in bit position 0 and zeros are placed in bit positions 1-15 of locations 172 and 173, and the contents of locations 168-171 remain unchanged. The identification and bit stored are as follows:

- If the CPU was in the primary-space, secondary-space, or access-register mode before the operation, the old PASN, bits 48-63 of control register 4 before the operation, is stored at locations 174 and 175, and the old primary space-switch-event-control bit, bit 57 of control register 1 before the operation, is placed in bit position 0 of locations 172 and 173.
- If the CPU was in the home-space mode before the operation, zeros are stored at locations 174 and 175, and the home space-switch-event-control bit, bit 57 of control register 13, is placed in bit position 0 of locations 172 and 173.

Translation-Exception Identification During PC-Number Translation: During a program interruption due to an EX-translation or LX-translation exception recognized by PROGRAM CALL when ASN-and-LX reuse is not installed or is not enabled by a one value of the ASN-and-LX-reuse control in control register 0, bits 44-63 of the second-operand address used by PROGRAM CALL (a 20-bit PC number), with 12 zeros appended on the left, are stored at locations 172-175, and the contents of locations 168-171 remain unchanged.

During a program interruption due to an EX-translation, LFX-translation, LSTE-sequence, or LSX-translation exception recognized by PROGRAM CALL when ASN-and-LX reuse is installed and enabled and bit 44 of the second-operand address used by PROGRAM CALL is zero, bits 44-63 of the second-operand address (a 20-bit PC number), with 12 zeros appended on the left, are stored at locations 172-175. If bit 44 of the second-operand address is one, bits 32-63 of the address (a 32-bit PC number) are stored at locations 172-175. In either of these cases, the contents of locations 168-171 remain unchanged.

Translation-Exception Identification for Protection Exceptions: During a program interruption due to a protection exception, information is stored at locations 168-175 as described in “Suppression on Protection” on page 3-14 and in “Enhanced Suppression on Protection” on page 3-15.

176-183 (B0-B7 hex) Real Address

Monitor Code: During a program interruption due to a monitor event, the monitor code is stored at locations 176-183.

184-187 (B8-BB hex) Real Address

Subsystem-Identification Word: During an I/O interruption, the subsystem-identification word is stored at locations 184-187.

188-191 (BC-BF hex) Real Address

I/O-Interruption Parameter: During an I/O interruption, the interruption parameter from the associated subchannel is stored at locations 188-191.

192-195 (C0-C3 hex) Real Address

I/O-Interruption-Identification Word: During an I/O interruption, the I/O-interruption-identification word, which further identifies the source of the I/O interruption, is stored at locations 192-195.

200-203 (C8-CB hex) Real Address

STFL Facility List: The STORE FACILITY LIST instruction stores information at real locations 200-203. The information describes which facilities are provided by the configuration. The information stored is identical in format to the first 32 bits stored by the STORE FACILITY LIST

EXTENDED instruction. Figure 4-19, “Assigned Facility Bits” on page 4-67 shows the meanings of the assigned facility bits.

232-239 (E8-EF hex) Real Address

Machine-Check-Interruption Code: During a machine-check interruption, the machine-check-interruption code is stored at locations 232-239.

244-247 (F4-F7 hex) Real Address

External-Damage Code: During a machine-check interruption due to certain external-damage conditions, depending on the model, an external-damage code may be stored at locations 244-247.

248-255 (F8-FF hex) Real Address

Failing-Storage Address: During a machine-check interruption, a 64-bit failing-storage address may be stored at locations 248-255.

256-263 (100-107 hex) Real Address

Enhanced-Monitor Counter-Array Origin: Bits 0-60 of the doubleword at location 256, appended on the right with three binary zeros, form the 64-bit virtual address of the enhanced-monitor counter array in the home address space.

During the execution of the MONITOR CALL instruction when the enhanced-monitor facility is installed, both the monitor-mask bit and enhanced-monitor-mask bit in control register 8 corresponding to the monitor class are ones, and the monitor code derived from the first-operand address is less than the enhanced-monitor counter-array size in location 264, then the enhanced-monitor counter designated by the first-operand location is incremented by one.

Programming Note: It is recommended that the enhanced-monitor counter array be allocated on a cache-line boundary of the CPU’s first-level data cache. The cache line size may be determined by the EXTRACT CACHE ATTRIBUTES (ECAG) instruction.

264-267 (108-111 hex) Real Address

Enhanced-Monitor Counter-Array Size: The word at location 264 contains a 32-bit unsigned binary value that is referenced during the execution of the MONITOR CALL instruction when the

enhanced-monitor facility is installed, and both the monitor-mask bit and enhanced-monitor-mask bit in control register 8 corresponding to the monitor class are both ones. In this case, if the monitor code derived from the first-operand address is less than the enhanced-monitor counter-array size in location 264, then the counter designated by the first-operand location is incremented by one; otherwise, the enhanced-monitor exception counter at location 268 is incremented.

268-271 (10C-10F hex) Real Address

Enhanced-Monitor Exception Count: The word at location 268 contains a 32-bit unsigned binary value that may be updated during the execution of the MONITOR CALL instruction when the enhanced-monitor facility is installed, and the monitor-class bit and corresponding enhanced-monitor-class bit in control register 8 are both ones. The word is updated in either of the following two cases:

- The monitor code derived from the first-operand address is greater than or equal to than the enhanced-monitor counter-array size in location 264.
- The counter designated by the first-operand location is inaccessible.

272-279 (110-117 hex) Real Address

Breaking-Event Address: If the PER-3 facility is installed, then, during a program interruption, the contents of the breaking-event-address register are stored in locations 272-279. If the breaking-event-address-recording facility is not installed, this location remains unchanged.

288-303 (120-12F hex) Real Address

Restart Old PSW: The current PSW is stored as the old PSW at locations 288-303 during a restart interruption.

304-319 (130-13F hex) Real Address

External Old PSW: The current PSW is stored as the old PSW at locations 304-319 during an external interruption.

320-335 (140-14F hex) Real Address

Supervisor-Call Old PSW: The current PSW is stored as the old PSW at locations 320-335 during a supervisor-call interruption.

336-351 (150-15F hex) Real Address

Program Old PSW: The current PSW is stored as the old PSW at locations 336-351 during a program interruption.

352-367 (160-16F hex) Real Address

Machine-Check Old PSW: The current PSW is stored as the old PSW at locations 352-367 during a machine-check interruption.

368-383 (170-17F hex) Real Address

Input/Output Old PSW: The current PSW is stored as the old PSW at locations 368-383 during an I/O interruption.

416-431 (1A0-1AF hex) Real Address

Restart New PSW: The new PSW is fetched from locations 416-431 during a restart interruption.

432-447 (1B0-1BF hex) Real Address

External New PSW: The new PSW is fetched from locations 432-447 during an external interruption.

448-463 (1C0-1CF hex) Real Address

Supervisor-Call New PSW: The new PSW is fetched from locations 448-463 during a supervisor-call interruption.

464-479 (1D0-1DF hex) Real Address

Program New PSW: The new PSW is fetched from locations 464-479 during a program interruption.

480-495 (1E0-1EF hex) Real Address

Machine-Check New PSW: The new PSW is fetched from locations 480-495 during a machine-check interruption.

496-511 (1F0-1FF hex) Real Address

Input/Output New PSW: The new PSW is fetched from locations 496-511 during an I/O interruption.

4544-4607 (11C0-11FF hex) Real Address

Available for Programming: Locations 4544-4607 are available for use by programming.

4608-4735 (1200-127F hex) Absolute Address

Store-Status Floating-Point-Register Save Area: During the execution of the store-status operation, the contents of the floating-point registers are stored at locations 4608-4735.

4608-4735 (1200-127F hex) Real Address

Machine-Check Floating-Point-Register Save Area: During a machine-check interruption, the contents of the floating-point registers are stored at locations 4608-4735.

4736-4863 (1280-12FF hex) Absolute Address

Store-Status General-Register Save Area: During the execution of the store-status operation, the contents of the general registers are stored at locations 4736-4863.

| 4736-4863 (1280-12FF hex) Real Address

Machine-Check General-Register Save Area: During a machine-check interruption, the contents of the general registers are stored at locations 4736-4863.

4864-4879 (1300-130F hex) Absolute Address

Store-Status PSW Save Area: During the execution of the store-status operation, the contents of the current PSW are stored at locations 4864-4879.

4864-4879 (1300-130F hex) Real Address

Fixed-Logout Area: Depending on the model, logout information may be stored at locations 4864-4879 during a machine-check interruption.

4888-4891 (1318-131B hex) Absolute Address

Store-Status Prefix Save Area: During the execution of the store-status operation, the contents of the prefix register are stored at locations 4888-4891.

4892-4895 (131C-131F hex) Absolute Address

Store-Status Floating-Point-Control-Register Save Area: During the execution of the store-status operation, the contents of the floating-point control register are stored at locations 4892-4895.

4892-4895 (131C-131F hex). Real Address

Machine-Check Floating-Point-Control-Register Save Area: During a machine-check interruption, the contents of the floating-point control register are stored at locations 4892-4895.

4900-4903 (1324-1327 hex) Absolute Address

Store-Status TOD-Programmable-Register Save Area: During the execution of the store-status operation, the contents of the TOD programmable register are stored at locations 4900-4903.

4900-4903 (1324-1327 hex) Real Address

Machine-Check TOD-Programmable-Register Save Area: During a machine-check interruption, the contents of the TOD programmable register are stored at locations 4900-4903.

4904-4911 (1328-132F hex) Absolute Address

Store-Status CPU-Timer Save Area: During the execution of the store-status operation, the contents of the CPU timer are stored at locations 4904-4911.

4904-4911 (1328-132F hex) Real Address

Machine-Check CPU-Timer Save Area: During a machine-check interruption, the contents of the CPU timer are stored at locations 4904-4911.

4913-4919 (1331-1337 hex) Absolute Address

Store-Status Clock-Comparator Save Area: During the execution of the store-status operation, the contents of bit positions 0-55 of the clock comparator are stored at locations 4913-4919. When this store occurs, zeros are stored at location 4912.

4913-4919 (1331-1337 hex) Real Address

Machine-Check Clock-Comparator Save Area: During a machine-check interruption, the contents of bit positions 0-55 of the clock comparator are stored at locations 4913-4919. When this store occurs, zeros are stored at location 4912.

4928-4991 (1340-137F hex) Absolute Address

Store-Status Access-Register Save Area: During the execution of the store-status operation, the contents of the access registers are stored at locations 4928-4991.

4928-4991 (1340-137F hex) Real Address

Machine-Check Access-Register Save Area: During a machine-check interruption, the contents of the access registers are stored at locations 4928-4991.

4992-5119 (1380-13FF hex) Absolute Address

Store-Status Control-Register Save Area: During the execution of the store-status operation, the contents of the control registers are stored at locations 4992-5119.

4992-5119 (1380-13FF hex) Real Address

Machine-Check Control-Register Save Area: During a machine-check interruption, the contents of the control registers are stored at locations 4992-5119.

Programming Notes:

1. When the CPU is in the access-register mode, some instructions, such as MVCL, which

address operands in more than one address space, may cause a storage-alteration PER event in one address space concurrently with a region-translation, segment-translation, or page-translation exception in another address space. The access registers used to cause these conditions in such a case are different. In order to identify both access registers, two access identifications, namely the exception access identification and the PER access identification, are provided.

2. The store-status and machine-check architectural-mode identifications at absolute and real locations 163, respectively, indicate that the CPU is in the z/Architecture architectural mode. When z/Architecture is installed on the CPU but the CPU is in the ESA/390 mode, the store-status and machine-check-interruption operations store zero at location 163.

Hex	Dec	Fields			
0	0				
4	4				
:	:				
78	120				
7C	124				
80	128	External-Interruption Parameter			
84	132	CPU Address		External-Interruption Code	
88	136	0 0 0 0 0 0 0 0 0 0 0 0	ILC	0	SVC-Interruption Code
8C	140	0 0 0 0 0 0 0 0 0 0 0 0	ILC	0	Program-Interruption Code
90	144	Data-Exception Code			
94	148	Monitor-Class Number		PER Code	ATMID AI
98	152	PER Address			
9C	156				
A0	160	Exception Access ID	PER Access ID	Operand Access ID	SS/MC Arch.-Mode ID
A4	164				
A8	168	Translation-Exception Identification			
AC	172				
B0	176	Monitor Code			
B4	180				
B8	184	Subsystem-Identification Word			
BC	188	I/O-Interruption Parameter			
C0	192	I/O-Interruption-Identification Word			
C4	196				
C8	200	STFL Facility List			

Figure 3-13. Assigned Storage Locations (Part 1 of 4).

Hex	Dec	Fields
CC	204	
D0	208	
D4	212	
D8	216	
DC	220	
E0	224	
E4	228	
E8	232	Machine-Check Interruption Code
EC	236	
F0	240	
F4	244	External-Damage Code
F8	248	Failing-Storage Address
FC	252	
100	256	Enhanced-Monitor Counter-Array Origin
104	260	
108	264	
10C	268	
110	272	Breaking-Event Address
114	276	
118	280	
11C	284	
120	288	Restart Old PSW
124	292	
128	296	
12C	300	
130	304	External Old PSW
134	308	
138	312	
13C	316	
140	320	Supervisor-Call Old PSW
144	324	
148	328	
14C	332	
150	336	Program Old PSW
154	340	
158	344	
15C	348	
160	352	Machine-Check Old PSW
164	356	
168	360	
16C	364	
170	368	Input/Output Old PSW
174	372	
178	376	
17C	380	

Figure 3-13. Assigned Storage Locations (Part 2 of 4).

Hex	Dec	Fields
180	384	
184	388	
188	392	
18C	396	
190	400	
194	404	
198	408	
19C	412	
1A0	416	Restart New PSW
1A4	420	
1A8	424	
1AC	428	
1B0	432	External New PSW
1B4	436	
1B8	440	
1BC	444	
1C0	448	Supervisor-Call New PSW
1C4	452	
1C8	456	
1CC	460	
1D0	464	Program New PSW
1D4	468	
1D8	472	
1DC	476	
1E0	480	Machine-Check New PSW
1E4	484	
1E8	488	
1EC	492	
1F0	496	Input/Output New PSW
1F4	500	
1F8	504	
1FC	508	
200	512	
204	516	
⋮	⋮	
FF8	4088	
FFC	4092	
1000	4096	
⋮	⋮	
11BC	4540	
11C0	4544	Available for Use by Programming (64 bytes)
11C4	4548	
⋮	⋮	
11F8	4600	
11FC	4604	Store-Status or Machine-Check Floating-Point-Register Save Area (128 bytes)
1200	4608	
1204	4612	
⋮	⋮	
1278	4728	
127C	4732	

Figure 3-13. Assigned Storage Locations (Part 3 of 4).

Hex	Dec	Fields
1280	4736	Store-Status or Machine-Check General-Register Save Area (128 bytes)
1284	4740	
⋮	⋮	
12F8	4856	
12FC	4860	
1300	4864	Store-Status PSW Save Area; or Fixed Logout Area
1304	4868	
1308	4872	
130C	4876	
1310	4880	
1314	4884	
1318	4888	Store-Status Prefix Save Area
131C	4892	Store-Status or Machine-Check Floating-Point-Control-Register Save Area
1320	4896	
1324	4900	Store-Status or Machine-Check TOD-Programmable-Register Save Area
1328	4904	Store-Status or Machine-Check CPU-Timer Save Area
132C	4908	
1330	4912	Store-Status or Machine-Check Clock-Comparator Bits 0-55 Save Area
1334	4916	
1338	4920	
133C	4924	
1340	4928	Store-Status or Machine-Check Access-Register Save Area (64 bytes)
1344	4932	
⋮	⋮	
1378	4994	
137C	4998	
1380	4992	Store-Status or Machine-Check Control-Register Save Area (128 bytes)
1384	4996	
⋮	⋮	
13F8	5112	
13FC	5116	

Figure 3-13. Assigned Storage Locations (Part 4 of 4).

Chapter 4. Control

Stopped, Operating, Load, and Check-Stop States	4-2	TOD-Clock Steering	4-44
Stopped State	4-2	TOD-Clock Steering Overview	4-45
Operating State	4-2	TOD-Offset-Update Events	4-47
Load State	4-3	Episodes	4-47
Check-Stop State	4-3	TOD-Clock-Steering Registers	4-47
Program-Status Word	4-3	Clock Comparator	4-48
Program-Status-Word Format	4-5	CPU Timer	4-49
Control Registers	4-7	Externally Initiated Functions	4-50
Tracing	4-10	Resets	4-50
Control-Register Allocation	4-12	CPU Reset	4-53
Trace Entries	4-12	Initial CPU Reset	4-54
Operation	4-22	Subsystem Reset	4-54
Program-Event Recording	4-23	Clear Reset	4-54
PER Instruction-Fetching Nullification	4-23	Power-On Reset	4-55
Control-Register Allocation and Address-Space-Control Element	4-23	Initial Program Loading	4-56
Operation	4-25	CCW-Type IPL	4-56
Identification of Cause	4-26	Store Status	4-56
Priority of Indication	4-29	Multiprocessing	4-57
Storage-Area Designation	4-30	Shared Main Storage	4-57
PER Events	4-31	CPU-Address Identification	4-57
Successful Branching	4-31	CPU Signaling and Response	4-58
Instruction Fetching	4-31	Signal-Processor Orders	4-58
Storage Alteration	4-32	Sense	4-58
Store Using Real Address	4-32	External Call	4-58
Indication of PER Events Concurrently with Other Interruption Conditions	4-33	Emergency Signal	4-58
Breaking-Event-Address Recording	4-36	Start	4-58
Breaking-Event-Address Register	4-37	Stop	4-58
Execution-Break Instructions	4-37	Restart	4-59
Timing	4-38	Stop and Store Status	4-59
Time-of-Day Clock	4-38	Initial CPU Reset	4-59
Format	4-38	CPU Reset	4-59
States	4-38	Set Prefix	4-59
Changes in Clock State	4-39	Store Status at Address	4-59
Setting and Inspecting the Clock	4-40	Set Architecture	4-60
TOD Programmable Register	4-41	Conditional Emergency Signal	4-61
TOD-Clock Synchronization	4-43	Sense Running Status:	4-62
Timing Mode	4-43	Conditions Determining Response	4-62
Timing State	4-44	Conditions Precluding Interpretation of the Order Code	4-62
STP Clock Source State	4-44	Status Bits	4-63
		Facility Indications	4-66

This chapter describes in detail the facilities for controlling, measuring, and recording the operation of one or more CPUs.

Stopped, Operating, Load, and Check-Stop States

The stopped, operating, load, and check-stop states are four mutually exclusive states of the CPU. When the CPU is in the stopped state, instructions and interruptions, other than the restart interruption, are not executed. In the operating state, the CPU executes instructions and takes interruptions, subject to the control of the program-status word (PSW) and control registers, and in the manner specified by the setting of the operator-facility rate control. The CPU is in the load state during the initial-program-loading operation of ESA/390. The CPU enters the check-stop state only as the result of machine malfunctions.

A change between these four CPU states can be effected by use of the operator facilities or by acceptance of certain SIGNAL PROCESSOR orders addressed to that CPU. The states are not controlled or identified by bits in the PSW. The stopped, load, and check-stop states are indicated to the operator by means of the manual indicator, load indicator, and check-stop indicator, respectively. These three indicators are off when the CPU is in the operating state.

The CPU timer is updated when the CPU is in the operating state or the load state. The TOD clock is not affected by the state of any CPU.

Stopped State

The CPU changes from the operating state to the stopped state by means of the stop function. The stop function is performed when:

- The stop key is activated while the CPU is in the operating state.
- The CPU accepts a stop or stop-and-store-status order specified by a SIGNAL PROCESSOR instruction addressed to this CPU while it is in the operating state.
- The CPU has finished the execution of a unit of operation initiated by performing the start function with the rate control set to the instruction-step position.

When the stop function is performed, the transition from the operating to the stopped state occurs at the

end of the current unit of operation. When the wait-state bit of the PSW is one, the transition takes place immediately, provided no interruptions are pending for which the CPU is enabled. In the case of interruptible instructions, the amount of data processed in a unit of operation depends on the particular instruction and may depend on the model.

Before entering the stopped state by means of the stop function, all pending allowed interruptions occur while the CPU is still in the operating state. They cause the old PSW to be stored and the new PSW to be fetched before the stopped state is entered. While the CPU is in the stopped state, interruption conditions remain pending.

The CPU is also placed in the stopped state when:

- CPU reset is completed. However, when the reset operation is performed as part of initial program loading for this CPU, then the CPU is placed in the load state and does not necessarily enter the stopped state.
- An address comparison indicates equality and stopping on the match is specified.

The execution of resets is described in “Resets” on page 4-50, and address comparison is described in “Address-Compare Controls” on page 12-1.

If the CPU is in the stopped state when an INVALIDATE PAGE TABLE ENTRY instruction is executed on another CPU in the configuration, the clearing of TLB entries is completed before the CPU leaves the stopped state.

Operating State

The CPU changes from the stopped state to the operating state by means of the start function or when a restart interruption (see “Restart Interruption” on page 6-49) occurs.

The start function is performed if the CPU is in the stopped state and (1) the start key associated with that CPU is activated or (2) that CPU accepts the start order specified by a SIGNAL PROCESSOR instruction addressed to that CPU. The effect of performing the start function is unpredictable when the stopped state has been entered by means of a reset.

When the rate control is set to the process position and the start function is performed, the CPU starts operating at normal speed. When the rate control is set to the instruction-step position and the wait-state bit is zero, one instruction or, for interruptible instructions, one unit of operation is executed, and all pending allowed interruptions occur before the CPU returns to the stopped state. When the rate control is set to the instruction-step position and the wait-state bit is one, the start function does not cause an instruction to be executed, but all pending allowed interruptions occur before the CPU returns to the stopped state.

Load State

The CPU enters the load state when the load-normal or load-clear key is activated. (See “Initial Program Loading” on page 4-56. See also “Initial Program Loading” on page 17-16.) This sets the architectural mode to the ESA/390 mode. For ease of reference, the additional elements of the description of the ESA/390 load state are given below.

If a CCW-type initial-program-loading operation is completed successfully, the CPU changes from the load state to the operating state, provided the rate control is set to the process position; if the rate control is set to the instruction-step position, the CPU changes from the load state to the stopped state.

Check-Stop State

The check-stop state, which the CPU enters on certain types of machine malfunction, is described in “Check-Stop State” on page 11-9. The CPU leaves the check-stop state when CPU reset is performed.

Programming Notes:

1. Except for the relationship between execution time and real time, the execution of a program is not affected by stopping the CPU.
2. When, because of a machine malfunction, the CPU is unable to end the execution of an instruction, the stop function is ineffective, and a reset function has to be invoked instead. A similar situation occurs when an unending string of interruptions results from a PSW with a PSW-format error of the type that is recognized early, or from

a persistent interruption condition, such as one due to the CPU timer.

3. Pending I/O operations may be initiated, and active I/O operations continue to suspension or completion, after the CPU enters the stopped state. The interruption conditions due to suspension or completion of I/O operations remain pending when the CPU is in the stopped state.

Program-Status Word

The current program-status word (PSW) in the CPU contains information required for the execution of the currently active program. The PSW is 128 bits in length and includes the instruction address, condition code, and other control fields. In general, the PSW is used to control instruction sequencing and to hold and indicate much of the status of the CPU in relation to the program currently being executed. Additional control and status information is contained in control registers and permanently assigned storage locations.

The status of the CPU can be changed by loading a new PSW or part of a PSW.

Control is switched during an interruption of the CPU by storing the current PSW, so as to preserve the status of the CPU, and then loading a new PSW.

Execution of LOAD PSW or LOAD PSW EXTENDED, or the successful conclusion of the initial-program-loading sequence, introduces a new PSW. The instruction address is updated by sequential instruction execution and replaced by successful branches. Other instructions are provided which operate on a portion of the PSW. Figure 4-1 on page 4-4 summarizes these instructions.

A new or modified PSW becomes active (that is, the information introduced into the current PSW assumes control over the CPU) when the interruption or the execution of an instruction that changes the PSW is completed. The interruption for PER associated with an instruction that changes the PSW occurs under control of the PER mask that is effective at the beginning of the operation.

Bits 0-7 of the PSW are collectively referred to as the system mask.

Instruction	System Mask (PSW Bits 0-7)		PSW Key (PSW Bits 8-11)		Problem State (PSW Bit 15)		Address-Space Control (PSW Bits 16-17)		Condition Code and Program Mask (PSW Bits 18-23)		Basic Addressing Mode (PSW Bit 32)		Extended Addressing Mode (PSW Bit 31)	
	Saved	Set	Saved	Set	Saved	Set	Saved	Set	Saved	Set	Saved	Set	Saved	Set
BRANCH AND LINK	—	—	—	—	—	—	—	—	24AM	—	31AM	—	—	—
BRANCH AND SAVE	—	—	—	—	—	—	—	—	—	—	BAM	—	—	—
BRANCH AND SAVE AND SET MODE	—	—	—	—	—	—	—	—	—	—	BAM	Yes ¹	Yes	Yes ¹
BRANCH AND SET AUTHORITY	—	—	Yes	Yes	Yes	Yes	—	—	—	—	BAM ²	BAM	—	—
BRANCH AND SET MODE	—	—	—	—	—	—	—	—	—	—	BAM ¹	Yes ¹	Yes ¹	Yes ¹
BRANCH AND STACK	Yes	—	Yes	—	Yes	—	Yes	—	Yes	—	BAM ³	—	Yes	—
BRANCH IN SUPSPACE GROUP	—	—	—	—	—	—	—	—	—	—	BAM ¹	BAM	—	—
BRANCH RELATIVE AND SAVE	—	—	—	—	—	—	—	—	—	—	BAM	—	—	—
BRANCH RELATIVE AND SAVE LONG	—	—	—	—	—	—	—	—	—	—	BAM	—	—	—
EXTRACT PSW	Yes	—	Yes	—	Yes	—	Yes	—	Yes	—	Yes	—	Yes	—
INSERT PROGRAM MASK	—	—	—	—	—	—	—	—	Yes	—	—	—	—	—
INSERT PSW KEY	—	—	Yes	—	—	—	—	—	—	—	—	—	—	—
INSERT ADDRESS SPACE CONTROL	—	—	—	—	—	—	Yes	—	—	—	—	—	—	—
Basic PROGRAM CALL	—	—	—	—	Yes	Yes	—	—	—	—	BAM	BAM	—	—
Stacking PROGRAM CALL	Yes	—	Yes	PKC	Yes	Yes	Yes	Yes	Yes	—	Yes	Yes	Yes	Yes
PROGRAM RETURN	—	Yes ⁴	—	Yes	—	Yes	—	Yes	—	Yes	—	Yes	—	Yes
PROGRAM TRANSFER	—	—	—	—	—	Yes ⁵	—	—	—	—	—	BAM	—	—
RESUME PROGRAM	—	—	—	—	—	—	—	Yes	—	Yes	—	Yes	—	Yes
SET ADDRESS SPACE CONTROL	—	—	—	—	—	—	—	Yes	—	—	—	—	—	—
SET ADDRESSING MODE	—	—	—	—	—	—	—	—	—	—	—	Yes	—	Yes
SET PROGRAM MASK	—	—	—	—	—	—	—	—	—	Yes	—	—	—	—
SET PSW KEY FROM ADDRESS	—	—	—	Yes	—	—	—	—	—	—	—	—	—	—
SET SYSTEM MASK	—	Yes	—	—	—	—	—	—	—	—	—	—	—	—
STORE THEN AND SYSTEM MASK	Yes	ANDs	—	—	—	—	—	—	—	—	—	—	—	—
STORE THEN OR SYSTEM MASK	Yes	ORs	—	—	—	—	—	—	—	—	—	—	—	—
TRAP	—	—	—	—	Yes	—	Yes	Yes	Yes	—	Yes	Yes	Yes	—
Explanation:														
—	No.													
¹	The action takes place only if the associated R field in the instruction is nonzero.													
²	In the reduced-authority state, the action takes place only if the R _i field in the instruction is nonzero.													
³	The action also takes place in the 64-bit addressing mode if the R _i field in the instruction is zero.													
⁴	PROGRAM RETURN does not change the PER mask.													
⁵	PROGRAM TRANSFER does not change the problem-state bit from one to zero.													
BAM	The basic-addressing-mode bit is saved or set in the 24-bit or 31-bit addressing mode.													
ANDs	The logical AND of the immediate field in the instruction and the current system mask replaces the current system mask.													
ORs	The logical OR of the immediate field in the instruction and the current system mask replaces the current system mask.													
PKC	When the PSW-key-control bit, bit 131 of the entry-table entry, is zero, the PSW key remains unchanged. When the PSW-key-control bit is one, the PSW key is set with the entry key, bits 136-139 of the entry-table entry.													
24AM	The condition code and program mask are saved in the 24-bit addressing mode.													
31AM	The basic-addressing-mode bit is saved in the 31-bit addressing mode.													

Figure 4-1. Operations on PSW Fields.

Programming Note: A summary of the operations which save or set the problem state, addressing mode, and instruction address is contained in “Subroutine Linkage without the Linkage Stack” on page 5-11.

Program-Status-Word Format

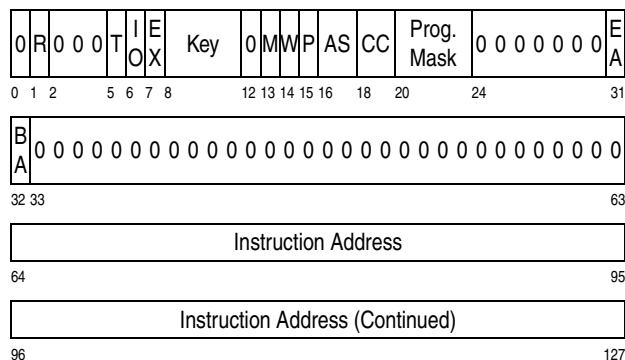


Figure 4-2. PSW Format

The following is a summary of the functions of the PSW fields. (See Figure 4-2.)

PER Mask (R): Bit 1 controls whether the CPU is enabled for interruptions associated with program-event recording (PER). When the bit is zero, no PER event can cause an interruption. When the bit is one, interruptions are permitted, subject to the PER-event-mask bits in control register 9.

DAT Mode (T): Bit 5 controls whether implicit dynamic address translation of logical and instruction addresses used to access storage takes place. When the bit is zero, DAT is off, and logical and instruction addresses are treated as real addresses. When the bit is one, DAT is on, and the dynamic-address-translation mechanism is invoked.

I/O Mask (IO): Bit 6 controls whether the CPU is enabled for I/O interruptions. When the bit is zero, an I/O interruption cannot occur. When the bit is one, I/O interruptions are subject to the I/O-interruption subclass-mask bits in control register 6. When an I/O-interruption subclass-mask bit is zero, an I/O interruption for that I/O-interruption subclass cannot occur; when the I/O-interruption subclass-mask bit is one, an I/O interruption for that I/O-interruption subclass can occur.

External Mask (EX): Bit 7 controls whether the CPU is enabled for interruption by conditions

included in the external class. When the bit is zero, an external interruption cannot occur. When the bit is one, an external interruption is subject to the corresponding external subclass-mask bits in control register 0; when the subclass-mask bit is zero, conditions associated with the subclass cannot cause an interruption; when the subclass-mask bit is one, an interruption in that subclass can occur.

PSW Key: Bits 8-11 form the access key for storage references by the CPU. If the reference is subject to key-controlled protection, the PSW key is matched with a storage key when information is stored or when information is fetched from a location that is protected against fetching. However, for one of the operands of each of MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH KEY, MOVE WITH SOURCE KEY, and MOVE WITH DESTINATION KEY, an access key specified as an operand is used instead of the PSW key.

Machine-Check Mask (M): Bit 13 controls whether the CPU is enabled for interruption by machine-check conditions. When the bit is zero, a machine-check interruption cannot occur. When the bit is one, machine-check interruptions due to system damage and instruction-processing damage are permitted, but interruptions due to other machine-check-subclass conditions are subject to the subclass-mask bits in control register 14.

Wait State (W): When bit 14 is one, the CPU is waiting; that is, no instructions are processed by the CPU, but interruptions may take place. When bit 14 is zero, instruction fetching and execution occur in the normal manner. The wait indicator is on when the bit is one.

Problem State (P): When bit 15 is one, the CPU is in the problem state. When bit 15 is zero, the CPU is in the supervisor state. In the supervisor state, all instructions are valid. In the problem state, only those instructions are valid that provide meaningful information to the problem program and that cannot affect system integrity; such instructions are called unprivileged instructions. The instructions that are never valid in the problem state are called privileged instructions. When a CPU in the problem state attempts to execute a privileged instruction, a privileged-operation exception is recognized. Another group of instructions, called semiprivileged instructions, are executed by a CPU in the problem state only if specific authority tests are met; otherwise, a

privileged-operation exception or a special-operation exception is recognized.

Address-Space Control (AS): Bits 16 and 17, in conjunction with PSW bit 5, control the translation mode. See “Translation Modes” on page 3-36.

Condition Code (CC): Bits 18 and 19 are the two bits of the condition code. The condition code is set to 0, 1, 2, or 3, depending on the result obtained in executing certain instructions. Most arithmetic and logical operations, as well as some other operations, set the condition code. The instruction BRANCH ON CONDITION can specify any selection of the condition-code values as a criterion for branching. A table in Appendix C summarizes the condition-code values that may be set for all instructions which set the condition code of the PSW.

Program Mask: Bits 20-23 are the four program-mask bits. Each bit is associated with a program exception, as follows:

Program-Mask Bit	Program Exception
20	Fixed-point overflow
21	Decimal overflow
22	HFP exponent underflow
23	HFP significance

When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The setting of the HFP-exponent-underflow-mask bit or the HFP-significance-mask bit also determines the manner in which the operation is completed when the corresponding exception occurs.

Extended Addressing Mode (EA): Bit 31 controls the size of effective addresses and effective-address generation in conjunction with bit 32, the basic-addressing-mode bit. When bit 31 is zero, the addressing mode is controlled by bit 32. When bits 31 and 32 are both one, 64-bit addressing is specified.

Basic Addressing Mode (BA): Bits 31 and 32 control the size of effective addresses and effective-address generation. When bits 31 and 32 are both zero, 24-bit addressing is specified. When bit 31 is zero and bit 32 is one, 31-bit addressing is specified. When bits 31 and 32 are both one, 64-bit addressing is specified. Bit 31 one and bit 32 zero is an invalid

combination that causes a specification exception to be recognized. The addressing mode does not control the size of PER addresses or of addresses used to access DAT, ASN, dispatchable-unit-control, linkage, entry, and trace tables or access lists or the linkage stack. See “Address Generation” on page 5-8 and “Address Size and Wraparound” on page 3-5. The control of the addressing mode by bits 31 and 32 of the PSW is summarized as follows:

PSW.31	PSW.32	Addressing Mode
0	0	24-bit
0	1	31-bit
1	1	64-bit

Instruction Address: Bits 64-127 of the PSW are the instruction address. This address designates the location of the leftmost byte of the next instruction to be executed, unless the CPU is in the wait state (bit 14 of the PSW is one).

Bit positions 0, 2-4, 24-30, and 33-63 are unassigned and must contain zeros. A specification exception is recognized when these bit positions do not contain zeros.

When bits 31 and 32 of the PSW specify the 24-bit addressing mode, bits 64-103 of the instruction address must be zeros, or, when bits 31 and 32 specify the 31-bit mode, bits 64-96 must be zeros. Otherwise, a specification exception is recognized. A specification exception is also recognized when bit 31 is one and bit 32 is zero or when bit position 12 does not contain a zero.

LOAD PSW EXTENDED has a 16-byte second operand. The instruction loads the operand unchanged and without examination as the current PSW.

LOAD PSW has an eight-byte second operand. The operand is treated as an ESA/390 PSW, except that bit 31 (the z/Architecture extended-addressing-mode bit) may be one.

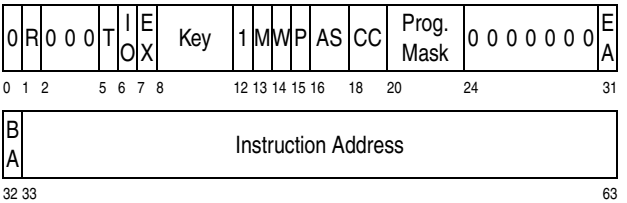


Figure 4-3. ESA/390 PSW Format, Except Bit 31 Shown as EA

Depending on the model, either LOAD PSW recognizes a specification exception if bit 12 of its second operand is not one or this error is indicated by an early specification exception after the completion of the execution of LOAD PSW. LOAD PSW loads bits 0-32 of its second operand, except with bit 12 inverted, and bits 33-63 of the operand as bits 0-32 and 97-127, respectively, of the current PSW, and it sets bits 33-96 of the current PSW to zeros.

Control Registers

The control registers provide for maintaining and manipulating control information outside the PSW. There are sixteen 64-bit control registers.

The LOAD CONTROL (LCTLG) instruction causes all control-register bit positions within those registers designated by the instruction to be loaded from storage. The LOAD CONTROL (LCTL) instruction loads only bit positions 32-63 of the control registers, and bits 0-31 of the registers remain unchanged. The instructions BRANCH AND SET AUTHORITY, BRANCH AND STACK, BRANCH IN SUBSPACE GROUP, EXTRACT AND SET EXTENDED AUTHORITY, LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, and SET SECONDARY ASN WITH INSTANCE, provide specialized functions to place information into certain control-register bit positions.

Information loaded into the control registers becomes active (that is, assumes control over the system) at the completion of the instruction that causes the information to be loaded.

At the time the registers are loaded, the information is not checked for exceptions, such as an address

designating an unavailable or protected location. The validity of the information is checked and the exceptions, if any, are indicated at the time the information is used.

The STORE CONTROL (STCTG) instruction causes the contents of all control-register bit positions, within those registers designated by the instruction, to be placed in storage. The STORE CONTROL (STCTL) instruction places the contents of bit positions 32-63 of the control registers in storage, and bits 0-31 of the registers are ignored. The instructions EXTRACT AND SET EXTENDED AUTHORITY, EXTRACT PRIMARY ASN, EXTRACT PRIMARY ASN AND INSTANCE, EXTRACT SECONDARY ASN, EXTRACT SECONDARY ASN AND INSTANCE, and PROGRAM CALL provide specialized functions to obtain information from certain control-register bit positions.

Only the general structure of the control registers is described here; the definition of a particular control-register bit position appears in the description of the facility with which the position is associated. Figure 4-4 on page 4-8 shows the control-register bit positions which are assigned and the initial values of the positions upon execution of initial CPU reset. All control-register bit positions not listed in the figure are initialized to zero.

Programming Notes:

1. The detailed definition of a particular control-register bit position can be located by referring to the entry "control-register assignment" in the Index.
2. To ensure that existing programs operate correctly if and when new facilities using additional control-register bit positions are installed, the program should load zeros in unassigned positions.

Control Register	Bits	Name of Field	Associated with	Initial Value
0	15	Measurement-counter-extraction-authorization control	CPU measurement ⁴	0
0	32	TRACE TOD-clock control	Tracing	0
0	33	SSM-suppression control	SET SYSTEM MASK	0
0	34	TOD-clock-sync control	TOD clock	0
0	35	Low-address-protection control	Low-address-protection	0
0	36	Extraction-authority control	Instruction authorization	0
0	37	Secondary-space control	Instruction authorization	0
0	38	Fetch-protection-override control	Key-controlled protection	0
0	39	Storage-protection-override control	Key-controlled protection	0
0	40	Enhanced-DAT-enablement control	Dynamic address translation	0
0	44	ASN-and-LX-reuse control	Instruction authorization	0
0	45	AFP-register control	Floating point	0
0	48	Malfunction-alert subclass mask	External interruptions	0
0	49	Emergency-signal subclass mask	External interruptions	0
0	50	External-call subclass mask	External interruptions	0
0	52	Clock-comparator subclass mask	External interruptions	0
0	53	CPU-timer subclass mask	External interruptions	0
0	54	Service-signal subclass mask	External interruptions	0
0	56	Unused ¹		1
0	57	Interrupt-key subclass mask	External interruptions	1
0	58	Measurement-alert subclass mask	CPU measurement ⁴	1
0	59	Timing-alert subclass mask	External interruptions	0
0	61	Crypto control	Cryptography	0
1	0-51	Primary region-table origin ²	Dynamic address translation	0
1	0-51	Primary segment-table origin ²	Dynamic address translation	0
1	0-51	Primary real-space token origin ²	Dynamic address translation	0
1	54	Primary subspace-group control	Subspace groups	0
1	55	Primary private-space control	Dynamic address translation	0
1	56	Primary storage-alteration-event	Program-event recording control	0
1	57	Primary space-switch-event control	Program interruptions	0
1	58	Primary real-space control	Dynamic address translation	0
1	60-61	Primary designation-type control ³	Dynamic address translation	0
1	62-63	Primary table length ³	Dynamic address translation	0
2	33-57	Dispatchable-unit-control-table origin	Access-register translation	0
3	0-31	Secondary ASN-second-table-entry instance number	Instruction authorization	0
3	32-47	PSW-key mask	Instruction authorization	0
3	48-63	Secondary ASN	Address spaces	0
4	0-31	Primary ASN-second-table-entry instance number	Instruction authorization	0
4	32-47	Authorization index	Instruction authorization	0
4	48-63	Primary ASN	Address spaces	0
5	33-57	Primary-ASN-second-table-entry origin	Access-register translation	0
6	32-39	I/O-interruption subclass mask	I/O interruptions	0

Figure 4-4. Assignment of Control-Register Fields (Part 1 of 3)

Control Register	Bits	Name of Field	Associated with	Initial Value
7	0-51	Secondary region-table origin ²	Dynamic address translation	0
7	0-51	Secondary segment-table origin ²	Dynamic address translation	0
7	0-51	Secondary real-space token origin ²	Dynamic address translation	0
7	54	Secondary subspace-group control	Subspace groups	0
7	55	Secondary private-space control	Dynamic address translation	0
7	56	Secondary storage-alteration-event	Program-event recording control	0
7	58	Secondary real-space control	Dynamic address translation	0
7	60-61	Secondary designation-type control ³	Dynamic address translation	0
7	62-63	Secondary table length ³	Dynamic address translation	0
8	16-31	Enhanced-monitor masks	MONITOR CALL	0
8	32-47	Extended authorization index	Access-register translation	0
8	48-63	Monitor masks	MONITOR CALL	0
9	32	Successful-branching-event mask	Program-event recording	0
9	33	Instruction-fetching-event mask	Program-event recording	0
9	34	Storage-alteration-event mask	Program-event recording	0
9	36	Store-using-real-address-event mask	Program-event recording	0
9	39	Instruction-fetching-nullification-event mask	Program-event recording	0
9	40	Branch-address control	Program-event recording	0
9	42	Storage-alteration-space control	Program-event recording	0
10	0-63	PER starting address	Program-event recording	0
11	0-63	PER ending address	Program-event recording	0
12	0	Branch-trace control	Tracing	0
12	1	Mode-trace control	Tracing	0
12	2-61	Trace-entry address	Tracing	0
12	62	ASN-trace control	Tracing	0
12	63	Explicit-trace control	Tracing	0
13	0-51	Home region-table origin ²	Dynamic address translation	0
13	0-51	Home segment-table origin ²	Dynamic address translation	0
13	0-51	Home real-space token origin ²	Dynamic address translation	0
13	55	Home private-space control	Dynamic address translation	0
13	56	Home storage-alteration-event	Program-event recording control	0
13	57	Home space-switch-event control	Program interruptions	0
13	58	Home real-space control	Dynamic address translation	0
13	60-61	Home designation-type control ³	Dynamic address translation	0
13	62-63	Home table length ³	Dynamic address translation	0
14	32	Unused ¹		1
14	33	Unused ¹		1
14	35	Channel-report-pending subclass mask	I/O machine-check handling	0
14	36	Recovery subclass mask	Machine-check handling	0
14	37	Degradation subclass mask	Machine-check handling	0
14	38	External-damage subclass mask	Machine-check handling	1
14	39	Warning subclass mask	Machine-check handling	0
14	42	TOD-clock-control-override control	TOD clock	0
14	44	ASN-translation control	Instruction authorization	0
14	45-63	ASN-first-table origin	ASN translation	0
15	0-60	Linkage-stack-entry address	Linkage-stack operations	0

Figure 4-4. Assignment of Control-Register Fields (Part 2 of 3)

Control Register	Bits	Name of Field	Associated with	Initial Value
Explanation: The fields not listed are unassigned. The initial value for all unlisted control-register bit positions is zero.				
1		This bit is not used but is initialized to one for consistency with the System/370 definition.		
2		The address-space-control element (ASCE) in the control register has one of three formats, depending on bit 58 of the register, the real-space control, and bits 60 and 61 of the register, the designation-type control. When bit 58 is zero, the ASCE is a region-table designation if bits 60 and 61 are 11, 10, or 01 binary, or it is a segment-table designation if bits 60 and 61 are 00 binary. When bit 58 is one, the ASCE is a real-space designation. Bits 0-51 are the region-table origin, the segment-table origin or the real-space token origin, depending on whether the ASCE is a region-table designation, a segment-table designation, or a real-space designation, respectively.		
3		Bits 60-63 are assigned when the ASCE in the control register is a region-table designation or a segment-table designation.		
4		See <i>The Set-Program-Parameter and CPU-Measurement Facilities (SA23-2260)</i> for details on these bits.		

Figure 4-4. Assignment of Control-Register Fields (Part 3 of 3)

Tracing

Tracing assists in the determination of system problems by providing an ongoing record in storage of significant events. Tracing consists of four separately controllable functions which cause entries to be made in a trace table: branch tracing, ASN tracing, mode tracing, and explicit tracing. Branch tracing, ASN tracing, and mode tracing together are referred to as implicit tracing.

When branch tracing is on, a branch trace entry is made in the trace table for each execution of certain branch instructions when they cause branching. The branch address is placed in the trace entry. The trace entry also indicates the following about the addressing mode in effect after branching and the branch address: (1) the CPU is in the 24-bit addressing mode, (2) the CPU either is in the 31-bit addressing mode or is in the 64-bit addressing mode and bits 0-32 of the branch address are all zeros, or (3) the CPU is in the 64-bit addressing mode and bits 0-32 of the branch address are not all zeros. The branch instructions that are traced are:

- BRANCH AND LINK (BALR only) when the R_2 field is not zero
- BRANCH AND SAVE (BASR only) when the R_2 field is not zero
- BRANCH AND SAVE AND SET MODE when the R_2 field is not zero
- BRANCH AND SET AUTHORITY

- BRANCH AND STACK when the R_2 field is not zero
- BRANCH IN SUBSPACE GROUP
- RESUME PROGRAM
- TRAP

However, a branch trace entry is made for BRANCH IN SUBSPACE GROUP only if ASN tracing is not on.

If both branch tracing and mode tracing are on and BRANCH AND SAVE AND SET MODE or RESUME PROGRAM changes the extended-addressing-mode bit, PSW bit 31, a mode-switching-branch trace entry is made instead of a branch trace entry.

When ASN tracing is on, an entry named the same as the instruction is made in the trace table for each execution of the following instructions:

- BRANCH IN SUBSPACE GROUP
- PROGRAM CALL
- PROGRAM RETURN
- PROGRAM TRANSFER
- SET SECONDARY ASN

However, the entry for PROGRAM RETURN is made only when PROGRAM RETURN unstacks a linkage-stack state entry that was formed by PROGRAM CALL, not when PROGRAM RETURN unstacks an entry formed by BRANCH AND STACK.

When ASN tracing is on, a PROGRAM TRANSFER trace entry is also made for each execution of PROGRAM TRANSFER WITH INSTANCE, and a SET SECONDARY ASN trace entry is also made for each

execution of SET SECONDARY ASN WITH INSTANCE. In either case, a bit in the trace entry indicates whether the entry was made due to the without-instance or the with-instance instruction.

If both ASN tracing and mode tracing are on and PROGRAM CALL uses a 20-bit PC number and changes PSW bit 31, first a PROGRAM CALL trace entry is made, and then a mode-switch trace entry is made. In this case except when PROGRAM CALL uses a 32-bit PC number, only a PROGRAM CALL trace entry is made since it indicates the old and new values of the extended-addressing-mode bit, PSW bit 31. A 32-bit PC number may be used if the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0.

Mode tracing records a switch from a basic (24-bit or 31-bit) addressing mode to the extended (64-bit) addressing mode or from the extended mode to a basic mode.

When mode tracing is on, a mode-switch trace entry is made in the trace table for each execution of the following instructions if the execution changes PSW bit 31:

- BRANCH AND SAVE AND SET MODE
- BRANCH AND SET MODE
- PROGRAM CALL

- PROGRAM RETURN
- RESUME PROGRAM
- SET ADDRESSING MODE

However, a mode-switch trace entry is not made for PROGRAM RETURN if ASN tracing is on and PROGRAM RETURN unstacks a state entry formed by PROGRAM CALL; a PROGRAM RETURN trace entry is made instead, and it contains information about PSW bit 31.

BRANCH AND SAVE AND SET MODE and RESUME PROGRAM cause trace entries to be made as follows: a branch trace entry if only branch tracing is on, a mode-switching-branch trace entry if both branch tracing and mode tracing are on, or a mode-switch trace entry if only mode tracing is on.

The trace entries produced by implicit tracing are summarized in Figure 4-5.

When explicit tracing is on, execution of TRACE (TRACE or TRACG) causes an entry to be made in the trace table. The entry for TRACE (TRACE) includes bits 16-63 from the TOD clock, the second operand of the TRACE instruction, and bits 32-63 of a range of general registers. The entry for TRACE (TRACG) is the same except that it includes bits 0-79 from the TOD clock and bits 0-63 of a range of general registers.

Instruction	Implicit Tracing Enabled						
	Branch	ASN	Mode	Branch and ASN	Branch and Mode	ASN and Mode	All
	Trace Entries Made						
BAKR	B	—	—	B	B	—	B
BALR	B	—	—	B	B	—	B
BASR	B	—	—	B	B	—	B
BASSM	B	—	MS	B	B MSB	MS	B MSB
BSA	B	—	—	B	B	—	B
BSG	B	BSG	—	BSG	B	BSG	BSG
BSM	—	—	MS	—	MS	—	MS
PC-20	—	PC	MS	PC	MS	PC & MS	PC & MS
PC-32	—	PC	MS	PC	MS	PC	PC
PR-b	—	—	MS	—	MS	MS	MS
PR-pc	—	PR	MS	PR	MS	PR	PR
PT or PTI	—	PT	—	PT	—	PT	PT
RP	B	—	MS	B	B MSB	MS	B MSB
SSAR or SSAIR	—	SSAR	—	SSAR	—	SSAR	SSAR
SAM24/31/64	—	—	MS	—	MS	MS	MS

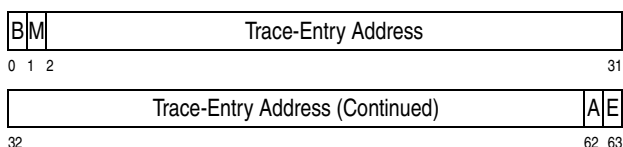
Figure 4-5. Summary of Implicit Tracing

Instruction	Implicit Tracing Enabled						
	Branch	ASN	Mode	Branch and ASN	Branch and Mode	ASN and Mode	All
	Trace Entries Made						
TRAP2/4	B	–	–	B	B	–	B
Explanation: – None. -20 The case when PROGRAM CALL uses a 20-bit PC number. -32 The case when PROGRAM CALL uses a 32-bit PC number. -b The case when PROGRAM RETURN unstacks a branch state entry. -pc The case when PROGRAM RETURN unstacks a program-call state entry. OR. & AND. B Branch trace entry. Made only if the branch is taken and a mode-switching-branch trace entry is not made. MS Mode-switch trace entry. Made only if PSW bit 31 is changed. MSB Mode-switching-branch trace entry. Made only if PSW bit 31 is changed (which can occur only if the branch is taken.							

Figure 4-5. Summary of Implicit Tracing

Control-Register Allocation

The information to control tracing is contained in control register 12 and has the following format:



Branch-Trace-Control Bit (B): Bit 0 of control register 12 controls whether branch tracing is turned on or off. If the bit is zero, branch tracing is off; if the bit is one, branch tracing is on.

Mode-Trace-Control Bit (M): Bit 1 of control register 12 controls whether mode tracing is turned on or off. If the bit is zero, mode tracing is off; if the bit is one, mode tracing is on.

Trace-Entry Address: Bits 2-61 of control register 12, with two zero bits appended on the left and two on the right, form the real address of the next trace entry to be made.

ASN-Trace-Control Bit (A): Bit 62 of control register 12 controls whether ASN tracing is turned on or off. If the bit is zero, ASN tracing is off; if the bit is one, ASN tracing is on.

Explicit-Trace-Control Bit (E): Bit 63 of control register 12 controls whether explicit tracing is turned

on or off. If the bit is zero, explicit tracing is off, which causes the TRACE instruction to be executed as a no-operation; if the bit is one, the execution of the TRACE instruction creates an entry in the trace table, except that no entry is made when bit 0 of the second operand of the TRACE instruction is one.

Trace Entries

Trace entries are of nine types, with most types having more than one detailed format. The types and numbers of formats are as follows:

- Branch (three formats)
- BRANCH IN SUBSPACE GROUP (two formats)
- Mode switch (three formats)
- Mode-switching branch (three formats)
- PROGRAM CALL (seven formats)
- PROGRAM RETURN (nine formats)
- PROGRAM TRANSFER (three formats)
- SET SECONDARY ASN (one format)
- TRACE (two formats)

Format-1 and format-2 PROGRAM CALL trace entries are made if the ASN-and-LX reuse facility is not enabled. Entries of formats 3-7 are made if the facility is enabled.

The PROGRAM TRANSFER trace entry is also made for PROGRAM TRANSFER WITH INSTANCE, and the SET SECONDARY ASN trace entry is also made for SET SECONDARY ASN WITH INSTANCE.

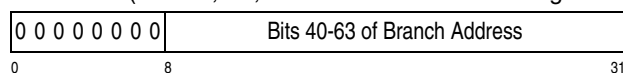
In either case, bit 15 (N) of the entry is one if the entry was made because of execution of the with-instance instruction.

monic of an instruction that causes a branch trace entry, refers to BAKR, BALR, BASR, BASSM, BSA, or BSG.

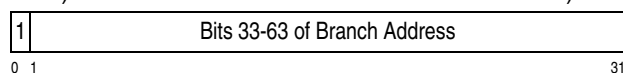
The entries are shown in Figure 4-6. In that figure, each entry is labeled with “Fn,” indicating a format number, to allow references to each format within a trace-entry type. Also, “Branch,” referring to the mne-

Figure 4-7 on page 4-19 lists the trace entries in ascending order of values in bit fields that identify the entries.

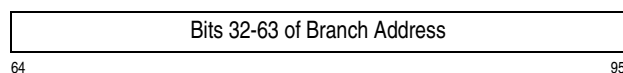
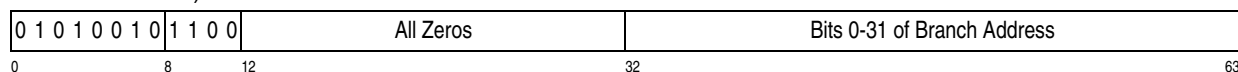
F1 Branch (Branch, RP, or TRAP2/4 when Resulting Mode Is 24-Bit)



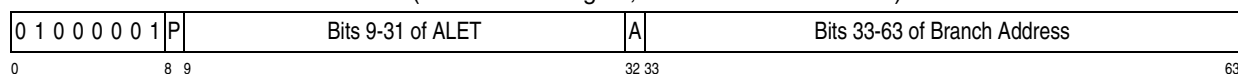
F2 Branch (Branch, RP, or TRAP2/4 when Resulting Mode Is 31-Bit, or when Resulting PSW Bit 31 is One (See Note) and Bits 0-32 of Branch Address Are All Zeros)



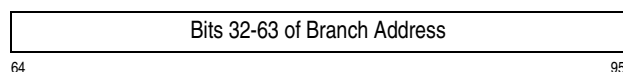
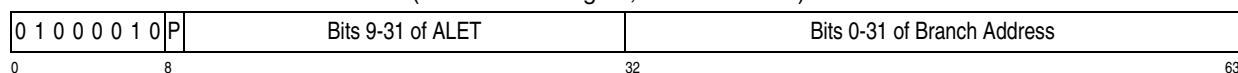
F3 Branch (Branch, RP, or TRAP2/4 when Resulting PSW Bit 31 Is One (See Note) and Bits 0-32 of Branch Address Are Not All Zeros)



F1 BRANCH IN SUBSPACE GROUP (if ASN Is Tracing on, in 24-Bit or 31-Bit Mode)



F2 BRANCH IN SUBSPACE GROUP (if ASN Is Tracing on, in 64-Bit Mode)



F1 Mode Switch (BASSM, BSM, PC, PR, RP, or SAM64 from 24-Bit or 31-Bit Mode when Resulting PSW Bit 31 is One (See Note))

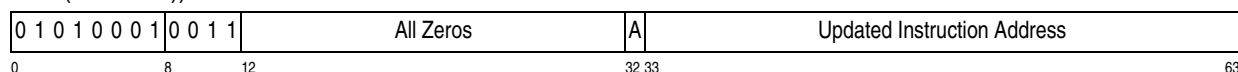


Figure 4-6. Trace Entries (Part 1 of 6)

0	1	0	1	0	0	0	1	0	0	1	0	All Zeros										Bits 32-63 of Updated Instruction Address																															
0								8		12		32										63																															

0	1	0	1	0	0	1	0	0	1	1	0	All Zeros																Bits 0-31 of Updated Instruction Address																																			
0								8				12				32																63																															

64 95

0	1	0	1	0	0	0	1	1	0	1	0	All Zeros										A	Branch Address																							
0								8				12								32 33		63																								

0	1	0	1	0	0	0	1	1	0	1	1	All Zeros																Bits 32-63 of Branch Address																			
0								8				12				32																63															

0	1	0	1	0	0	1	0	1	1	1	1	All Zeros										Bits 0-31 of Branch Address																																									
0								8				12				32																63																															

64 95

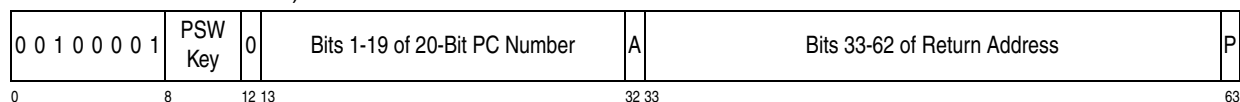
0 0 1 0 0 0 0 1	PSW Key	PC Number	A	Bits 33-62 of Return Address	P
0	8	12	32 33		63

0 0 1 0 0 0 1 0	PSW Key	PC Number	Bits 0-31 of Return Address
0	8	12	32

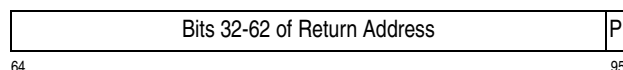
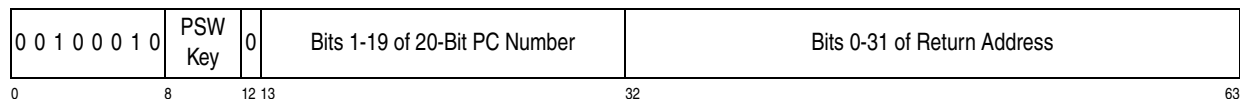
64 95

Figure 4-6. Trace Entries (Part 2 of 6)

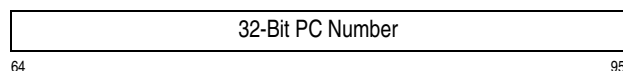
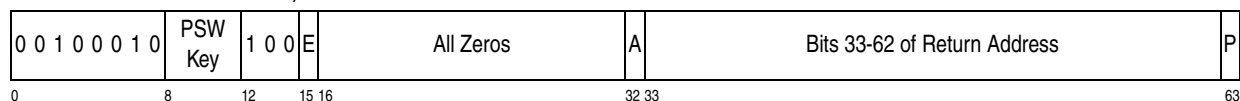
F3 PROGRAM CALL (in 24-Bit or 31-Bit Mode, Regardless of Resulting Mode, if ASN-and-LX Reuse Is Enabled and 20-Bit PC Number is Used)



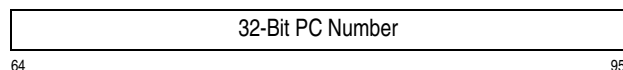
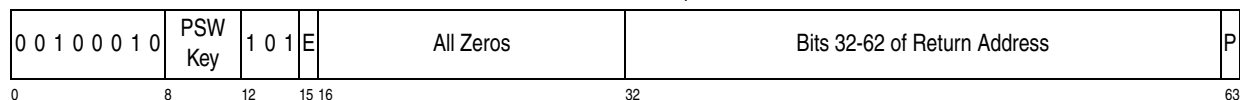
F4 PROGRAM CALL (in 64-Bit Mode, Regardless of Resulting Mode, if ASN-and-LX Reuse Is Enabled and 20-Bit PC Number Is Used)



F5 PROGRAM CALL (in 24-Bit or 31-Bit Mode, Regardless of Resulting Mode, if ASN-and-LX Reuse Is Enabled and 32-Bit PC Number is Used)



F6 PROGRAM CALL (in 64-Bit Mode, Regardless of Resulting Mode, if ASN-and-LX Reuse Is Enabled, 32-Bit PC Number Is Used, and Bits 0-31 of Return Address Are All Zeros)



F7 PROGRAM CALL (in 64-Bit Mode, Regardless of Resulting Mode, if ASN-and-LX Reuse Is Enabled, 32-Bit PC Number Is Used, and Bits 0-31 of Return Address Are Not All Zeros)

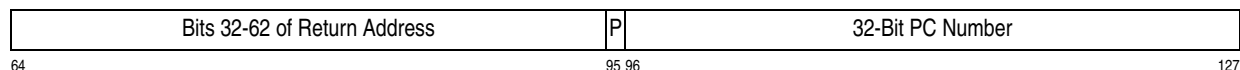
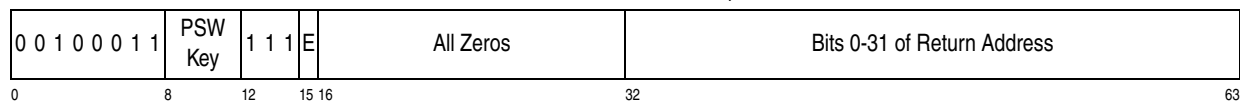
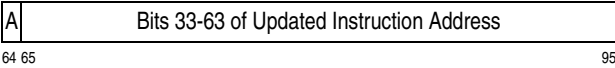
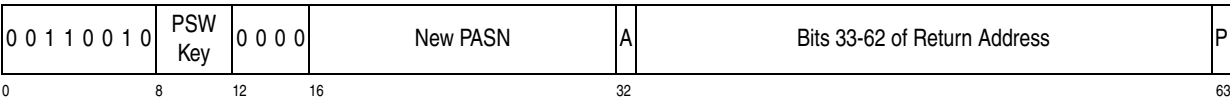
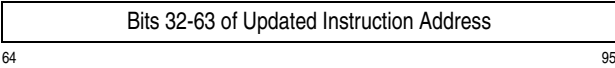
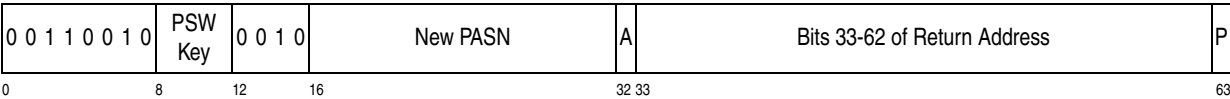


Figure 4-6. Trace Entries (Part 3 of 6)

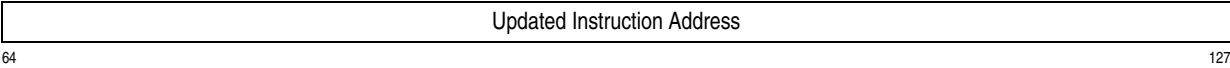
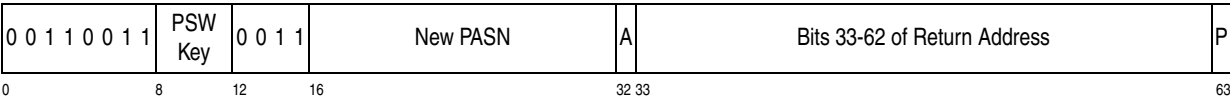
F1 PROGRAM RETURN (in 24-Bit or 31-Bit Mode when Resulting Mode Is 24-Bit or 31-Bit)



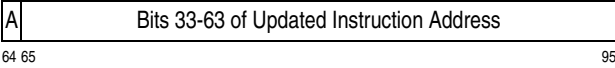
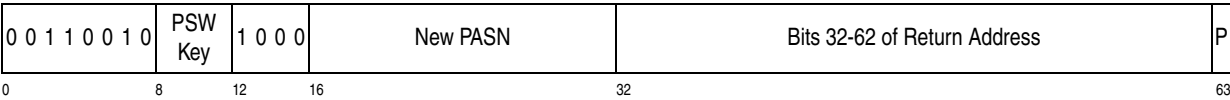
F2 PROGRAM RETURN (in 64-Bit Mode when Bits 0-31 of Updated Instruction Address Are All Zeros and Resulting Mode Is 24-Bit or 31-Bit)



F3 PROGRAM RETURN (in 64-Bit Mode when Bits 0-31 of Updated Instruction Address Are Not All Zeros and Resulting Mode Is 24-Bit or 31-Bit)



F4 PROGRAM RETURN (in 24-Bit or 31-Bit Mode when Resulting PSW Bit 31 Is One (See Note) and Bits 0-31 of Return Address Are All Zeros)



F5 PROGRAM RETURN (in 64-Bit Mode when Bits 0-31 of Updated Instruction Address Are All Zeros, Resulting PSW Bit 31 Is One (See Note), and Bits 0-31 of Return Address Are All Zeros)

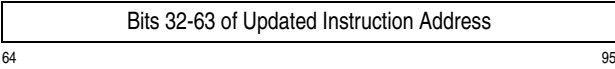
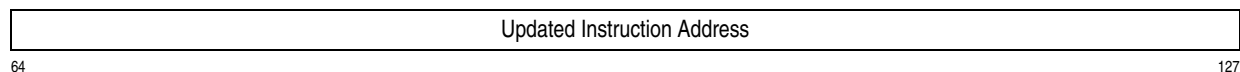
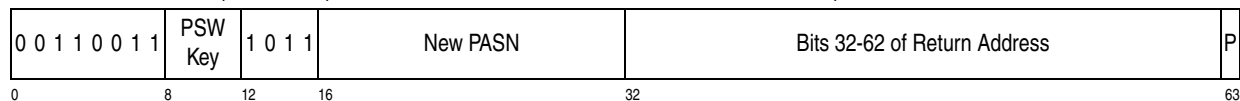
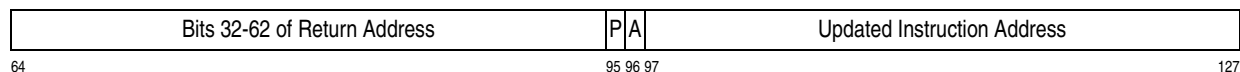
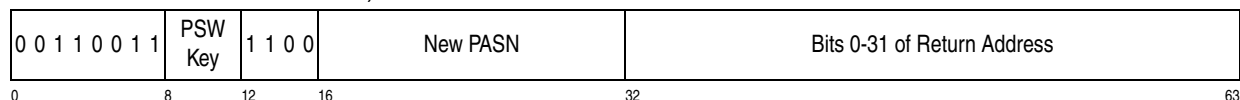


Figure 4-6. Trace Entries (Part 4 of 6)

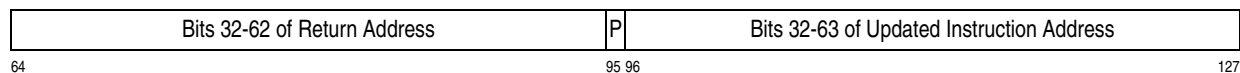
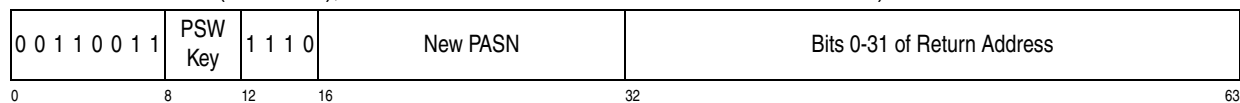
F6 PROGRAM RETURN (in 64-Bit Mode when Bits 0-31 of Updated Instruction Address Are Not All Zeros, Resulting PSW Bit 31 Is One (See Note), and Bits 0-31 of Return Address Are All Zeros)



F7 PROGRAM RETURN (in 24-Bit or 31-Bit Mode when Resulting PSW Bit 31 Is One (See Note) and Bits 0-31 of Return Address Are Not All Zeros)



F8 PROGRAM RETURN (in 64-Bit Mode when Bits 0-31 of Updated Instruction Address Are All Zeros, Resulting PSW Bit 31 Is One (See Note), and Bits 0-31 of Return Address Are Not All Zeros)



F9 PROGRAM RETURN (in 64-Bit Mode when Bits 0-31 of Updated Instruction Address Are Not All Zeros, Resulting PSW Bit 31 Is One (See Note), and Bits 0-31 of Return Address Are Not All Zeros)

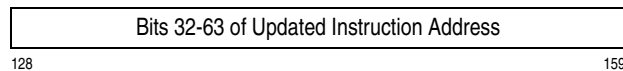
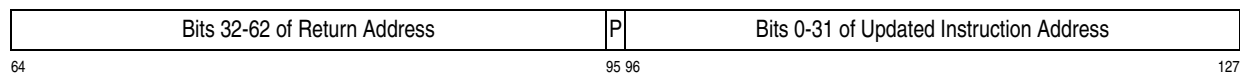
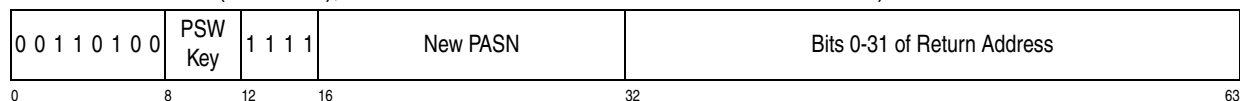
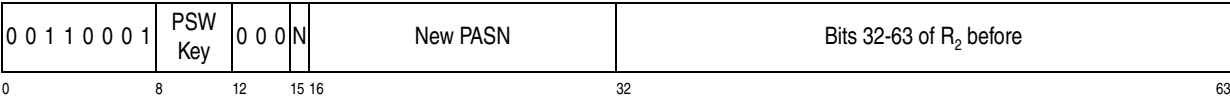
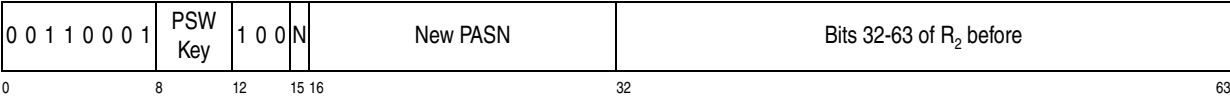


Figure 4-6. Trace Entries (Part 5 of 6)

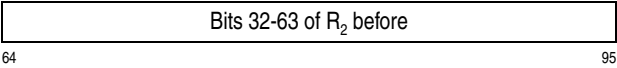
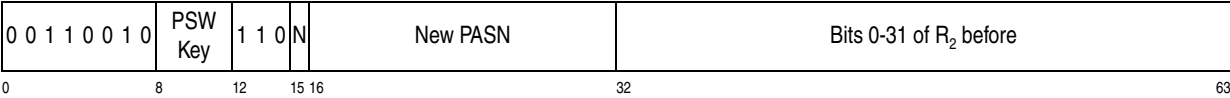
F1 PROGRAM TRANSFER (WITH INSTANCE if N Is One) (in 24-Bit or 31-Bit Mode)



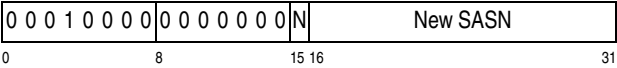
F2 PROGRAM TRANSFER (WITH INSTANCE if N Is One) (in 64-Bit Mode when Bits 0-31 of R₂ Are All Zeros)



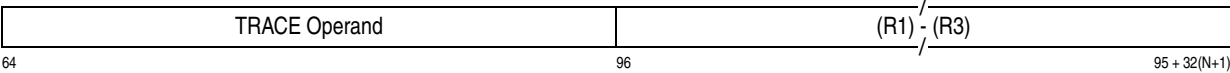
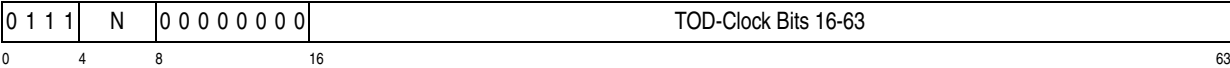
F3 PROGRAM TRANSFER (WITH INSTANCE if N Is One) (in 64-Bit Mode when Bits 0-31 of R₂ Are Not All Zeros)



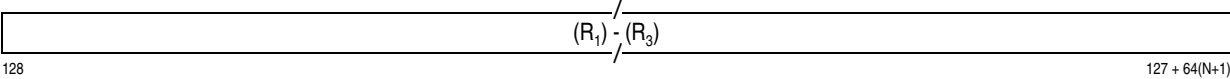
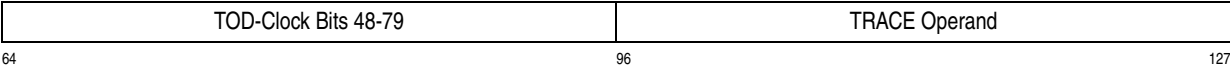
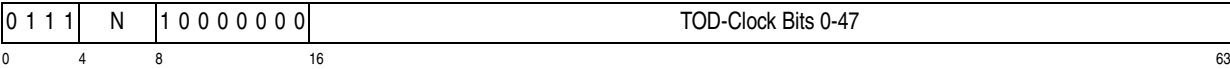
F1 SET SECONDARY ASN (WITH INSTANCE if N Is One)



F1 TRACE (TRACE)



F2 TRACE (TRACG)



Note: The terminology “when Resulting PSW Bit 31 Is One” is used instead of “when Resulting Mode Is 64-Bit” because, if the resulting PSW bit 32 is zero, an early specification exception will be recognized. PROGRAM RETURN can set PSW bit 31 to one and bit 32 to zero.

Figure 4-6. Trace Entries (Part 6 of 6)

Trace-Entry Bits			Trace Entry	
0-7	8-11	12-15	Type	Format
00000000			Branch	1
00010000		000N	SET SECONDARY ASN	1
00100001			PROGRAM CALL	1 ¹
00100010			PROGRAM CALL	2 ¹
00100001		0	PROGRAM CALL	3 ¹
00100010		0	PROGRAM CALL	4 ¹
00100010		100E	PROGRAM CALL	5 ¹
00100010		101E	PROGRAM CALL	6 ¹
00100011		111E	PROGRAM CALL	7 ¹
00110001		000N	PROGRAM TRANSFER	1
00110001		100N	PROGRAM TRANSFER	2
00110010		0000	PROGRAM RETURN	1
00110010		0010	PROGRAM RETURN	2
00110010		1000	PROGRAM RETURN	4
00110010		1010	PROGRAM RETURN	5
00110010		110N	PROGRAM TRANSFER	3
00110011		0011	PROGRAM RETURN	3
00110011		1011	PROGRAM RETURN	6
00110011		1100	PROGRAM RETURN	7
00110011		1110	PROGRAM RETURN	8
00110100		1111	PROGRAM RETURN	9
01000001			BRANCH IN SUBSPACE GROUP	1
01000010			BRANCH IN SUBSPACE GROUP	2
01010001	0010		Mode Switch	2
01010001	0011		Mode Switch	1
01010001	1010		Mode-Switching Branch	1
01010001	1011		Mode-Switching Branch	2
01010010	0110		Mode-Switch	3
01010010	1100		Branch	3
01010010	1111		Mode-Switching Branch	3
0111	0		TRACE	1
0111	1		TRACE	2
1			Branch	2
Explanation: 1 Format-1 and format-2 entries are made when the ASN-and-LX reuse facility is not enabled. Entries of formats 3-7 are made when the facility is enabled. E Indicates, when one, that the extended-addressing-mode bit, PSW bit 31, was set to one. N Indicates, when one, that a PROGRAM TRANSFER entry was made because of PROGRAM TRANSFER WITH INSTANCE or a SET SECONDARY ASN entry was made because of SET SECONDARY ASN WITH INSTANCE.				

Figure 4-7. Trace Entries Arranged by Identifying Bits

The fields in the trace entries are defined as follows. The fields are described in the order in which they first appear in Figure 4-6 on page 4-13.

Branch Address: The branch address is the address of the next instruction to be executed when the branch is taken. In a branch trace entry made when the 24-bit addressing mode is in effect after branching (a format-1 entry), bit positions 8-31 contain bits 40-63 of the branch address. When the 31-bit addressing mode is in effect after branching or PSW bit 31 is one after branching and bits 0-32 of the branch address are all zeros, bit positions 1-31 of the trace entry (format 2) contain bits 33-63 of the branch address. When PSW bit 31 is one after branching and bits 0-32 of the branch address are not all zeros, bit positions 32-95 of the trace entry (format 3), contain bits 0-63 of the branch address.

In a BRANCH IN SUBSPACE GROUP trace entry made on execution in the 24-bit or 31-bit addressing mode, bit positions 33-63 of the trace entry (format 1) contain bits 33-63 of the branch address, or, in the 64-bit addressing mode, bit positions 32-95 of the trace entry (format 2) contain bits 0-63 of the branch address.

In a mode-switching-branch trace entry made on a switch from the 64-bit addressing mode to the 24-bit or 31-bit addressing mode, bit positions 33-63 of the entry (format 1) contain bits 33-63 of the branch address; or, on a switch from PSW bit 31 being off to the bit being on, bit positions 32-63 of the entry (format 2) contain bits 32-63 of the branch address if bits 0-31 of the branch address are zeros, or bits 32-95 of the entry (format 3) contain bits 0-63 of the branch address if bits 0-31 of the branch address are not all zeros.

Primary-List Bit (P) and Bits 9-31 of ALET: Bit position 8 of a BRANCH IN SUBSPACE GROUP trace entry contains bit 7 of the access-list-entry token (ALET) in the access register designated by the R₂ field of the instruction. Bit positions 9-31 of the trace entry contain bits 9-31 of the ALET.

Basic-Addressing-Mode Bit (A): Bit position 32 of a BRANCH IN SUBSPACE GROUP trace entry made on execution in the 24-bit or 31-bit addressing mode (a format-1 entry) contains the basic-addressing-mode bit that replaces bit 32 of the PSW.

Bit position 32 of a mode-switch trace entry that indicates a switch from PSW bit 31 being off to the bit

being on (a format-1 entry) contains the value of PSW bit 32 that existed before the mode-switch operation.

Bit position 32 of a mode-switching-branch trace entry that indicates a switch from the 64-bit addressing mode to the 24-bit or 31-bit addressing mode (a format-1 entry) contains the value that replaces PSW bit 32.

Bit position 32 of a PROGRAM CALL trace entry made on execution in the 24-bit or 31-bit addressing mode (regardless of the resulting addressing mode) (a format-1 entry) contains the basic-addressing-mode bit, bit 32, from the current PSW.

Bit position 32 of a PROGRAM RETURN trace entry made when the resulting addressing mode is the 24-bit or 31-bit mode (a format-1, format-2, or format-3 entry) contains the basic-addressing-mode bit that replaces bit 32 of the PSW.

Bit position 64 of a PROGRAM RETURN trace entry made in the 24-bit or 31-bit addressing mode when the return address occupies only one word in the entry, (a format-1 or format-4 entry), contains the value of PSW bit 32 that existed before the PROGRAM RETURN operation. When the return address occupies two words (a format-7 entry), bit position 96 contains that value of PSW bit 32.

Updated Instruction Address: Bit positions 33-63 of a mode-switch trace entry that indicates a switch from PSW bit 31 being off to the bit being on (a format-1 entry) contains bits 33-63 of the updated instruction address in the PSW (bits 97-127 of the PSW) before that address is replaced, if it is replaced, by the mode-switch operation. Bit positions 32-63 of a mode-switch trace entry (format 2) that indicates a switch from the 64-bit addressing mode to the 24-bit or 31-bit addressing mode contains bits 32-63 of the updated instruction address in the PSW (bits 96-127 of the PSW) before that address is replaced, if it is replaced, by the mode-switch operation, if bits 0-31 of the updated instruction address are zeros; or bit positions 32-95 of the trace entry (format 3) contain bits 0-63 of that updated instruction address (bits 64-127 of the PSW) if bits 0-31 of the address are not all zeros.

The following description of a PROGRAM RETURN trace entry applies when the return address in the entry occupies only one word in the entry. Bit positions 65-95 of the trace entry made on execution in

the 24-bit or 31-bit addressing mode (a format-1 or format-4 entry) contain bits 33-63 of the updated instruction address in the PSW (bits 97-127 of the PSW) before that address is replaced from the linkage-stack state entry; or, when the execution is in the 64-bit addressing mode, bit positions 64-95 of the trace entry (format 2 or 5) contain bits 32-63 of that updated instruction address (bits 96-127 of the PSW) if bits 0-31 of the address are zeros, or bit positions 64-127 of the trace entry (format 3 or 6) contain bits 0-63 of that updated instruction address (bits 64-127 of the PSW) if bits 0-31 of the address are not all zeros. If the return address in the PROGRAM RETURN trace entry occupies two words, the updated instruction address in the entry is moved one word to the right in the entry (formats 7-9).

PSW Key: Bit positions 8-11 of a PROGRAM CALL, PROGRAM TRANSFER, or PROGRAM RETURN trace entry contain the PSW key from the current PSW.

PC Number: Bit positions 12-31 of a PROGRAM CALL trace entry of format 1-4 contain the value of the rightmost 20 bits of the second-operand address. Bit positions 64-95 of a format-5 or format-6 PROGRAM CALL trace entry, or bit positions 96-127 of a format-7 entry, contain the value of the rightmost 32 bits of the second-operand address.

Return Address: Bit positions 33-62 of a PROGRAM CALL trace entry made on execution in the 24-bit or 31-bit addressing mode (a format-1, format-3, or format-5 entry) contain bits 33-62 of the updated instruction address in the PSW (bits 97-126 of the PSW) before that address is replaced from the entry-table entry; or, when the execution is in the 64-bit addressing mode, bit positions 32-94 of the trace entry (format 2, 4, or 7) contain bits 0-62 of that updated instruction address (bits 64-126 of the PSW), or, when bits 0-31 of the address are all zeros, bit positions 32-62 of the trace entry (format 6) contain bits 32-62 of the address.

Extended-Addressing-Mode Bit (E): Bit position 15 of a PROGRAM CALL trace entry made using a 32-bit PC number (a format-5, format-6, or format-7 entry) contains the extended-addressing-mode bit that replaces bit 31 of the PSW.

Bit positions 33-62 of a PROGRAM RETURN trace entry made when the resulting addressing mode is the 24-bit or 31-bit mode (a format-1, format-2, or format-3 entry) contain bits 33-62 of the instruction

address that replaces bits 64-127 of the PSW; or, when the resulting PSW bit 31 is one (which causes the addressing mode be the 64-bit mode unless the resulting PSW bit 32 is zero), bit positions 32-62 of the trace entry (formats 4-6) contain bits 32-62 of that instruction address if bits 0-31 of the address are zeros, or bit positions 32-94 of the trace entry (formats 7-9) contain bits 0-62 of that instruction address if bits 0-31 of the address are not all zeros.

Problem-State Bit (P): Bit position 63 of a PROGRAM CALL trace entry made on execution in the 24-bit or 31-bit addressing mode (regardless of the resulting mode) (a format-1, format-3, format-5, or format-6 entry), or bit position 95 of the entry (format 2, 4, or 7) made on execution in the 64-bit addressing mode, contains the problem-state bit from the current PSW.

Bit position 63 of a PROGRAM RETURN trace entry made when the resulting addressing mode is the 24-bit or 31-bit mode (a format-1, format-2, or format-3 entry) or when the resulting PSW bit 31 is one and bits 0-31 of the return address are zeros (formats 4-6) contains the problem-state bit that replaces bit 15 of the PSW. Bit position 95 of a PROGRAM RETURN trace entry made when the resulting PSW bit 31 is one and bits 0-31 of the return address are not all zeros (formats 7-9) contains that problem-state bit.

New PASN: Bit positions 16-31 a PROGRAM TRANSFER trace entry contain the new PASN (which may be zero) specified in bit positions 48-63 of general register R_1 .

Bit positions 16-31 of a PROGRAM RETURN trace entry contain the new PASN that is restored from the linkage-stack state entry.

Bits 32-63 of R_2 before: Bit positions 32-63 of a PROGRAM TRANSFER trace entry made on execution in the 24-bit or 31-bit addressing mode (a format-1 entry) contain bits 32-63 of the general register designated by the R_2 field of the instruction. (Bits 32 and 33-62 of that register replace bits 32 and 97-126, respectively, of the PSW. Bit 63 of the register replaces the problem-state bit in the PSW.) When PROGRAM TRANSFER or PROGRAM TRANSFER WITH INSTANCE is executed in the 64-bit addressing mode, bit positions 32-63 of the trace entry (format 2) contain bits 32-63 of the R_2 general register if bits 0-31 of the register are zeros, or bit positions 32-95 of the trace entry (format 3) contain bits 0-63

of the register if bits 0-31 of the register are not all zeros.

New SASN: Bit positions 16-31 of a SET SECONDARY ASN trace entry contain the ASN value loaded into control register 3 by the instruction.

Number of Registers (N): Bits 4-7 of the trace entry for TRACE contain a value which is one less than the number of general registers which have been provided in the trace entry. The value of N ranges from zero, meaning the contents of one general register are provided in the trace entry, to 15, meaning the contents of all 16 general registers are provided.

TOD-Clock Bits 16-63 or 0-79: When the store-clock-fast facility is not installed, or when the TRACE TOD-clock control in bit 32 of control register 0 is zero, bits 16-63 of the trace entry for TRACE (TRACE) are obtained from bit positions 16-63 of the TOD clock, as would be provided by a STORE CLOCK instruction executed at the time the TRACE instruction was executed. When the store-clock-fast facility is installed and the TRACE TOD-clock control in bit 32 of control register zero is one, bits 16-63 of the trace entry for TRACE (TRACE) are obtained from bit positions 16-63 of the TOD clock, as would be provided by a STORE CLOCK FAST instruction executed at the time the TRACE instruction was executed.

Bits 16-95 of the trace entry for TRACE (TRACG) are obtained from bit positions 0-79 of the TOD clock, as would be provided by a STORE CLOCK EXTENDED instruction executed at the time the TRACE instruction was executed. See programming note 2 for information about a carry from bit position 0 of the TOD clock.

TRACE Operand: When the store-clock-fast facility is not installed, or when the TRACE TOD-clock control in bit 32 of control register 0 is zero, bit positions 64-95 of the trace entry for TRACE (TRACE) and bit positions 96-127 of the trace entry for TRACE (TRACG) contain a copy of the 32 bits of the second operand of the TRACE instruction for which the entry is made.

When the store-clock-fast facility is installed and the TRACE TOD-clock control in bit 32 of control register 0 is one, the trace-operand field in the trace entry is formed as follows.

- Bit positions 64-71 and 80-95 of the trace entry for TRACE (TRACE) and bit positions 96-103 and 112-127 of the trace entry for TRACE (TRACG) contain a copy of bits 0-7 and 16-31, respectively, of the second operand of the TRACE instruction for which the entry is made.
- The contents of bit positions 72-79 of the trace entry for TRACE (TRACE) and the contents of bit positions 104-111 for TRACE (TRACG) are set to a model-dependent value.

(R₁)-(R₃): The four-byte fields starting with bit 96 of the trace entry for TRACE (TRACE) contain the contents of bit positions 32-63 of the general registers whose range is specified by the R₁ and R₃ fields of the TRACE instruction. The general registers are stored in ascending order of register numbers, starting with general register R₁ and continuing up to and including general register R₃, with general register 0 following general register 15. The eight-byte fields starting with bit 128 of the trace entry for TRACE (TRACG) similarly contain the contents of bit positions 0-63 of those registers.

Programming Notes:

1. The size of the trace entry for TRACE (TRACE) in units of words is $3 + (N + 1)$. The maximum size of an entry is 19 words, or 76 bytes. For TRACE (TRACG), the size in units of words is $4 + 2(N + 1)$, and the maximum size is 36 words, or 144 bytes.
2. At some time in the future, the TOD clock on new models will have a leftmost extension so that there can be a carry from bit position 0 of the clock into the extension; see programming note 14 on page 4-43. On these models, the rightmost bit of the extension will be stored in bit position 15 of the TRACE (TRACG) trace entry. It may be desired to have programs that process TRACE (TRACG) trace entries take this future development into account.

Operation

When an instruction which is subject to tracing is executed and the corresponding tracing function is turned on, a trace entry of the appropriate type and format is made. The real address of the trace entry is formed by appending two zero bits on the left and two on the right to the value in bit positions 2-61 of con-

trol register 12. The address in control register 12 is subsequently increased by the size of the entry created.

No trace entry is stored if the incrementing of the address in control register 12 would cause a carry to be propagated into bit position 51 (that is, if the trace-entry address would be in the next 4K-byte block). If this would be the case for the entry to be made, a trace-table exception is recognized and the operation is nullified. When PROGRAM CALL is to form both a PROGRAM CALL trace entry and a mode-switch trace entry, neither entry is stored, and a trace-table exception is recognized, if either entry would cause a carry into bit position 51. For the purpose of recognizing the trace-table exception in the case of a TRACE instruction, the maximum length of 76 (TRACE) or 144 (TRACG) bytes is used instead of the actual length.

The storing of a trace entry is not subject to key-controlled protection (nor, since the trace-entry address is real, is it subject to DAT protection), but it is subject to low-address protection; that is, if the address of the trace entry due to be created is in the range 0-511 or 4096-4607 and bit 35 of control register 0 is one, a protection exception is recognized, and instruction execution is suppressed. If the address of a trace entry is invalid, an addressing exception is recognized, and instruction execution is suppressed.

The three exceptions associated with storing a trace entry (addressing, protection, and trace table) are collectively referred to as trace exceptions.

If a program interruption takes place for a condition which is not a trace-exception condition and for which execution of an instruction is not completed, it is unpredictable whether part or all of any trace entry due to be made for such an interrupted instruction is stored in the trace table. Thus, for a condition which would ordinarily cause nullification or suppression of instruction execution, storage locations may have been altered beginning at the location designated by control register 12 and extending up to the length of the entry that would have been created.

When PROGRAM RETURN unstacks a linkage-stack state entry that was formed by BRANCH AND STACK and ASN tracing is on, trace exceptions may be recognized, even though a trace entry is not made and no part of a trace entry is stored.

The order in which information is placed in a trace entry is unpredictable. Furthermore, as observed by other CPUs and by channel programs, the contents of a byte of a trace entry may appear to change more than once before completion of the instruction for which the entry is made.

The trace-entry address in control register 12 is updated only on completion of execution of an instruction for which a trace entry is made.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed. However, it is unpredictable whether or not a store into a traceable entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store. Additionally, when the store-clock-fast facility is installed and the TRACE TOD-clock control in bit 32 of control register 0 is one, it is unpredictable whether explicit tracing causes serialization to be performed.

Program-Event Recording

The purpose of PER is to assist in debugging programs. It permits the program to be alerted to the following types of events:

- Execution of a successful branch instruction. The option is provided of having an event occur only when the branch-target location is within the designated storage area.
- Fetching of an instruction from the designated storage area.
- Alteration of the contents of the designated storage area. The option is provided of having an event occur only when the storage area is within designated address spaces.
- Execution of the STORE USING REAL ADDRESS instruction.

The program can selectively specify that one or more of the above types of events be recognized, except that the event for STORE USING REAL ADDRESS can be specified only along with the storage-alteration event. The information concerning a PER event is provided to the program by means of a program interruption, with the cause of the interruption being identified in the interruption code.

PER Instruction-Fetching Nullification

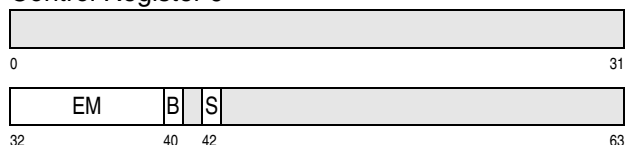
The PER-3 facility may be available on a model implementing z/Architecture and is available only in the z/Architecture architectural mode. When this facility is installed, bit 39 of control register 9, when one, specifies that PER instruction-fetching events force nullification. Bit 39 is effective for this purpose only when bit 33 of control register 9, the instruction-fetching PER-event mask bit, is also one. When bit 33 is zero, PER instruction-fetching events are not recognized, and bit 39 has no effect. When the PER-3 facility is not installed or bit 39 is zero, PER instruction-fetching events do not force nullification. A PER instruction-fetching event that forces nullification is referred to as a PER instruction-fetching nullification event. A PER event that does not force nullification is referred to as a PER basic event.

When the PER-3 facility is installed, and bit 39 is one, the interruption caused by a PER instruction-fetching event occurs before the fetched instruction is executed, the PER instruction-fetching nullification event is indicated, no other PER events and no other program interruption conditions are reported, and execution of the instruction is nullified. When the PER-3 facility is not installed, or bit 39 is zero, nullification is not forced, the PER instruction-fetching basic event is indicated, other PER events and other program interruption conditions may be concurrently reported, and the execution of the instruction may be completed, terminated, suppressed, or nullified. In the absence of other conditions, the interruption caused by the PER instruction-fetching basic event occurs after execution of the fetched instruction, or units of operation thereof, are completed.

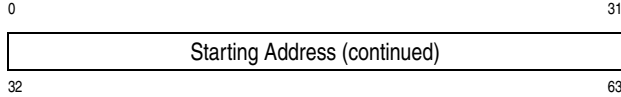
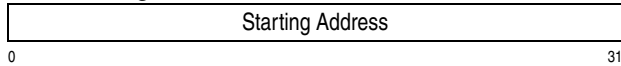
Control-Register Allocation and Address-Space-Control Element

The information for controlling PER resides in control registers 9, 10, and 11 and the address-space-control element. The information in the control registers has the following format:

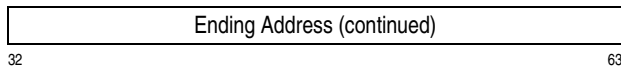
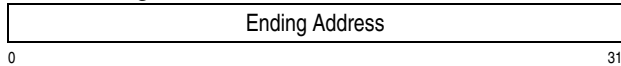
Control Register 9



Control Register 10



Control Register 11



PER-Event Masks (EM): Bits 32-34 and 36 specify which types of events are recognized. When the PER-3 facility is installed, bit 39 of the PER-event masks is also used. The bits are assigned as follows:

- Bit 32: Successful-branching event
- Bit 33: Instruction-fetching event
- Bit 34: Storage-alteration event
- Bit 36: Store-using-real-address event (bit 34 must be one also)
- Bit 39: Instruction-fetching nullification event (bit 33 must also be one)

Bits 32-34 and bit 36, when ones, specify that the corresponding types of events be recognized. However, bit 36 is effective for this purpose only when bit 34 is also one. When bit 34 is one, the storage-alteration event is recognized. When bits 34 and 36 are ones, both the storage-alteration event and the store-using-real-address event are recognized. When a bit is zero, the corresponding type of event is not recognized. When bit 34 is zero, both the storage-alteration event and the store-using-real-address event are not recognized.

When the PER-3 facility is not installed, bit 39 is ignored. Bit 39 is effective only when bit 33 is also one. When bit 33 is one, and the PER-3 facility is installed, and bit 39 is one, the PER instruction-fetching nullification event is recognized. When bit 33 is one and bit 39 is zero (or the PER-3 facility is not installed) the PER instruction-fetching basic event is recognized. When bit 33 is zero, neither the PER instruction-fetching basic event nor the PER instruction-fetching nullification event is recognized.

Branch-Address Control (B): Bit 40 of control register 9 specifies, when one, that successful-branching events occur only for branches that are to a location within the designated storage area. When

bit 40 is zero, successful branching events occur regardless of the branch-target address.

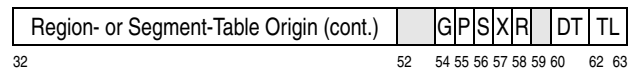
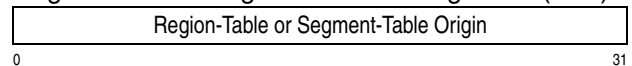
Storage-Alteration-Space Control (S): Bit 42 of control register 9 specifies, when one, that storage-alteration events occur as a result of references to the designated storage area only within designated address spaces. An address space is designated as one for which storage-alteration events occur by means of the storage-alteration-event bit in the address-space-control element that is used to translate references to the address space. Bit 42 is ignored when DAT is not in effect. When DAT is not in effect or bit 42 is zero, storage-alteration events are not restricted to occurring for only particular address spaces.

PER Starting Address: Bits 0-63 of control register 10 are the address of the beginning of the designated storage area.

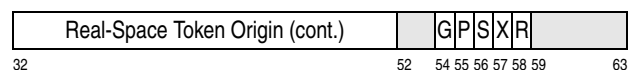
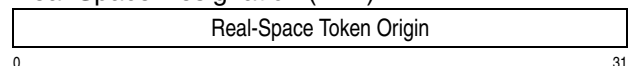
PER Ending Address: Bits 0-63 of control register 11 are the address of the end of the designated storage area.

The address-space-control element has one of the following formats:

Region-Table or Segment-Table Designation (R=0)



Real-Space Designation (R=1)



Storage-Alteration-Event Bit (S): When the storage-alteration-space control in control register 9 is one, bit 56 of the address-space control element specifies, when one, that the address space defined by the address-space-control element is one for which storage-alteration events can occur. Bit 56 is examined when the address-space-control element is used to perform dynamic-address translation for a storage-operand store reference. The address-space-control element may be the PASCE, SASCE, or HASCE in control register 1, 7, or 13, respectively, or it may be obtained from an ASN-second-table entry during access-register translation. Instead of

being obtained from an ASN-second-table entry in main storage, bit 56 may be obtained from an ASN-second-table entry in the ART-lookaside buffer (ALB). Bit 56 is ignored when the storage-alteration-space control is zero.

Programming Notes:

1. Models may operate at reduced performance while the CPU is enabled for PER events. In order to ensure that CPU performance is not degraded because of the operation of the PER facility, programs that do not use it should disable the CPU for PER events by setting either the PER mask in the PSW to zero or the PER-event masks in control register 9 to zero, or both. No degradation due to PER occurs when either of these fields is zero.
2. Some degradation may be experienced on some models every time control registers 9, 10, and 11 are loaded, even when the CPU is disabled for PER events (see the programming note under "Storage-Area Designation").
3. Enabling the CPU for PER instruction-fetching nullification may be used to determine the state of the CPU before execution of any instruction within the storage area designated by control registers 10 and 11. The instruction nullified could be the first instruction after a successful branch, after LOAD PSW, or after LOAD PSW EXTENDED; or it could be the target of an execute-type instruction or the leftmost instruction in the storage area and accessed in the process of sequential execution. After recording the desired information, in order to allow the CPU to execute this instruction, either the CPU must be disabled for instruction-fetching nullification or control registers 10 and 11 must be changed to designate a different storage area. This can be contrasted to enabling for PER successful branching within the same storage area, which causes a PER event to be reported only in the first case mentioned above, but does not require special action to continue.

Operation

PER is under control of bit 1 of the PSW, the PER mask. When the PER mask and a particular PER-event mask bit are all ones, the CPU is enabled for the corresponding type of event; otherwise, it is disabled. However, the CPU is enabled for the store-

using-real-address event only when the storage-alteration mask bit and the store-using-real-address mask bit are both one.

The CPU is enabled for the PER instruction-fetching nullification event only when the PER-3 facility is installed, and then only when the instruction-fetching-event mask bit, the instruction-fetching-nullification-event mask bit, and the PER mask are all ones. An interruption due to a PER instruction-fetching nullification event causes the execution of the instruction causing the event to be nullified.

An interruption due to a PER basic event normally occurs after the execution of the instruction responsible for the event. The occurrence of the event does not affect the execution of the instruction, which may be completed, partially completed, terminated, suppressed, or nullified. However, recognition of a storage-alteration event causes no more than 4K bytes to be stored beginning with the byte that caused the event, and this may result in partial completion of an interruptible instruction.

An interruption for an instruction-fetching nullification event occurs before the instruction responsible for the event is executed, and the operation is nullified.

When the CPU is disabled for a particular PER event at the time it occurs, either by the PER mask in the PSW or by the masks in control register 9, the event is not recognized.

A change to the PER mask in the PSW or to the PER control fields in control registers 9, 10, and 11 affects PER starting with the execution of the immediately following instruction. Thus, if, as a result of the change, an instruction-fetching nullification event applies to the immediately following instruction, execution of that instruction will be nullified and the instruction-fetching nullification event reported.

A change to the storage-alteration-event bit in an address-space-control element in control register 1, 7, or 13 also affects PER starting with the execution of the immediately following instruction. A change to the storage-alteration-event bit in an address-space-control element that may be obtained, during access-register translation, from an ASN-second-table entry in either main storage or the ALB does not necessarily have an immediate, if any, effect on PER. However, PER is affected immediately after either PURGE ALB or COMPARE AND SWAP AND PURGE that purges the ALB is executed.

If a PER basic event occurs during the execution of an instruction which changes the CPU from being enabled to being disabled for that type of event, that PER event is recognized.

PER basic events may be recognized in a trial execution of an instruction, and subsequently the instruction, DAT-table entries, and operands may be refetched for the actual execution. If any refetched field was modified by another CPU or by a channel program between the trial execution and the actual execution, it is unpredictable whether the PER events indicated are for the trial or the actual execution.

For special-purpose instructions that are not described in this publication, the operation of PER may not be exactly as described in this section.

Identification of Cause

A program interruption for PER sets bit 8 of the interruption code to one and places identifying information in real storage locations 150-159. When the PER event is a storage-alteration event, information is also stored in location 161. Additional information is provided by means of the instruction address in the program old PSW and the ILC.

Locations 150-151:

PER Code	ATMID	AI
0	8	14 15

PER Code: The occurrence of PER events is indicated by ones in bit positions 0-2, 4 and 7. The bit position in the PER code for a particular type of event is as follows:

Bit	PER Event
0	Successful-branching
1	Instruction-fetching
2	Storage-alteration
4	Store-using-real-address
7	Instruction-fetching nullification (PER-3)

A one in bit position 2 and a zero in bit position 4 of location 150 indicate a storage-alteration event, while ones in bit positions 2 and 4 indicate a store-using-real-address event. When a program interruption occurs, more than one type of PER basic event can be concurrently indicated. Additionally, if another program-interruption condition exists, the interruption code for the program interruption may indicate both the PER basic events and the other condition.

When a program interruption occurs for a PER instruction-fetching nullification event, bits 1 and 7 are set to one in the PER code. No other PER events are concurrently indicated.

Zeros are stored in bit positions 3 and 6 of locations 150-151. When PER-3 is not installed, zero is stored in bit position 7.

Addressing-and-Translation-Mode Identification (ATMID): During a program interruption when a PER event is indicated, bits 31, 32, 5, 16, and 17 of the PSW at the beginning of the execution of the instruction that caused the event may be stored in bit positions 8 and 10-13, respectively, of real locations 150-151. If bits 31, 32, 5, 16, and 17 are stored, then a one bit is stored in bit position 9 of locations 150-151. If bits 31, 32, 5, 16, and 17 are not stored, then zero bits are stored in bit positions 8-13 of locations 150-151.

Bits 8-13 of real locations 150-151 are named the addressing-and-translation-mode identification (ATMID). Bit 9 is named the ATMID-validity bit. When bit 9 is zero, it indicates that an invalid ATMID (all zeros) was stored.

The meanings of the bits of a valid ATMID are as follows:

Bit	Meaning
8	PSW bit 31
9	ATMID-validity bit
10	PSW bit 32
11	PSW bit 5
12	PSW bit 16
13	PSW bit 17

A valid ATMID is necessarily stored only if the PER event was caused by one of the following instructions:

- BRANCH AND SAVE AND SET MODE (BASSM)
- BRANCH AND SET AUTHORITY (BSA)
- BRANCH AND SET MODE (BSM)
- BRANCH IN SUBSPACE GROUP (BSG)
- LOAD PSW (LPSW)
- LOAD PSW EXTENDED (LPSWE)
- PROGRAM CALL (PC)
- PROGRAM RETURN (PR)
- PROGRAM TRANSFER (PT)
- PROGRAM TRANSFER WITH INSTANCE (PTI)
- RESUME PROGRAM (RP)

- SET ADDRESS SPACE CONTROL (SAC)
- SET ADDRESS SPACE CONTROL FAST (SACF)
- SET ADDRESSING MODE (SAM24, SAM31, SAM64)
- SET SYSTEM MASK (SSM)
- STORE THEN AND SYSTEM MASK (STNSM)
- STORE THEN OR SYSTEM MASK (STOSM)
- SUPERVISOR CALL (SVC)
- TRAP (TRAP2, TRAP4)

It is unpredictable whether a valid AT MID is stored if the PER event was caused by any other instruction. The value of the PER instruction-fetching-nullification-event mask bit does not affect the contents of the AT MID field.

PER ASCE Identification (AI): If the PER code contains an indication of a storage-alteration event (bit 2 is one and bit 4 is zero), and the event occurred when DAT was on, bits 14 and 15 of locations 150-151 are set to identify the address-space-control element (ASCE) that was used to translate the reference that caused the event, as follows:

Bits

14-15 Meaning

- | | |
|----|--|
| 00 | Primary ASCE was used. |
| 01 | An AR-specified ASCE was used. The PER access ID, real location 161, can be examined to determine the ASCE used. Even when an AR-specified ASCE is used, if the contents of the AR designate the primary, secondary, or home ASCE, bits 14 and 15 may be set to 00, 10, or 11, respectively, instead of to 01. |
| 10 | Secondary ASCE was used. |
| 11 | Home ASCE was used. |

The CPU may avoid setting bits 14 and 15 to 01 by recognizing that access-list-entry token (ALET) 00000000 or 00000001 hex was used or that the ALET designated, through an access-list entry, an ASN-second-table entry containing an ASCE equal to the primary ASCE, secondary ASCE, or home ASCE.

If the PER storage-alteration event occurred as a result of STORE HALFWORD RELATIVE LONG or STORE RELATIVE LONG, bits 14 and 15 correspond to the ASCE used to fetch the instruction: 00 when the CPU was in the primary-space mode, the secondary-space mode, or the access-register mode, and 11 when the CPU was in the home-space mode.

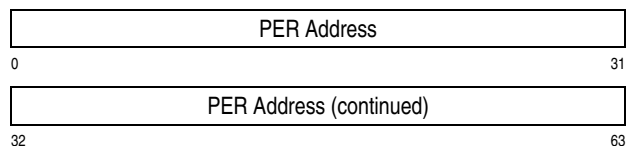
If any of the following are true, zeros are stored in bit positions 14 and 15 of locations 150-151.

- An ASCE was not used to translate the reference that caused the event.
- The PER code indicates that neither a storage-alteration event, nor a store-using-real-address event occurred (that is, bit 2 of the code is zero).

Programming Note: An ASCE is not used to translate the reference when DAT is off, or when the operand is defined to contain a real address (as is the case for INVALIDATE DAT TABLE ENTRY, INVALIDATE PAGE TABLE ENTRY, LOAD USING REAL ADDRESS, and STORE USING REAL ADDRESS). An ASCE is also not used to translate the reference when the operand is defined to contain a real or absolute address (as is the case for PERFORM FRAME MANAGEMENT FUNCTION).

PER Address: The PER-address field at locations 152-159 contains the instruction address used to fetch the instruction responsible for the recognized PER event or events, as shown below.

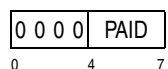
Locations 152-159:



When the instruction is the target of an execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG), the instruction address used to fetch the execute-type instruction is placed in the PER-address field.

PER Access Identification (PAID): If a storage-alteration event is indicated in the PER code, and the PER ASCE identification (AI, bits 14-15 of locations 150-151) contains 01 binary, an indication of the address space to which the event applies is stored at location 161, as shown below.

Location 161:



The number of the access register used is stored in bit positions 4-7 of location 161, and zeros are stored in bit positions 0-3. The contents of location 161 are

unpredictable if when the PER ASCE identification does not contain 01 binary.

Instruction Address: The instruction address in the program old PSW is the address of the instruction which would have been executed next, unless another program condition is also indicated, in which case the instruction address is that determined by the instruction ending due to that condition. When a PER instruction-fetching nullification event is recognized, the instruction address in the program old PSW is the address of the instruction responsible for the event. This is the same address stored in the PER address field in real storage locations 152-159.

ILC: For PER instruction nullification events, the ILC is 0. For PER basic events, the ILC indicates the length of the instruction designated by the PER address, except when a concurrent specification exception for the PSW introduced by LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, or a supervisor-call interruption sets an ILC of 0.

Programming Notes:

1. PSW bit 31 is the extended-addressing-mode bit, and PSW bit 32 is the basic-addressing-mode bit. When PSW bit 31 and 32 are both one, they specify the 64-bit addressing mode. When PSW bit 31 is zero, PSW 32 specifies the 24-bit addressing mode if the bit is zero or the 31-bit addressing mode if the bit is one. PSW bit 5 is the DAT-mode bit, and PSW bits 16 and 17 are the address-space-control bits. For the handling of instruction and logical addresses in the different translation modes, see "Translation Modes" on page 3-36.
2. A valid ATMID allows the program handling the PER event to determine the address space from which the instruction that caused the event was fetched and also to determine which translation mode applied to the storage-operand references of the instruction, if any. Each of the instructions for which a valid ATMID is necessarily stored can change one or more of PSW bits 5, 16, and 17, with the result that the values of those bits in the program old PSW that is stored because of the PER event are not necessarily the values that existed at the beginning of the execution of the instruction that caused the event. The instructions for which a valid ATMID is necessarily stored are the only instructions that can change any of PSW bits 5, 16, and 17.
3. If a storage-alteration PER event is indicated and DAT was on when the event occurred, an indication of the address-space-control element that was used to translate the reference that caused the event is given by the PER ASCE identification, bits 14 and 15 of real locations 150-151. If bits 14 and 15 indicate that an AR-specified address-space-control element was used, the PER access identification in real location 161 can be used to determine the address space that was referenced. To determine if DAT was on, the program handling the PER event should first examine the ATMID-validity bit to determine whether a valid ATMID was stored and, if it was stored, then examine the DAT-mode bit in the ATMID. If a valid ATMID was not stored, the program should examine the DAT-mode bit in the program old PSW.
4. If a valid ATMID is stored, it also allows the program handling the PER event to determine the addressing mode (24-bit, 31-bit, or 64-bit) that existed for the instruction that caused the PER event. This knowledge of the addressing mode allows the program to determine, without any chance of error, the meaning of one bits in bit positions 0-39 of the addresses of the instruction and of the storage operands, if any, of the instruction and, thus, to determine accurately the locations of the instruction and operands. Note that the address of the instruction is not necessarily provided without error by the PER address in real locations 152-159 because that address may be the address of an execute-type instruction, with the address of the target instruction still to be determined from the fields that specify the second-operand address of the execute-type instruction. Also note that another possible source of error is that, in the 24-bit or 31-bit addressing mode, an instruction or operand may wrap around in storage by beginning just below the 16M-byte or 2G-byte boundary, respectively.
5. A valid ATMID is necessarily stored for all instructions that can change the addressing-mode bits. However, the ATMID mechanism does not provide complete assurance that the instruction causing a PER event and the instruction's operands can be located accurately because LOAD CONTROL and LOAD ADDRESS SPACE PARAMETERS can change the address-space-control element that was used to fetch the instruction.

Priority of Indication

When a PER instruction-fetching nullification event is recognized and other program interruption conditions exist, only the program interruption condition with the highest priority is indicated. See “Multiple Program-Interruption Conditions” on page 6-44 for a description of the priority of program interruption conditions.

When a PER instruction-fetching nullification event is indicated, no other PER events are indicated. When a PER instruction-fetching nullification event is not indicated, then more than one PER basic event may be recognized and reported. The remainder of this section applies to these cases.

When a program interruption occurs and more than one PER basic event has been recognized, all recognized PER events are concurrently indicated in the PER code. Additionally, if another program-interruption condition concurrently exists, the interruption code for the program interruption indicates both the PER condition and the other condition.

In the case of an instruction-fetching basic event for SUPERVISOR CALL, the program interruption occurs immediately after the supervisor-call interruption.

If a PER basic event is recognized during the execution of an instruction which also introduces a new PSW with the type of PSW-format error which is recognized early (see “Exceptions Associated with the PSW” on page 6-8), both the specification exception and PER are indicated concurrently in the interruption code of the program interruption. If the PSW-format error is of the type which is recognized late, only PER is indicated in the interruption code. In both cases, the invalid PSW is stored as the program old PSW.

Recognition of a PER basic event does not normally affect the ending of instruction execution. However, in the following cases, execution of an interruptible instruction is not completed normally:

1. When the instruction is due to be interrupted for an asynchronous condition (I/O, external, restart, or repressible machine-check condition), a program interruption for the PER event occurs first, and the other interruptions occur subsequently (subject to the mask bits in the new PSW) in the normal priority order.

2. When the stop function is performed, a program interruption indicating the PER event occurs before the CPU enters the stopped state.
3. When any program exception is recognized, PER events recognized for that instruction execution are indicated concurrently.
4. Depending on the model, in certain situations, recognition of a PER event may appear to cause the instruction to be interrupted prematurely without concurrent indication of a program exception, without an interruption for any asynchronous condition, and without the CPU entering the stopped state. In particular, recognition of a storage-alteration event causes no more than 4K bytes to be stored beginning with the byte that caused the event.

In cases 1 and 2 above, if the only PER event that has been recognized is an instruction-fetching basic event and another unit of operation of the instruction remains to be executed, the event may be discarded, with the result that a program interruption does not occur. Whether the event is discarded is unpredictable.

Recognition of a PER instruction-fetching nullification event causes execution of the instruction responsible for the event to be nullified.

Programming Notes:

1. In the following cases, an instruction can both cause a program interruption for a PER basic event and change the value of fields controlling an interruption for PER events. The original field values determine whether a program interruption takes place for the PER event.
 - a. The instructions LOAD PSW, LOAD PSW EXTENDED, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and SUPERVISOR CALL can cause an instruction-fetching event and disable the CPU for PER interruptions. Additionally, STORE THEN AND SYSTEM MASK can cause a storage-alteration event to be indicated. In all these cases, the program old PSW associated with the program interruption for the PER event may indicate that the CPU was disabled for PER events.
 - b. An instruction-fetching event may be recognized during execution of a LOAD CON-

TROL instruction that changes the value of the PER-event masks in control register 9 or the addresses in control registers 10 and 11 controlling indication of instruction-fetching events.

- c. In the access-register mode, a storage-alteration event that is permitted by a one value of the storage-alteration-event bit in an address-space-control element in an ASN-second-table entry (designated by an access-list entry) may be caused by any store-type instruction that changes the value of the bit from one to zero.
2. When a PER interruption for a PER basic event occurs during the execution of an interruptible instruction, the ILC indicates the length of that instruction or execute-type instruction, as appropriate. When a PER interruption for a PER basic event occurs as a result of LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, or SUPERVISOR CALL, the ILC indicates the length of the instruction or of the execute-type instruction which designates the interrupted type of instruction as its target, as appropriate, unless a concurrent specification exception on LOAD PSW, LOAD PSW EXTENDED, or PROGRAM RETURN calls for an ILC of 0.
3. When a PER interruption is caused by branching, the PER address identifies the branch instruction (or execute-type instruction, as appropriate), whereas the old PSW points to the next instruction to be executed. When the interruption occurs during the execution of an interruptible instruction, the PER address and the instruction address in the old PSW are the same.

Storage-Area Designation

Two types of PER events — instruction fetching and storage alteration — always involve the designation of an area in storage. Successful-branching events may involve this designation. The storage area starts at the location designated by the starting address in control register 10 and extends up to and including the location designated by the ending address in control register 11. The area extends to the right of the starting address.

An instruction-fetching event occurs whenever the first byte of an instruction or the first byte of the target

of an execute-type instruction is fetched from the designated area. A storage-alteration event occurs when a store access is made to the designated area by using an operand address that is defined to be a logical or a virtual address. However, when DAT is on and the storage-alteration-space control in control register 9 is one, a storage-alteration event occurs only when the storage area is within an address space for which the storage-alteration-event bit in the address-space-control element is one. A storage-alteration event does not occur for a store access made with an operand address defined to be a real address. When the branch-address control in control register 9 is one, a successful-branching event occurs when the first byte of the branch-target instruction is within the designated area.

The set of addresses designated for successful-branching, instruction-fetching, and storage-alteration events wraps around at address $2^{64} - 1$; that is, address 0 is considered to follow address $2^{64} - 1$. When the starting address is less than the ending address, the area is contiguous. When the starting address is greater than the ending address, the set of locations designated includes the area from the starting address to address $2^{64} - 1$ and the area from address 0 to, and including, the ending address. When the starting address is equal to the ending address, only that one location is designated.

Address comparison for successful-branching, instruction-fetching, and storage-alteration events is always performed using 64-bit addresses. This is accomplished in the 24-bit or 31-bit addressing mode by extending the virtual, logical, or instruction address on the left with 40 or 33 zeros, respectively, before comparing it with the starting and ending addresses.

Programming Note: In some models, performance of address-range checking is assisted by means of an extension to each page-table entry in the TLB. In such an implementation, changing the contents of control registers 10 and 11 when the successful-branching, instruction-fetching, or storage-alteration-event mask is one, or setting any of these PER-event masks to one, may cause the TLB to be cleared of entries. This degradation may be experienced even when the CPU is disabled for PER events. Thus, when possible, the program should avoid loading control registers 9, 10, or 11.

PER Events

Successful Branching

When the branch-address control in control register 9 is zero, a successful-branching event occurs independent of the branch-target address. When the branch-address control is one, a successful-branching event occurs only when the first byte of the branch-target instruction is in the storage area designated by control registers 10 and 11.

Subject to the effect of the branch-address control, a successful-branching event occurs whenever one of the following instructions causes branching:

- BRANCH AND LINK (BAL, BALR)
- BRANCH AND SAVE (BAS, BASR)
- BRANCH AND SAVE AND SET MODE (BASSM)
- BRANCH AND SET AUTHORITY (BSA)
- BRANCH AND SET MODE (BSM)
- BRANCH AND STACK (BAKR)
- BRANCH IN SUBSPACE GROUP (BSG)
- BRANCH ON CONDITION (BC, BCR)
- BRANCH ON COUNT (BCT, BCTR, BCTG, BCTGR)
- BRANCH ON INDEX HIGH (BXH, BXHG)
- BRANCH ON INDEX LOW OR EQUAL (BXLE, BXLEG)
- BRANCH RELATIVE AND SAVE (BRAS)
- BRANCH RELATIVE AND SAVE LONG (BRASL)
- BRANCH RELATIVE ON CONDITION (BRC)
- BRANCH RELATIVE ON CONDITION LONG (BRCL)
- BRANCH RELATIVE ON COUNT (BRCT, BRCTG)
- BRANCH RELATIVE ON COUNT HIGH (BRCTH)
- BRANCH RELATIVE ON INDEX HIGH (BRXH, BRXHG)
- BRANCH RELATIVE ON INDEX LOW OR EQUAL (BRXLE, BRXLG)
- COMPARE AND BRANCH (CRB, CGRB)
- COMPARE AND BRANCH RELATIVE (CRJ, CGRJ)
- COMPARE IMMEDIATE AND BRANCH (CIB, CGIB)
- COMPARE IMMEDIATE AND BRANCH RELATIVE (CIJ, CGIJ)
- COMPARE LOGICAL AND BRANCH (CLRB, CLGRB)

- COMPARE LOGICAL AND BRANCH RELATIVE (CLRJ, CLGRJ)
- COMPARE LOGICAL IMMEDIATE AND BRANCH (CLIB, CLGIB)
- COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (CLIJ, CLGIJ)
- RESUME PROGRAM (RP)
- TRAP (TRAP2, TRAP4)

Subject to the effect of the branch-address control, a successful-branching event also occurs whenever one of the following instructions causes branching:

- PROGRAM CALL (PC)
- PROGRAM RETURN (PR)
- PROGRAM TRANSFER (PT)
- PROGRAM TRANSFER WITH INSTANCE (PTI)

For PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, and PROGRAM TRANSFER WITH INSTANCE, the branch-target address is considered to be the new instruction address that is placed in the PSW by the instruction.

A successful-branching event causes a PER successful-branching event to be recognized if bit 32 of the PER-event masks is one and the PER mask in the PSW is one.

A PER successful-branching event is indicated by setting bit 0 of the PER code to one.

Instruction Fetching

An instruction-fetching event occurs if the first byte of the instruction is within the storage area designated by control registers 10 and 11. An instruction-fetching event also occurs if the first byte of the target of an execute-type instruction is within the designated storage area.

Instruction-Fetching Basic Event: An instruction-fetching event causes a PER instruction-fetching basic event to be recognized if the PER mask in the PSW is one and bit 33 of the PER-event masks is one and either the PER-3 facility is not installed, or bit 39 of the PER-event masks is zero.

If an instruction-fetching basic event is the only PER event recognized for an interruptible instruction that is to be interrupted because of an asynchronous condition (I/O, external, restart, or repressible machine-check condition) or the performance of the stop function, and if a unit of operation of the instruction

remains to be executed, the instruction-fetching event may be discarded, and whether it is discarded is unpredictable.

The PER instruction-fetching basic event is indicated by setting bit 1 of the PER code to one and bit 7 of the PER code to zero.

Instruction-Fetching Nullification Event: An instruction-fetching event causes a PER instruction-fetching nullification event to be recognized if the PER mask in the PSW is one and bit 33 of the PER-event masks is one and the PER-3 facility is installed and bit 39 of the PER-event masks is one.

The PER instruction-fetching nullification event is indicated by setting bits 1 and 7 of the PER code to one.

Storage Alteration

A storage-alteration event occurs whenever a CPU, by using a logical or virtual address, makes a store access without an access exception to the storage area designated by control registers 10 and 11. However, when DAT is on and the storage-alteration-space control in control register 9 is one, the event occurs only if the storage-alteration-event bit is one in the address-space-control element that is used by DAT to translate the reference to the storage location.

The contents of storage are considered to have been altered whenever the CPU executes an instruction that causes all or part of an operand to be stored within the designated storage area. Alteration is considered to take place whenever storing is considered to take place for purposes of indicating protection exceptions, except that recognition does not occur for the storing of data by a channel program. (See "Recognition of Access Exceptions" on page 6-40.) Storing constitutes alteration for PER purposes even if the value stored is the same as the original value.

Implied locations that are referred to by the CPU, either in the process of performing an interruption or in the formation of a trace entry, are not monitored. Such locations include PSW and interruption-code locations and the trace entry designated by control register 12. These locations, however, are monitored when information is stored there explicitly by an instruction. Similarly, monitoring does not apply to the storing of data by a channel program. Implied locations in the linkage stack, which are stored in by

instructions that operate on the linkage stack, are monitored.

The I/O instructions are considered to alter the second-operand location only when storing actually occurs.

Storage alteration does not apply to instructions whose operands are specified to have real or absolute addresses. Thus, storage alteration does not apply to INVALIDATE DAT TABLE ENTRY (invalidation-and-clearing operation), INVALIDATE PAGE TABLE ENTRY, PAGE IN, PERFORM FRAME MANAGEMENT FUNCTION, RESET REFERENCE BIT EXTENDED, RESET REFERENCE BITS MULTIPLE, SET STORAGE KEY EXTENDED, STORE USING REAL ADDRESS, TEST BLOCK, and TEST PENDING INTERRUPTION (when the effective address is zero). Storage alteration does not apply to the store to real location 200 by STORE FACILITY LIST, nor does it apply to stores to the trace table by instructions that cause tracing to occur.

A storage-alteration event causes a PER storage-alteration event to be recognized if bit 34 of the PER-event masks is one and the PER mask in the PSW is one. Bit 36 of the PER-event masks is ignored when determining whether a PER storage-alteration event is to be recognized.

A PER storage-alteration event is indicated by setting bit 2 of the PER code to one and bit 4 of the PER code to zero.

Store Using Real Address

A store-using-real-address event occurs whenever the STORE USING REAL ADDRESS instruction is executed.

There is no relationship between the store-using-real-address event and the designated storage area.

A store-using-real-address event causes a PER store-using-real-address event to be recognized if bits 34 and 36 of the PER-event mask are ones and the PER mask in the PSW is one.

A PER store-using-real-address event is indicated by setting bits 2 and 4 of the PER code to one.

Indication of PER Events Concurrently with Other Interruption Conditions

When a PER instruction-fetching nullification event is reported, no other PER events and no other program interruption conditions are reported.

The following rules govern the indication of PER basic events caused by an instruction that also causes a program exception, a monitor event, a space-switch event, or a supervisor-call interruption.

1. The indication of an instruction-fetching basic event does not depend on whether the execution of the instruction was completed, terminated, suppressed, or nullified. However, special cases of suppression and nullification are as follows:
 - a. When the instruction is designated by an odd instruction address in the PSW, the instruction-fetching event is not indicated.
 - b. When an access exception applies to the first, second, or third halfword of the instruction, and the PER-3 facility is installed, the instruction-fetching event is not indicated. However, if the PER-3 facility is not installed, it is unpredictable whether the instruction-fetching event is indicated.
 - c. When the target address of EXECUTE is odd or an access exception applies to the first, second, or third halfword of the target instruction, and the PER-3 facility is installed, the instruction-fetching event is not indicated for the target location. However, if the PER-3 facility is not installed, it is unpredictable whether the instruction-fetching event is indicated for the target instruction, and it is also unpredictable whether the event is indicated for the EXECUTE instruction.
2. When the operation is completed or partially completed, the event is indicated, regardless of whether any program exception, space-switch event, or monitor event is also recognized.
3. Successful branching, storage alteration, and store using real address are not indicated for an operation or, in case the instruction is interruptible, for a unit of operation that is suppressed or nullified.
4. When the execution of the instruction is terminated, storage alteration is indicated whenever the event has occurred. A model may indicate the event if the event would have occurred had the execution of the instruction been completed, even if altering the contents of the result field is contingent on operand values. For purposes of this definition, the occurrence of those exceptions which permit termination (addressing, protection, and data) is considered to cause termination, even if no result area is changed.
5. When LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, SET SYSTEM MASK, STORE THEN OR SYSTEM MASK, or SUPERVISOR CALL causes a PER basic condition and at the same time introduces a new PSW with the type of PSW-format error that is recognized immediately after the PSW becomes active, the interruption code identifies both the PER basic condition and the specification exception.
6. When LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, or SUPERVISOR CALL causes a PER condition and at the same time introduces a new PSW with the type of PSW-format error that is recognized as part of the execution of the following instruction, the introduced PSW is stored as the old PSW without the following instruction being fetched or executed and without the specification exception being recognized.

The indication of PER basic events concurrently with other program-interruption conditions for the same instruction, as described in cases 1-4 above, is summarized in Figure 4-8 on page 4-34. Cases 5 and 6 are shown in Figure 4-9 on page 4-36.

Programming Notes:

1. The execution of the interruptible instructions COMPARE AND FORM CODEWORD, COMPARE LOGICAL LONG, COMPARE UNTIL SUBSTRING EQUAL, COMPRESSION CALL, MOVE LONG, TEST BLOCK, and UPDATE TREE can cause events for instruction fetching. The execution of the interruptible instructions PERFORM FRAME MANAGEMENT FUNCTION (when the enhanced-DAT facility is installed, and the frame-size code designates a 1M-byte frame), SET STORAGE KEY EXTENDED (when the enhanced-DAT facility is installed, and the multiple-block control is one) and TEST BLOCK can

cause events for instruction-fetching. Execution of COMPRESSION CALL, MOVE LONG, and UPDATE TREE can events for instruction-fetching and storage-alteration.

Interruption of such an instruction may cause a PER basic event to be indicated more than once. It may be necessary, therefore, for a program to remove the redundant event indications from the PER data. The following rules govern the indication of the applicable events during execution of these instructions:

- a. The instruction-fetching basic event is indicated whenever the instruction is fetched for execution, regardless of whether it is the initial execution or a resumption, except that the event may be discarded (not indicated) if it is the only PER event to be indicated, the interruption is due to an asynchronous interruption condition or the performance of the stop function, and a unit of operation of the instruction remains to be executed.
- b. The storage-alteration event is indicated only when data has been stored in the designated storage area by the portion of the operation starting with the last initiation and ending with the last byte transferred before the interruption. No special indication is provided on premature interruptions as to whether the event will occur again upon the resumption of the operation. When the designated storage area is a single byte location, a storage-alteration event can be recognized only once

in the execution of MOVE LONG or COMPRESSION CALL, but could be recognized more than once for UPDATE TREE.

2. The following is an outline of the general action a program must take to delete multiple entries for PER basic events in the PER data for an interruptible instruction so that only one entry for each complete execution of the instruction is obtained:
 - a. Check to see if the PER address is equal to the instruction address in the old PSW and if the last instruction executed was interruptible.
 - b. If both conditions are met, delete instruction-fetching events.
 - c. If both conditions are met and the event is storage alteration, delete the event if some part of the remaining destination operand is within the designated storage area.
3. An example of the indication of a PER instruction-fetching basic event caused by either a LOAD PSW (or LOAD PSW EXTENDED) instruction or the following instruction, in connection with an early PSW-format error or odd instruction address introduced by the LOAD PSW instruction, is shown in Figure 4-9 on page 4-36.

Concurrent Condition	Type of Ending	PER Basic Event			
		Branch	Instr. Fetch	Alter. Storage	STURA
Specification					
Odd instruction address in the PSW	S	No	No	No	No
Instruction access (without PER-3)	N or S	No	U	No	No
Instruction access (with PER-3)	N or S	No	No	No	No
Specification					
EXECUTE target address odd (without PER-3)	S	No	U	No	-
EXECUTE target address odd (with PER-3)	S	No	No	No	-
EXECUTE or EXECUTE RELATIVE LONG target access (without PER-3)	N or S	No	U	No	-
(with PER-3)	N or S	No	No	No	-
Other nullifying	N	No	Yes	No ¹	-
Other suppressing	S	No	Yes	No ¹	No
All terminating	T	No	Yes	Yes ²	-
All completing	C	Yes	Yes	Yes	-

Figure 4-8. Indication of PER Basic Events with Other Concurrent Conditions

Concurrent Condition	Type of Ending	PER Basic Event			
		Branch	Instr. Fetch	Alter. Storage	STURA
Explanation:					
-		The condition does not apply.			
1		Although PER events of this type are not indicated for the current unit of operation of an interruptible instruction, PER events of this type that were recognized on completed units of operation of the interruptible instruction are indicated.			
2		This event may be indicated, depending on the model, if the event has not occurred but would have been indicated if execution had been completed.			
C		The operation or, in the case of the interruptible instructions, the unit of operation is completed.			
N		The operation or, in the case of the interruptible instructions, the unit of operation is nullified.			
S		The operation or, in the case of the interruptible instructions, the unit of operation is suppressed.			
T		The execution of the instruction is terminated.			
Yes		The PER event is indicated with the other program interruption condition if the event has occurred; that is, the instruction address in the PSW was replaced and the branch-address control and designated storage area allow the event occurrence, an attempt was made to execute an instruction whose first byte is located in the designated storage area, or the contents of the designated storage area were altered.			
No		The PER event is not indicated.			
U		It is unpredictable whether the PER event is indicated.			

Figure 4-8. Indication of PER Basic Events with Other Concurrent Conditions

LPSW at 4000 Loads a PSW		Designated Storage Area Includes		Two-Byte Instruction Is at 6000			
PSW Has Early PSW-Format Error	Instruction Address in PSW	4000	6000-6001	Interruption Code	Address in Program Old PSW	ILC	PER Address
N	6000	N	N	None	-	-	-
N	6000	N	Y	P	6002	1	6000
N	6000	Y	-	P	6000	2	4000
N	6001	N	N	S	6001+J ¹	K ¹	None
N	6001	N	Y	S ²	6001+J ¹	K ¹	None
N	6001	Y	-	P ^{2 3}	6001 ^{2 3}	2	4000
Y	6000	N	-	S	6000	0 ⁴	None
Y	6000	Y	-	P,S ^{2 3}	6000 ^{2 3}	0 ⁴	4000
Y	6001	N	-	S	6001	0 ⁴	None
Y	6001	Y	-	P,S ^{2 3}	6001 ^{2 3}	0 ⁴	4000

Explanation:

- Immaterial or not applicable.
¹ See "ILC on Instruction-Fetching Exceptions" on page 6-7.
² See "Indication of PER Events Concurrently with Other Interruption Conditions" on page 4-33.
³ See "Priority of Indication" on page 4-29.
⁴ See "Zero ILC" on page 6-6.
J Unpredictably 2, 4, or 6.
K 1, 2, or 3 depending on whether J is 2, 4, or 6, respectively.
N No.
P PER instruction-fetching basic event.
S Specification exception.
Y Yes.

Figure 4-9. Example of Instruction-Fetching PER Basic Event and Early PSW-Format Error or Odd Instruction Address

Breaking-Event-Address Recording

When the PER-3 facility is installed, it provides the program with the address of the last instruction to cause a break in the sequential execution of the CPU. Breaking-event-address recording can be used as a debugging assist for wild-branch detection. This facility provides a 64-bit register in the CPU, called the breaking-event-address register. Each time an instruction causes a break in the sequential instruction execution (that is, the instruction address in the PSW is replaced, rather than incremented by the length of the instruction) the address of that instruction is placed in the breaking-event-address register. Whenever a program interruption occurs, whether or not PER is indicated, the current contents of the

breaking-event-address register are placed in real storage locations 272-279.

Normally, operation of the CPU is controlled by instructions in storage that are executed sequentially, one at a time, left to right in an ascending sequence of storage addresses. This is called sequential instruction execution, and, as part of this operation, the instruction address in the PSW is incremented by the length of each instruction executed. The execution of some instructions cause the contents of the instruction address to be replaced, rather than incremented. The action to replace the instruction address is called a breaking event, and the instruction causing the action is called an execution-break instruction.

Programming Note: The addressing mode and address space used to translate the instruction address are not preserved; therefore, breaking-event-address recording is most effective for debugging code that does not contain address-space switches or addressing mode changes.

Breaking-Event-Address Register

When the PER-3 facility is installed, each CPU has a 64-bit register called the breaking-event-address register.

Each time execution of an instruction replaces the instruction address in the PSW by any means other than sequential instruction execution, the instruction address used to fetch that instruction is placed in the breaking-event-address register. If the instruction causing the breaking event is the target of an execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG), then the instruction address used to fetch the execute-type instruction is placed in the breaking-event-address register.

Each time a program interruption occurs, the contents of the breaking-event-address register are placed in real storage locations 272-279.

The contents of the breaking-event-address register are unpredictable after the SIGNAL PROCESSOR set-architecture order.

The contents of the breaking-event-address register are reset to a value of 0000000000000001 hex by initial CPU reset.

Execution-Break Instructions

A breaking event is considered to occur whenever one of the following instructions causes branching:

- BRANCH AND LINK (BAL, BALR)
- BRANCH AND SAVE (BAS, BASR)
- BRANCH AND SAVE AND SET MODE (BASSM)
- BRANCH AND SET MODE (BSM)
- BRANCH AND STACK (BAKR)
- BRANCH ON CONDITION (BC, BCR)
- BRANCH ON COUNT (BCT, BCTR, BCTG, BCTGR)

- BRANCH ON INDEX HIGH (BXH, BXHG)
- BRANCH ON INDEX LOW OR EQUAL (BXLE, BXLEG)
- BRANCH RELATIVE ON CONDITION (BRC)
- BRANCH RELATIVE ON CONDITION LONG (BRCL)
- BRANCH RELATIVE ON COUNT (BRCT, BRCTG)
- BRANCH RELATIVE ON COUNT HIGH (BRCTH)
- BRANCH RELATIVE ON INDEX HIGH (BRXH, BRXHG)
- BRANCH RELATIVE ON INDEX LOW OR EQUAL (BRXLE, BRXLG)
- COMPARE AND BRANCH (CRB, CGRB)
- COMPARE AND BRANCH RELATIVE (CRJ, CGRJ)
- COMPARE IMMEDIATE AND BRANCH (CIB, CGIB)
- COMPARE IMMEDIATE AND BRANCH RELATIVE (CIJ, CGIJ)
- COMPARE LOGICAL AND BRANCH (CLRB, CLGRB)
- COMPARE LOGICAL AND BRANCH RELATIVE (CLRJ, CLGRJ)
- COMPARE LOGICAL IMMEDIATE AND BRANCH (CLIB, CLGIB)
- COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (CLIJ, CLGIJ)

A breaking event is also considered to occur whenever one of the following instructions completes:

- BRANCH AND SET AUTHORITY (BSA)
- BRANCH IN SUBSPACE GROUP (BSG)
- BRANCH RELATIVE AND SAVE (BRAS)
- BRANCH RELATIVE AND SAVE LONG (BRASL)
- LOAD PSW (LPSW)
- LOAD PSW EXTENDED (LPSWE)
- PROGRAM CALL (PC)
- PROGRAM RETURN (PR)
- PROGRAM TRANSFER (PT)
- PROGRAM TRANSFER WITH INSTANCE (PTI)
- RESUME PROGRAM (RP)
- TRAP (TRAP2, TRAP4)

Timing

The timing facilities include three facilities for measuring time: the TOD clock, the clock comparator, and the CPU timer. A TOD programmable register is associated with the TOD clock.

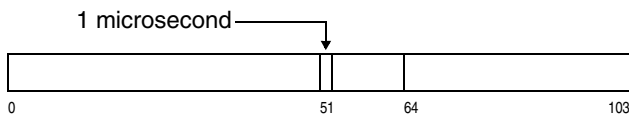
In a multiprocessing configuration, a single TOD clock is shared by all CPUs. Each CPU has its own clock comparator, CPU timer, and TOD programmable register.

Time-of-Day Clock

The time-of-day (TOD) clock provides a high-resolution measure of real time suitable for the indication of date and time of day. The cycle of the clock is approximately 143 years. A single TOD clock is shared by all CPUs in the configuration.

Format

The TOD clock is a 104-bit register. It is a binary counter with the format shown in the following illustration.



The TOD clock nominally is incremented by adding a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is incremented at such a frequency that the rate of advancing the clock is the same as if a one were added in bit position 51 every microsecond. The resolution of the TOD clock is such that the incrementing rate is comparable to the instruction-execution rate of the model.

When incrementing of the clock causes a carry to be propagated out of bit position 0, the carry is ignored, and counting continues from zero. The program is not alerted, and no interruption condition is generated as a result of the overflow.

The operation of the clock is not affected by any normal activity or event in the system. Incrementing of the clock does not depend on whether the wait-state bit of the PSW is one or whether the CPU is in the operating, load, stopped, or check-stop state. Its operation is not affected by CPU, initial-CPU, or clear

resets or by initial program loading. Operation of the clock is also not affected by the setting of the rate control or by an initial-machine-loading operation. Depending on the model and the configuration, the TOD clock may or may not be powered independent of the CPU.

States

The following states are distinguished for the TOD clock: set, not set, stopped, error, and not operational. The state determines the condition code set by execution of STORE CLOCK, STORE CLOCK EXTENDED, and STORE CLOCK FAST. The clock is incremented, and is said to be running, when it is in either the set state or the not-set state.

Not-Set State: When the power for the clock is turned on, the clock is set to zero, and the clock enters the not-set state. The clock is incremented when in the not-set state. When the TOD-clock-steering facility is installed, the TOD clock is never reported to be in the not-set state, as the TOD clock is placed in the set state as part of the initial-machine-loading (IML) process.

When the clock is in the not-set state, execution of STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST causes condition code 1 to be set and the current value of the running clock to be stored.

Stopped State: The clock enters the stopped state when SET CLOCK is executed and the execution results in the clock being set. This occurs when SET CLOCK is executed without encountering any exceptions and either any manual TOD-clock control in the configuration is set to the enable-set position or the TOD-clock-control-override control, bit 42 of control register 14, is one. The clock can be placed in the stopped state from the set, not-set, and error states. The clock is not incremented while in the stopped state.

When the clock is in the stopped state, execution of STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST causes condition code 3 to be set and the value of the stopped clock to be stored.

Set State: The clock enters the set state only from the stopped state. The change of state is under control of the TOD-clock-sync-control bit, bit 34 of control register 0, of the CPU which most recently caused the clock to enter the stopped state. If the bit is zero,

the clock enters the set state at the completion of execution of SET CLOCK. If the bit is one, the clock remains in the stopped state until the bit is set to zero on that CPU or until another CPU executes a SET CLOCK instruction affecting the clock. If an external time reference (ETR) is installed, a signal from the ETR may be used to set the set state from the stopped state. When the system is not in the interpretive-execution mode, and an external time reference (ETR) is not attached to the configuration, the TOD-clock-sync control is treated as being zero, regardless of its actual value; in this case, the clock enters the set state and resumes incrementing upon completion of the instruction.

Programming Note: The STP facility does not provide a signal that will cause the clock to transition from the stopped state to the set state. If SET CLOCK is executed when the configuration is in STP-timing mode and bit 34 of control register 0 is one, the clock remains in the stopped state until the bit is set to zero on that CPU or until another CPU executes a SET CLOCK instruction affecting the clock.

Incrementing of the clock begins with the first stepping pulse after the clock enters the set state.

In the absence of an ETR, depending on the model, the clock may enter the set state from the stopped state without any programming action.

When the clock is in the set state, execution of STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST causes condition code 0 to be set and the current value of the running clock to be stored.

Error State: The clock enters the error state when a malfunction is detected that is likely to have affected the validity of the clock value. It depends on the model whether the clock can be placed in this state. A timing-facility-damage machine-check-interruption condition is generated on each CPU in the configuration whenever the clock enters the error state. When the TOD-clock-steering facility is installed, the TOD clock is never reported to be in the error state. Errors in the TOD clock cause a system check stop.

When STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST is executed and the clock is in the error state, condition code 2 is set, and the value stored is zero.

Not-Operational State: The clock is in the not-operational state when its power is off or when it is disabled for maintenance. It depends on the model whether the clock can be placed in this state. Whenever the clock enters the not-operational state, a timing-facility-damage machine-check-interruption condition is generated on each CPU in the configuration. When the TOD-clock-steering facility is installed, the TOD clock is never reported to be in the not-operational state.

When the clock is in the not-operational state, execution of STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST causes condition code 3 to be set, and zero is stored.

Programming Note: When the TOD-clock-steering facility is installed, the TOD clock has only two states, the set state and the stopped state. Assuming proper operation by the operating system, problem programs are never dispatched while the TOD clock is in the stopped state. Thus, as observed by the problem program, the TOD clock is always in the set state and there is no need to test the condition code after issuing STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST.

Changes in Clock State

When the TOD-clock-steering facility is not installed, and the TOD clock changes value because of the execution of SET CLOCK or changes state, interruption conditions pending for the clock comparator and CPU timer may or may not be recognized for up to 1.048576 seconds (2^{20} microseconds) after the change.

When the TOD-clock-steering facility is installed and SET CLOCK is issued, interruption conditions for the clock comparator and CPU timer may or may not be recognized while the physical clock is in the stopped state. After the physical clock enters the set state, interruption conditions for the clock comparator are not necessarily recognized until one of the following instructions is issued: SET CLOCK COMPARATOR (SCKC), STORE CLOCK (STCK), or STORE CLOCK EXTENDED (STCKE). After the physical clock enters the set state, interruption conditions for the CPU timer are not necessarily recognized until SET CPU TIMER (SPT) is issued. If a CPU is in the wait state when the physical clock is set or changes state, the CPU does not necessarily recognize interruption conditions for the clock comparator or CPU

timer until after it leaves the wait state and the appropriate aforementioned instruction is executed.

When the TOD-clock-steering facility is installed and the logical TOD clock is changed by PTFF-ATO or PTFF-STO, interruption conditions for the clock comparator are not necessarily recognized until one of the following instructions is issued: SET CLOCK COMPARATOR (SCKC), STORE CLOCK (STCK), or STORE CLOCK EXTENDED (STCKE). If a CPU is in the wait state when the logical TOD clock is changed, the CPU may or may not recognize interruption conditions for the clock comparator until after it leaves the wait state and one of the aforementioned instructions is executed. The CPU timer and CPU-timer interruptions are not affected by PTFF-STO and PTFF-ATO.

The results of channel-subsystem-monitoring-facility operations may be unpredictable as a result of changes to the TOD clock.

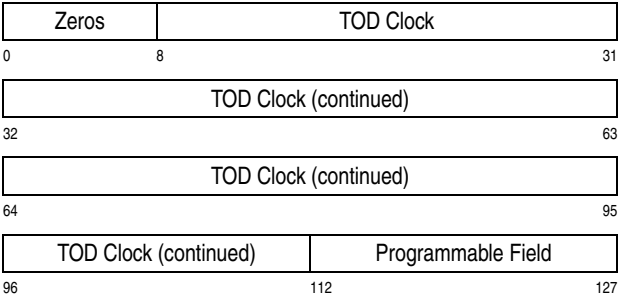
Setting and Inspecting the Clock

The clock can be set to a specified value by execution of SET CLOCK if the manual TOD-clock control of any CPU in the configuration is in the enable-set position or the TOD-clock-control-override control, bit 42 of control register 14, is one. SET CLOCK sets bits of the clock with the contents of corresponding bit positions of a doubleword operand in storage.

Setting the clock replaces the values in all bit positions from bit position 0 through the rightmost position that is incremented when the clock is running. However, on some models, the rightmost bits starting at or to the right of bit 52 of the specified value are ignored, and zeros are placed in the corresponding positions of the clock. Zeros are also placed in positions to the right of bit position 63 of the clock.

The TOD clock can be inspected by executing STORE CLOCK or STORE CLOCK FAST, which causes bits 0-63 of the clock to be stored in an eight-byte operand in storage, or by executing STORE CLOCK EXTENDED, which causes bits 0-103 of the clock to be stored in bytes 1-13 of a 16-byte operand in storage. STORE CLOCK EXTENDED stores zeros in the leftmost byte, byte 0, of its storage operand, and it obtains the TOD programmable field from bit positions 16-31 of the TOD programmable register and stores it in byte positions 14 and 15 of the stor-

age operand. The operand stored by STORE CLOCK EXTENDED has the following format:



At some time in the future, STORE CLOCK EXTENDED on new models will store a leftmost extension of the TOD clock in byte position 0 of its storage operand; see programming note 14 on page 4-43.

Two executions of STORE CLOCK or STORE CLOCK EXTENDED, possibly on different CPUs in the same configuration, always store different values of the clock if the clock is running.

The values stored for a running clock by STORE CLOCK or STORE CLOCK EXTENDED always correctly imply the sequence of execution of these instructions by one or more CPUs for all cases where the sequence can be discovered by the program. To ensure that unique values are obtained when the value of a running clock is stored, nonzero values may be stored in positions to the right of the rightmost incremented bit position. When the value of a running clock is stored by STORE CLOCK EXTENDED, the value in bit positions 64-103 of the clock (bit positions 72-111 of the storage operand) is always nonzero; this ensures that values stored by STORE CLOCK EXTENDED are always unique when compared with values stored by STORE CLOCK or STORE CLOCK FAST, extended on the right with zeros.

For the purpose of establishing uniqueness and sequence of occurrence of the results of STORE CLOCK and STORE CLOCK EXTENDED, the 64-bit value provided by STORE CLOCK may be considered to be extended to 104 bits by appending 40 zeros on the right, with the STORE CLOCK value and STORE CLOCK EXTENDED bits 8-111 then both being treated as 104-bit unsigned binary integers.

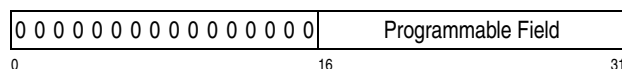
Two executions of STORE CLOCK FAST, or an execution of STORE CLOCK FAST and STORE CLOCK, either on the same or different CPUs, do not necessarily return different values of the clock if the clock is running. Thus, the values returned by STORE CLOCK FAST do not necessarily indicate the correct sequence of execution of the instruction by one or more CPUs.

Provided the TOD clock is in the not-set or set state and does not change state, and, in the absence of a carry out of bit position 0 of the TOD clock, then the value of bits 0-63 of the TOD clock as stored by STORE CLOCK FAST are never less than the value of bits 0-63 of the TOD clock as stored by a previous STORE CLOCK EXTENDED and never more than the value of bits 0-63 of the TOD clock as stored by a subsequent STORE CLOCK EXTENDED.

In a configuration where more than one CPU accesses the clock, SET CLOCK is interlocked such that the entire contents appear to be updated concurrently; that is, if SET CLOCK instructions are executed simultaneously by two CPUs, the final result is either one or the other value. If SET CLOCK is executed by one CPU and STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST by the other, the result obtained by STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST is either the entire old value or the entire new value. When SET CLOCK is executed by one CPU, a STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST instruction executed by another CPU may find the clock in the stopped state even when the TOD-clock-sync-control bit, bit 34 of control register 0, of each CPU is zero. Since the clock enters the set state before incrementing, the first STORE CLOCK, STORE CLOCK EXTENDED, or STORE CLOCK FAST instruction executed after the clock enters the set state may still find the original value introduced by SET CLOCK.

TOD Programmable Register

Each CPU has a TOD programmable register. Bits 16-31 of the register contain the programmable field that is appended on the right to the TOD-clock value by STORE CLOCK EXTENDED. The register has the following format:



The register is loaded by SET CLOCK PROGRAMMABLE FIELD. The contents of the register are reset to a value of all zeros by initial CPU reset.

Programming Notes:

1. Bit position 31 of the clock is incremented every 1.048576 seconds; for some applications, reference to the leftmost 32 bits of the clock may provide sufficient resolution.
2. Communication between systems is facilitated by establishing a standard time origin that is the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 a.m. Coordinated Universal Time (UTC) is recommended as this origin, and it is said to begin the standard epoch for the clock. This is also the epoch used when the TOD clock is synchronized to the external time reference (ETR), and, for this reason, the epoch is sometimes referred to as ETR time. The former term, Greenwich Mean Time (GMT), is now obsolete and has been replaced with the more precise UTC.
3. Historically, one of the most important uses of standard time has been for navigation. Prior to 1972, standard time, then called GMT, was defined to have a variable-length second and was synchronized to within 100 milliseconds of the rotational position of the earth. Synchronization was accomplished by occasional changes in the length of the second, typically in parts per billion, and also by occasional insertion and deletion of small increments of time, typically 50 or 100 milliseconds. Beginning in 1972, a new standard time scale, called UTC, was defined to have a fixed-length second and be kept synchronized to within 900 milliseconds of the rotational position of the earth by means of occasional adjustments of exactly one second called a leap second. The change from GMT to UTC occurred between the last second of the day on December 31, 1971 and the first second of the day on January 1, 1972 and included insertion of 107.758 milliseconds in the standard time scale to make UTC exactly 10 seconds behind International Atomic Time (TAI). For reasons of simplicity in this document, the term UTC is sometimes extrapolated backward before 1972 by assuming no time adjustments in that time scale before 1972. For the same reasons, conversion between ETR time and UTC does not take into consideration the time adjustments prior to 1972,

and, thus, ETR time differs from TAI by a fixed amount of 10 seconds. Because of the occurrence of 24 leap seconds, UTC now is behind TAI by 34 seconds.

4. A program using the clock value as a time-of-day and calendar indication must be consistent with the programming support under which the program is to be executed. If the programming support uses the standard epoch, bit 0 of the clock remains one through the years 1972-2041. (Bit 0 turned on at 11:56:53.685248 (UTC) May 11, 1971.) Ordinarily, testing bit 0 for a one is sufficient to determine if the clock value is in the standard epoch.
5. In converting to or from the current date or time, the programming support must take into account that "leap seconds" have been inserted or deleted because of time-correction standards. When the TOD clock has been set correctly to a time within the standard epoch, the sum of the accumulated leap seconds must be subtracted from the clock time to determine UTC time.
6. Because of the limited accuracy of manually setting the clock value, the rightmost bit positions of the clock, expressing fractions of a second, are normally not valid as indications of the time of day. However, they permit elapsed-time measurements of high resolution.
7. The following chart shows the time interval between instants at which various bit positions of the TOD clock are stepped. This time value may also be considered as the weighted time value that the bit, when one, represents.

TOD-Clock Bit	Stepping Interval			
	Days	Hours	Min.	Seconds
51				0.000 001
47				0.000 016
43				0.000 256
39				0.004 096
35				0.065 536
31				1.048 576
27				16.777 216
23			4	28.435 456
19		1	11	34.967 296

TOD-Clock Bit	Stepping Interval			
	Days	Hours	Min.	Seconds
15		19	5	19.476 736
11	12	17	25	11.627 776
7	203	14	43	6.044 416
3	3257	19	29	36.710 656

8. The following chart shows the TOD clock setting for 00:00:00 (0 am), UTC time, for several dates: January 1, 1900, January 1, 1972, and for that instant in time just after each of the 24 leap seconds that have occurred through January, 2009. Each of these leap seconds was inserted in the UTC time scale beginning at 23:59:60 UTC of the day previous to the one listed and ending at 00:00:00 UTC of the day listed.

Year	Month	Day	Leap Sec.	Clock Setting (Hex)
1900	1	1		0000 0000 0000 0000
1972	1	1		8126 D60E 4600 0000
1972	7	1	1	820B A981 1E24 0000
1973	1	1	2	82F3 00AE E248 0000
1974	1	1	3	84BD E971 146C 0000
1975	1	1	4	8688 D233 4690 0000
1976	1	1	5	8853 BAF5 78B4 0000
1977	1	1	6	8A1F E595 20D8 0000
1978	1	1	7	8BEA CE57 52FC 0000
1979	1	1	8	8DB5 B719 8520 0000
1980	1	1	9	8F80 9FDB B744 0000
1981	7	1	10	9230 5C0F CD68 0000
1982	7	1	11	93FB 44D1 FF8C 0000
1983	7	1	12	95C6 2D94 31B0 0000
1985	7	1	13	995D 40F5 17D4 0000
1988	1	1	14	9DDA 69A5 57F8 0000
1990	1	1	15	A171 7D06 3E1C 0000
1991	1	1	16	A33C 65C8 7040 0000
1992	7	1	17	A5EC 21FC 8664 0000
1993	7	1	18	A7B7 0ABE B888 0000
1994	7	1	19	A981 F380 EAAC 0000
1996	1	1	20	AC34 336F ECD0 0000
1997	7	1	21	AEE3 EFA4 02F4 0000
1999	1	1	22	B196 2F93 0518 0000
2006	1	1	23	BE25 1097 973C 0000
2009	1	1	24	C387 0CB9 BB60 0000

9. The stepping value of TOD-clock bit position 63, if implemented, is 2^{-12} microseconds, or approximately 244 picoseconds. This value is called a clock unit.

The following chart shows various time intervals in clock units expressed in hexadecimal notation.

Interval	Clock Units (Hex)
1 microsecond	1000
1 millisecond	3E 8000
1 second	F424 0000
1 minute	39 3870 0000
1 hour	D69 3A40 0000
1 day	1 41DD 7600 0000
365 days	1CA E8C1 3E00 0000
366 days	1CC 2A9E B400 0000
1,461 days*	72C E4E2 6E00 0000
Explanation:	
* Number of days in four years, including a leap year. Note that the year 1900 was not a leap year. Thus, the four-year span starting in 1900 has only 1,460 days.	

10. The charts in notes 6-8 are useful when examining the value stored by STORE CLOCK. Similar charts for use when examining the value stored by STORE CLOCK EXTENDED are in programming notes at the end of the definition of that instruction.
11. In a multiprocessing configuration, after the TOD clock is set and begins running, the program should delay activity for 2^{20} microseconds (1.048576 seconds) to ensure that the CPU-timer and clock-comparator interruption conditions are recognized by the CPU.
12. Due to the sequencing rules for the results of STORE CLOCK and STORE CLOCK EXTENDED, the execution of STORE CLOCK may be considerably slower than that of STORE CLOCK EXTENDED and STORE CLOCK FAST.
13. Uniqueness of TOD-clock values can be extended to apply to processors in separate configurations by including a configuration identification in the TOD programmable field.
14. At some time in the future, new models will use a carry from bit position 0 of the TOD clock to increment an additional eight-bit binary counter. STORE CLOCK EXTENDED will store the contents of this counter in byte position 0 of its storage operand. A variation of SET CLOCK will set the counter, as well as the TOD clock. Variations of SET CLOCK COMPARATOR and STORE CLOCK COMPARATOR will manipulate a comparable byte at the left of the clock comparator.

These actions will allow the TOD clock to continue to measure time within the standard epoch after the current 143-year limit caused by a carry from bit position 0 has been exceeded, and they will allow continued use of the clock comparator. It may be desired to have programs that process 16-byte STORE CLOCK EXTENDED operands take these future developments into account.

TOD-Clock Synchronization

The following functions are provided if the configuration is part of an ETR network:

- A clock in the stopped state, with the TOD-clock-sync-control bit (bit 34 of control register 0) set to one, is placed in the set state and starts incrementing when an ETR signal occurs.
- The stepping rates for the TOD clock and the ETR are synchronized.
- Bits 32 through the rightmost incremented bit of a clock in the set state are compared with the same bits of the ETR. An unequal condition is signaled by an external-damage machine-check-interruption condition. The machine-check-interruption condition may not be recognized for up to 1.048576 seconds (2^{20} microseconds) after the unequal condition occurs.

When the server timer protocol (STP) facility is installed, the timing mode, timing state and STP-clock-source state are defined as described below.

Timing Mode

The timing mode specifies the method by which the TOD clock is maintained for purposes of synchronization within a timing network. A TOD clock operates in one of the following timing modes:

Local Timing Mode: When the configuration is in local timing mode, the TOD clock has been initialized to a local time and is being stepped at the rate of the local hardware oscillator. The configuration is not part of a synchronized timing network.

ETR Timing Mode: When the configuration is in ETR-timing mode, the TOD clock has been initialized to the ETR and is being stepped by stepping signals from ETR. To be in ETR-timing mode, the configuration must be part of an ETR network.

STP Timing Mode: When the configuration is in STP-timing mode, the TOD clock has been initialized to coordinated server time (CST) and is being stepped at the rate of the local hardware oscillator. In STP timing mode, the TOD clock is steered so as to maintain, or attain, synchronization with CST. To be in STP-timing mode, the configuration must be part of an STP network.

Timing State

The timing state indicates the synchronization state of the TOD clock with respect to the timing network reference time.

Synchronized State: When a configuration is in the synchronized timing state, the TOD clock is in synchronization with the timing-network reference time as defined below:

- If the configuration is in ETR-timing mode, the configuration is synchronized with the ETR.
- If the configuration is in STP timing mode, the configuration is synchronized with coordinated server time (CST).

A configuration that is in the local-timing or uninitialized-timing mode is never in the synchronized state.

Unsynchronized State: When a configuration is in the unsynchronized timing state, the TOD clock is not in synchronization with the timing network reference time as defined below:

- If the configuration is in ETR-timing mode, the configuration has lost synchronization with the ETR.
- If the configuration is in STP timing mode, the configuration has lost or has not been able to attain synchronization with coordinated server time (CST). The configuration is out of synchronization with CST when the TOD clock differs from CST by an amount that exceeds a model dependent STP-sync-check-threshold value.

Stopped State: When a configuration is in the stopped state, the TOD clock is either in the stopped state or TOD-clock recovery is in progress. After TOD-clock recovery completes, the TOD clock enters either the synchronized or unsynchronized state.

STP Clock Source State

The STP-clock-source state indicates whether a usable STP-clock source is available. The STP-clock source is used to determine the coordinated server time (CST) required to be able to synchronize the TOD clock.

Not Usable: The not-usable-STP-clock-source state indicates that a usable STP-clock source is not available to the STP facility. When a usable STP-clock source is not available, CST can not be determined.

Usable: The usable-STP-clock-source state indicates that a usable STP-clock source is available to the STP facility. When a usable STP-clock source is available, CST has been determined and can be used to synchronize the TOD clock to the STP network.

Programming Notes:

1. ETR TOD-clock synchronization provides for synchronizing and checking only bits 32 through the rightmost incremented bit of the TOD clock. Bits 0-31 of the TOD clock may be different from those of the ETR.
2. If the configuration is part of an ETR network, SET CLOCK must place all zeros in bit positions 32 through the rightmost incremented bit position of the TOD clock; otherwise, an external-damage machine-check-interruption condition will be recognized.

TOD-Clock Steering

TOD-clock steering provides a means to change the apparent stepping rate of the TOD clock without changing the physical hardware oscillator which steps the physical clock. This is accomplished by means of a TOD-offset register which is added to the physical clock to produce a logical-TOD-clock value.

The TOD-offset register nominally consists of bits 0-95, which are updated every microsecond by adding a 32-bit signed value, called the current total steering rate (r), aligned with bits 64-95 of the TOD-offset register. A carry, if any, out of bit position 0 of the TOD-offset register is ignored in this addition. In models having a higher or lower resolution, the addition is performed at such a frequency, and with r appropriately aligned, that the rate of changing the

TOD-offset register is the same as if r were added to bit positions 64-95 every microsecond.

Depending on the model, the TOD-offset register may be less than 64 bits and is updated periodically by the machine using a mathematical formula. The mathematical formula produces a result which is equivalent to the nominal rate, and is designed such that updates are never lost, even in the case when the machine operation is suspended for long periods of time. In normal operation, the update is performed frequently enough that the effect, as observed by the program, is indistinguishable from a uniform stepping rate.

TOD-clock steering includes the semiprivileged instruction `PERFORM TIMING FACILITY FUNCTION (PTFF)`, which is in the “E” format, provides a 7-bit function code in general register 0 and a parameter block address in general register 1.

TOD-clock steering permits the timing-facility-control program to adjust the apparent stepping rate of the TOD clock. The stepping rate can be adjusted ± 122 parts per million (ppm) with precision to one part in 2^{44} (about 4.9 nanoseconds per day). The stepping rate can be changed in a nondisruptive manner; that is, application programs may be running in the configuration when the change is made, with an almost imperceptible effect on the programs.

The presence of the TOD-clock steering facility is indicated by bit 28 stored by the `STORE FACILITIES LIST (STFL)` instruction. STFL bit 28, when one indicates that the TOD-clock steering facility is installed. The instruction `PERFORM TIMING FACILITY FUNCTION (PTFF)` is installed only in the z/Architecture architectural mode. Steering, however, is not turned off if the machine is set back to ESA/390 mode.

The total steering rate is made up of two components, a fine-steering rate and a gross-steering rate.

The fine-steering rate is used to correct that inaccuracy in the local oscillator which is stable over a relatively long period of time. The value will normally be less than the specified tolerance of the local oscillator (typically ± 2.0 ppm), changes will occur infrequently (on the order of once per day to once per week), and changes will be small (typically less than ± 0.2 ppm).

The gross-steering rate is used as a dynamic correction for all other effects, the most predominate being

to synchronize time with an external time source and with other clocks in the timing network. The value will normally change frequently (on the order of once per second to once per minute); and the value may range to more than ± 10 ppm.

While the fine-steering rate and gross-steering rate are used by the timing facility control program for different purposes, these components are treated interchangeably in the clock adjustment algorithm. The 32-bit fine-steering rate (f) is added to the 32-bit gross-steering rate (g) to form a 32-bit current total steering rate (r); a carry, if any, out of bit position 0, is ignored in this addition.

TOD-clock steering includes four control functions which can be issued by the timing-facility-control program: set fine-steering rate, set gross-steering rate, adjust TOD offset, and set TOD offset. The set-fine-steering-rate and set-gross-steering-rate functions operate on the fine-steering-rate and gross-steering-rate registers, respectively, and can be issued without causing any discontinuity in the logical TOD clock. These functions are available only to the timing-facility-control program.

When operating at the basic-machine level, the set-TOD-offset and adjust-TOD-offset functions permit the logical TOD clock to be set to any arbitrary value, but with a discontinuity. These functions are intended to be used by the timing-facility-control program only for initialization, testing, and in extreme error situations.

When operating at the logical-partition and virtual-machine levels, the set-TOD-offset function, when issued in the supervisor state, may be emulated by the hypervisor.

TOD-clock steering also includes several query functions which may be used, not only by the timing-facility-control program, but also by the problem program to determine the quality of the TOD clock.

TOD-Clock Steering Overview

Figure 4-10 on page 4-46 shows an overview of the TOD-clock operation, including steering. TOD-clock steering is accomplished by means of three values: a start time, a base offset, and a steering rate. These three values are used to compute a TOD offset which is added to the physical clock to form the basic-machine TOD clock. The start time and base offset

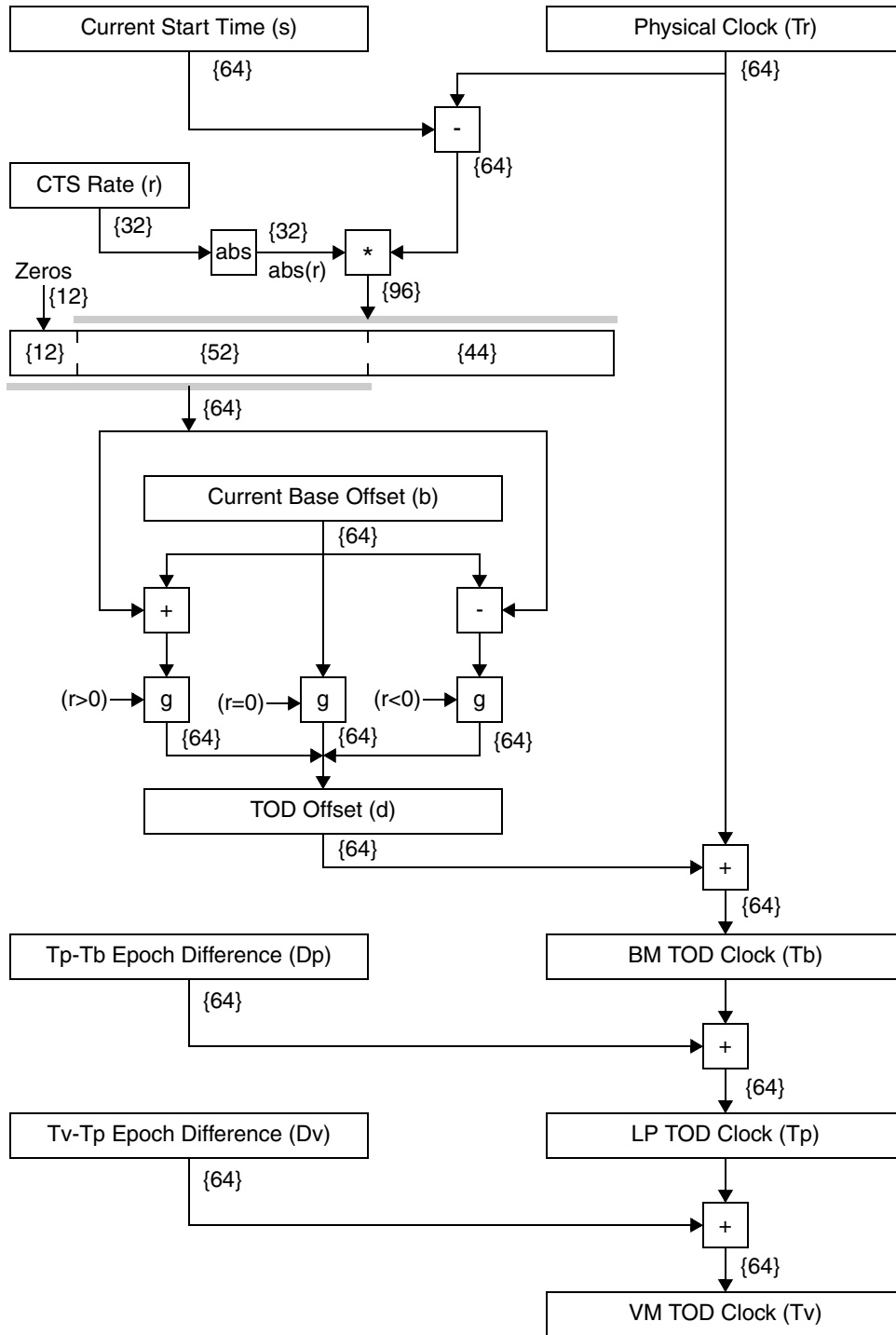


Figure 4-10. Overview of TOD Clock With Steering

are 64-bit unsigned binary integers and are considered to be aligned with bits 0-63 of the TOD clock. The steering rate is a 32-bit signed binary fixed-point value and considered to be scaled by a factor of 2^{-44} . The following formulae show the derivation of the

TOD offset (d) and basic-machine TOD clock (Tb) from the physical clock (Tr), current start time (s), current base offset (b), and current total steering rate (r):

$$d = b + (Tr - s) \cdot r \cdot 2^{-44}$$

$$T_b = T_r + d$$

Steering is also applied to the logical-partition TOD clock (T_p) and the virtual-machine TOD clock (T_v) as shown in the figure.

TOD-Offset-Update Events

In the actual implementation, rather than being computed continuously, the TOD offset (d) is updated periodically. This update, referred to as a TOD-offset-update event, is triggered by the carry out of a bit position of the physical clock. The bit position depends on the model, but is chosen such that for normal steering rates, the difference between the values computed for the TOD offset (d) by consecutive TOD-offset-update events is less than the resolution of the TOD clock.

Episodes

The three values, s , b , and r , define a linear steering adjustment which can be applied indefinitely. The duration that these values are applied without being changed is called an episode. Whenever the timing-facility-control program requests a change to the steering rate, the machine schedules a new episode to take effect at a future time. To provide a smooth transition, the machine schedules the start time for the new episode to be at the next TOD-offset-update event and computes a new base offset such that there will be no discontinuity in the value of the TOD offset at the instant the new values take effect. The machine places the new values into special registers called new-episode start time (new.s), new-episode base offset (new.b), new-episode fine-steering rate (new.f); and new-episode gross-steering rate (new.g); and the previous contents of these four registers are preserved by placing them into registers called old-episode start time (old.s), old-episode base offset (old.b), old-episode fine-steering rate (old.f), and old-episode gross-steering rate (old.g), respectively. The machine continues to use the values for the old episode until the physical clock reaches the new-episode start time (new.s) and then automatically switches to use the values for the new episode. The registers in use at any particular instant in time are called current start time (s), current base offset (b), and current total steering rate (r). These are collectively referred to as the current-episode registers.

TOD-Clock-Steering Registers

Figure 4-11 summarizes the TOD-clock-steering registers. The contents of all TOD-clock-steering regis-

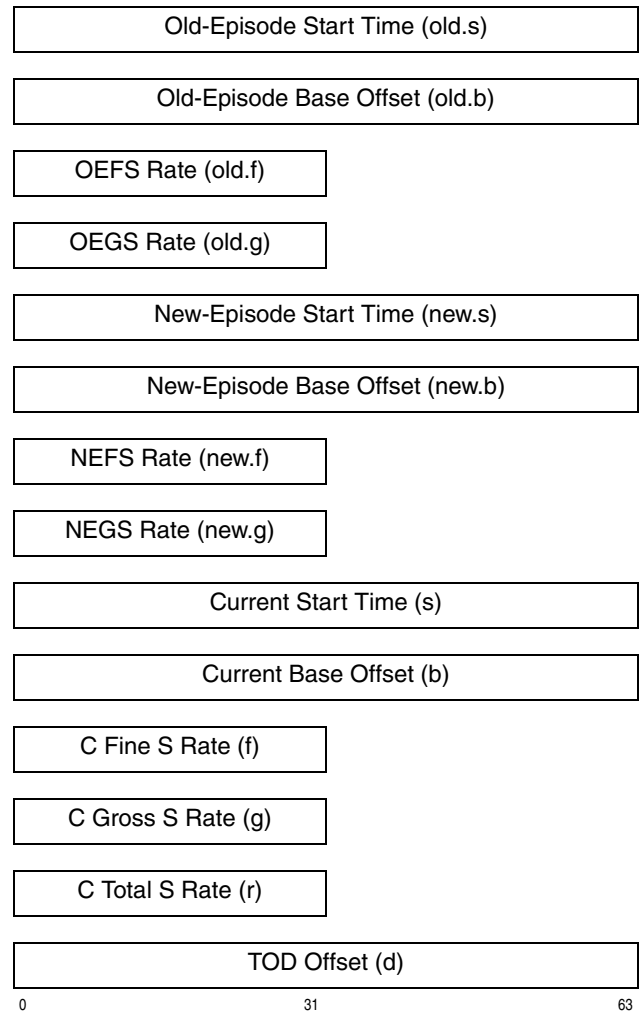


Figure 4-11. TOD-Clock-Steering Registers

ters are initialized to zero by power-on reset. Execution of SET CLOCK also sets the contents of all TOD-clock steering registers to zero regardless of whether the CPU is in the ESA/390 or z/Architecture architectural mode when the instruction is issued.

Current Start Time (s): When the machine is operating in the old episode, the current start time is obtained from the old-episode start time (old.s); and when in the new episode, it is obtained from the new-episode start time (new.s). The current start time (s) is a 64-bit unsigned binary integer and is considered to be aligned with bits 0-63 of the TOD clock. In the calculation of the steering adjustment, the current

start time (s) is subtracted from the value of the physical clock (T_r) to form a 64-bit unsigned difference ($T_r - s$). A borrow (or carry) out of bit position 0 is ignored during this subtraction.

Current Base Offset (b): When the machine is operating in the old episode, the current base offset is obtained from the old-episode base offset (old.b); and when in the new episode, it is obtained from the new-episode base offset (new.b). The current base offset (b) is a 64-bit unsigned binary integer and is considered to be aligned with bits 0-63 of the TOD clock.

Current Steering Rates (f,g,r): When the machine is operating in the old episode, the current fine-steering rate (f) and current gross-steering rate (g) are obtained from the old-episode fine-steering rate (old.f) and gross-steering rate (old.g), respectively; when in the new episode, they are obtained from the new-episode fine-steering rate (new.f) and gross-steering rate (new.g), respectively.

The current total steering rate (r) is obtained from the sum of the current fine-steering rate (f) and the current gross-steering rate (g). A carry, if any, out of bit position 0, is ignored in this addition. The current total steering rate (r) is a 32-bit signed binary fixed-point value and considered to be scaled by a factor of 2^{-44} . The absolute value of r, treated as a 32-bit binary unsigned number is multiplied by the 64-bit unsigned difference ($T_r - s$) to form a 96-bit unsigned binary product, which is shifted right by 44 bit positions to form a 52-bit intermediate result. Twelve zeros are then appended on the left to form a 64-bit adjustment value. If r is zero, the current base offset is used directly as the TOD offset and no multiplication is necessary. If r is positive, the adjustment value is added to the current base offset (b) to form the TOD offset (d); if r is negative, the adjustment value is subtracted from the current base offset (b) to form the TOD offset (d). A carry, (or borrow) occurring during this addition (or subtraction) is ignored. When r is negative, then each time an TOD-offset-update event occurs within a single episode, after computing the current TOD offset (d), the machine provides an interlock by delaying long enough to ensure that the logical TOD clock appears to step forwards as viewed by the program.

Programming Note: Bits 0 and 31 of the steering-rate represent steering rates of -2^{-13} and 2^{-44} , respectively. Thus, steering rates of ± 122 parts per million

(10.5 seconds per day) may be specified with a precision of 4.9 nanoseconds per day.

TOD Offset (d): The TOD offset is a 64-bit value aligned with bits 0-63 of the TOD clock. The contents of the TOD offset, treated as a 64-bit unsigned binary value are added to the physical clock to obtain the basic-machine TOD clock. A carry, if any, out of bit position 0, is ignored in this addition. Depending on the model, rightmost bits of the TOD offset corresponding to bits beyond the resolution of the TOD clock may not be implemented and are treated as zeros.

Programming Note: Although the architecture defines the TOD offset as a 64-bit unsigned binary value, it is often convenient to consider this as a signed value. For example, if the high order 32 bits of the TOD-offset register contain a value of FFFFFFFF hex, and this value is added to bits 0-31 of the physical clock, the carry out of bit position 0 is ignored and the effect is the same as if 1.048578 seconds were subtracted from the TOD clock.

Clock Comparator

The clock comparator provides a means of causing an interruption when the TOD-clock value exceeds a value specified by the program.

In a configuration with more than one CPU, each CPU has a separate clock comparator.

The clock comparator has the same format as bits 0-63 of the TOD clock. The clock comparator nominally consists of bits 0-47, which are compared with the corresponding bits of the TOD clock. In some models, higher resolution is obtained by providing more than 48 bits. The bits in positions provided in the clock comparator are compared with the corresponding bits of the clock. When the resolution of the clock is less than that of the clock comparator, the contents of the clock comparator are compared with the clock value as this value would be stored by executing STORE CLOCK or STORE CLOCK FAST.

The clock comparator causes an external interruption with the interruption code 1004 hex. When the TOD-clock-steering facility is not installed, a request for a clock-comparator interruption exists whenever either of the following conditions exists:

1. The TOD clock is running and the value of the clock comparator is less than the value in the compared portion of the clock, both values being considered unsigned binary integers. Comparison follows the rules of unsigned binary arithmetic.
 2. The TOD clock is in the error state or the not-operational state.
- When the TOD-clock-steering facility is installed, a request for a clock-comparator interruption exists whenever the physical clock is in the set state and the value of the clock comparator is less than the value in the compared portion of the logical TOD clock, both values being considered unsigned binary integers. Comparison follows the rules of unsigned binary arithmetic.
- A request for a clock-comparator interruption does not remain pending when the value of the clock comparator is made equal to or greater than that of the logical TOD clock or when the value of the logical TOD clock is made less than the clock-comparator value. The latter may occur as a result of the physical clock or logical TOD clock being set or the logical TOD clock wrapping to zero.
- The clock comparator can be inspected by executing the instruction `STORE CLOCK COMPARATOR` and can be set to a specified value by executing the `SET CLOCK COMPARATOR` instruction.
- The contents of the clock comparator are initialized to zero by initial CPU reset.

Programming Notes:

1. An interruption request for the clock comparator persists as long as the clock-comparator value is less than that of the TOD clock or as long as the TOD clock is in the error state or the not-operational state. Therefore, one of the following actions must be taken after an external interruption for the clock comparator has occurred and before the CPU is again enabled for external interruptions: the value of the clock comparator must be replaced, the TOD clock must be set, the TOD clock must wrap to zero, or the clock-comparator-subclass mask must be set to zero. Otherwise, loops of external interruptions are formed.
2. The instruction `STORE CLOCK`, `STORE CLOCK EXTENDED`, or `STORE CLOCK FAST` may store a value which is greater than that in the clock comparator, even though the CPU is enabled for the clock-comparator interruption. This is because the TOD clock may be incremented one or more times between when instruction execution is begun and when the clock value is accessed. In this situation, the interruption occurs when the execution of `STORE CLOCK`, `STORE CLOCK EXTENDED`, or `STORE CLOCK FAST` is completed.

CPU Timer

The CPU timer provides a means for measuring elapsed CPU time and for causing an interruption when a specified amount of time has elapsed.

In a configuration with more than one CPU, each CPU has a separate CPU timer.

The CPU timer is a binary counter with a format which is the same as that of bits 0-63 of the TOD clock, except that bit 0 is considered a sign. The CPU timer nominally is decremented by subtracting a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is decremented at such a frequency that the rate of decrementing the CPU timer is the same as if a one were subtracted in bit position 51 every microsecond. The resolution of the CPU timer is such that the stepping rate is comparable to the instruction-execution rate of the model.

The CPU timer requests an external interruption with the interruption code 1005 hex whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). The request does not remain pending when the CPU-timer value is changed to a nonnegative value.

When the TOD-clock-steering facility is not installed, and both the CPU timer and the TOD clock are running, the stepping rates are synchronized such that both are stepped at the same rate. When the TOD-clock-steering facility is installed, the CPU timer and TOD clock do not necessarily step at the same rate; and may differ by as much as the maximum steering rate. Normally, decrementing the CPU timer is not affected by concurrent I/O activity. However, in some models the CPU timer may stop during extreme I/O activity and other similar interference situations. In these cases, the time recorded by the CPU timer pro-

vides a more accurate measure of the CPU time used by the program than would have been recorded had the CPU timer continued to step.

The CPU timer is decremented when the CPU is in the operating state or the load state. When the manual rate control is set to instruction step, the CPU timer is decremented only during the time in which the CPU is actually performing a unit of operation. However, depending on the model, the CPU timer may or may not be decremented when the TOD clock is in the error, stopped, or not-operational state.

Depending on the model, the CPU timer may or may not be decremented when the CPU is in the check-stop state.

The CPU timer can be inspected by executing the privileged instruction STORE CPU TIMER and can be set to a specified value by executing the privileged SET CPU TIMER instruction. The general instruction EXTRACT CPU TIME may be used in determining the amount of CPU time consumed by a task.

The CPU timer is set to zero by initial CPU reset.

Programming Notes:

1. The CPU timer in association with a program may be used both to measure CPU-execution time and to signal the end of a time interval on the CPU.
2. The time measured for the execution of a sequence of instructions may depend on the effects of such things as I/O interference, the availability of pages, and instruction retry. Therefore, repeated measurements of the same sequence on the same installation may differ.
3. The fact that a CPU-timer interruption does not remain pending when the CPU timer is set to a positive value eliminates the problem of an undesired interruption. This would occur if, between the time when the old value is stored and a new value is set, the CPU is disabled for CPU-timer interruptions and the CPU timer value goes from positive to negative.
4. The fact that CPU-timer interruptions are requested whenever the CPU timer is negative (rather than just when the CPU timer goes from positive to negative) eliminates the requirement for testing a value to ensure that it is positive before setting the CPU timer to that value.

As an example, assume that a program being timed by the CPU timer is interrupted for a cause other than the CPU timer, external interruptions are disallowed by the new PSW, and the CPU-timer value is then saved by STORE CPU TIMER. This value could be negative if the CPU timer went from positive to negative since the interruption. Subsequently, when the program being timed is to continue, the CPU timer may be set to the saved value by SET CPU TIMER. A CPU-timer interruption occurs immediately after external interruptions are again enabled if the saved value was negative.

The persistence of the CPU-timer-interruption request means, however, that after an external interruption for the CPU timer has occurred, the value of the CPU timer must be replaced, the value in the CPU timer must wrap to a positive value, or the CPU-timer-subclass mask must be set to zero before the CPU is again enabled for external interruptions. Otherwise, loops of external interruptions are formed.

5. The instruction STORE CPU TIMER may store a negative value even though the CPU is enabled for the interruption. This is because the CPU-timer value may be decremented one or more times between when instruction execution is begun and when the CPU timer is accessed. In this situation, the interruption occurs when the execution of STORE CPU TIMER is completed.

Externally Initiated Functions

Resets

Five reset functions are provided:

- CPU reset
- Initial CPU reset
- Subsystem reset
- Clear reset
- Power-on reset

CPU reset provides a means of clearing equipment-check indications and any resultant unpredictability in the CPU state with the least amount of information destroyed. In particular, it is used to clear check conditions when the CPU state is to be preserved for analysis or resumption of the operation. CPU reset sets the architectural mode to the ESA/390 mode if it

is caused by activation of the load-normal key. When CPU reset sets the ESA/390 mode, it saves the current PSW so that PSW can be restored by a SIGNAL PROCESSOR set-architecture order that changes the architectural mode back to z/Architecture.

Initial CPU reset provides the functions of CPU reset together with initialization of the current PSW, captured-z/Architecture-PSW, CPU timer, clock comparator, prefix, breaking-event-address, control, floating-point-control, and TOD programmable registers. Initial CPU reset sets the architectural mode to the ESA/390 mode if it is caused by activation of the load-normal key.

Subsystem reset provides a means for clearing floating interruption conditions as well as for invoking I/O-system reset.

Clear reset causes initial CPU reset and subsystem reset to be performed and, additionally, clears or initializes all storage locations and registers in all CPUs in the configuration, with the exception of the TOD clock. Such clearing is useful in debugging programs and in ensuring user privacy. Clear reset also releases all locks used by the PERFORM LOCKED OPERATION instruction. Clear reset sets the architectural mode to the ESA/390 mode. Clearing does not affect external storage, such as direct-access storage devices used by the control program to hold the contents of unaddressable pages.

CPU power-on reset causes initial CPU reset to be performed and clears the contents of general registers, access registers, and floating-point registers to zeros with valid checking-block code. Locks used by PERFORM LOCKED OPERATION and associated with the CPU are released unless they are held by a CPU already powered on. The power-on-reset sequences for the TOD clock, main storage, and the channel subsystem may be included as part of the CPU power-on sequence, or the power-on sequence for these units may be initiated separately. If CPU power-on reset establishes the configuration, it sets the architectural mode to the ESA/390 mode; otherwise, it sets the architectural mode to that of the CPUs already in the configuration.

CPU reset, initial CPU reset, subsystem reset, and clear reset may be initiated manually by using the operator facilities (see Chapter 12, “Operator Facilities”). Initial CPU reset is part of the initial-program-loading function. Figure 4-12 on page 4-51 summarizes how these four resets are manually initiated. Power-on reset is performed as part of turning power on. The reset actions are tabulated in Figure 4-13 on page 4-52. For information concerning which resets can be performed by the SIGNAL PROCESSOR instruction, see “Signal-Processor Orders” on page 4-58.

Key Activated	Function Performed on ¹		
	CPU on Which Key Was Activated	Other CPUs in Config.	Remainder of Config.
System-reset-normal key	CPU reset	CPU reset	Subsystem reset
System-reset-clear key	Clear reset ²	Clear reset ²	Clear reset ³
Load-normal key	Initial CPU reset, followed by IPL	CPU reset	Subsystem reset
Load-clear key	Clear reset ² , followed by IPL	Clear reset ²	Clear reset ³
Explanation: ¹ Activation of a system-reset or load key may change the configuration, including the connection with I/O, storage units, and other CPUs. ² Only the CPU elements of this reset apply. ³ Only the non-CPU elements of this reset apply.			

Figure 4-12. Manual Initiation of Resets

Area Affected	Reset Function				
	Subsystem Reset	CPU Reset	Initial CPU Reset	Clear Reset	Power-On Reset
CPU	U	S	S ¹	S ¹	S
PSW	U	U/V#	C* ¹	C* ¹	C* ¹
Captured-z/Architecture-PSW register	U	U/sv	C	C	C
Prefix	U	U/V	C	C	C
CPU timer	U	U/V	C	C	C
Clock comparator	U	U/V	C	C	C
TOD programmable register	U	U/V	C	C	C
Control registers	U	U/V	I	I	I
Breaking-event-address register	U	U/V	K	K	K
Floating-point-control register	U	U/V	C	C	C
Access registers	U	U/V	U/V	C	C
General registers	U	U/V	U/V	C	C
Floating-point registers	U	U/V	U/V	C	C
Storage keys	U	U	U	C	C ²
Volatile main storage	U	U	U	C	C ²
Nonvolatile main storage	U	U	U	C	U
Expanded storage	U ³	U ³	U ³	U ³	C ²
TOD clock	U ⁴	U ⁴	U ⁴	U ⁴	T ²
TOD clock steering registers	U	U	U	U	C
Floating interruption conditions	C	U	U	C	C ²
I/O system	R	U	U	R	R ⁵
PERFORM LOCKED OPERATION locks	U	U	U	RC	RP

Explanation:

If the architectural mode is changed from z/Architecture to ESA/390 (the reset is due to activation of the load-normal key on another CPU), the 16-byte PSW first is placed in the captured-z/Architecture-PSW register, and then does not remain unchanged. Instead, it is changed to an eight-byte PSW, and the bits of the eight-byte PSW are set as follows. Bits 0-11 and 13-32 are set equal to the same bits of the 16-byte PSW, bit 12 is set to one, and bits 33-63 are set equal to bits 97-127 of the 16-byte PSW. The PSW is invalid in the ESA/390 mode if PSW bit 31 is one.

* Clearing the contents of the PSW to zero causes the PSW to be invalid if the architectural mode is ESA/390.

1 When the IPL sequence follows the reset function on that CPU, the CPU does not necessarily enter the stopped state, and the PSW is not necessarily cleared to zeros.

2 When these units are separately powered, the action is performed only when the power for the unit is turned on.

3 Access to change expanded storage at the time a reset function is performed may cause the contents of the 4K-byte block in expanded storage to be unpredictable. Access to examine expanded storage does not affect the contents of the expanded storage.

4 Access to the TOD clock by means of STORE CLOCK at the time a reset function is performed does not cause the value of the TOD clock to be affected.

5 When the channel subsystem is separately powered or consists of multiple elements which are separately powered, the reset action is applied only to those subchannels, channel paths, and I/O control units and devices on those paths associated with the element which is being powered on.

C The condition or contents are cleared. If the area affected is a field, the contents are set to zero with valid checking-block code.

I The state or contents are initialized. If the area affected is a field, the contents are set to the initial value with valid checking-block code.

K The breaking-event-address register is initialized to 0000000000000001 hex with valid checking-block code.

R I/O-system reset is performed in the channel subsystem. As part of this reset, system reset is signaled to all I/O control units and devices attached to the channel subsystem.

RC All locks in the configuration are released.

RP All locks in the configuration are released except for ones held by CPUs already powered on.

Figure 4-13. Summary of Reset Actions (Part 1 of 2)

Area Affected		Reset Function			
		Subsystem Reset	CPU Reset	Initial CPU Reset	Power-On Reset
S	The CPU is reset; current operations, if any, are terminated; the ALB and TLB are cleared of entries; interruption conditions in the CPU are cleared; and the CPU is placed in the stopped state. The effect of performing the start function is unpredictable when the stopped state has been entered by means of a reset. If the reset is initiated by the system-reset-clear, load-normal, or load-clear key or by a CPU power-on reset that establishes the configuration, the architectural mode is set to the ESA/390 mode; otherwise, the architectural mode is unchanged, except that power-on reset sets the mode to that of the CPUs already in the configuration.				
T	The TOD clock is initialized to zero and validated; it enters the not-set state.				
U	The state, condition, or contents of the field remain unchanged. However, the result is unpredictable if an operation is in progress that changes the state, condition, or contents of the field at the time of reset.				
U/sv	The captured-z/Architecture-PSW register remains unchanged if the reset is due to activation of the system-reset-normal key or the SIGNAL PROCESSOR CPU-reset order, or it is set with the value of the current 16-byte PSW if the reset is due to activation of the load-normal key.				
U/V	The contents remain unchanged, provided the field is not being changed at the time the reset function is performed. However, on some models the checking-block code of the contents may be made valid. The result is unpredictable if an operation is in progress that changes the contents of the field at the time of reset.				

Figure 4-13. Summary of Reset Actions (Part 2 of 2)

CPU Reset

CPU reset causes the following actions:

1. The execution of the current instruction or other processing sequence, such as an interruption, is terminated, and all program-interruption and supervisor-call-interruption conditions are cleared.
2. Any pending external-interruption conditions which are local to the CPU are cleared. Floating external-interruption conditions are not cleared.
3. Any pending machine-check-interruption conditions and error indications which are local to the CPU and any check-stop states are cleared. Floating machine-check-interruption conditions are not cleared. Any machine-check condition which is reported to all CPUs in the configuration and which has been made pending to a CPU is said to be local to the CPU.
4. All copies of prefetched instructions or operands are cleared. Additionally, any results to be stored because of the execution of instructions in the current checkpoint interval are cleared.
5. The ART-lookaside buffer and translation-lookaside buffer are cleared of entries.
6. If the reset is caused by activation of the load-normal key on any CPU in the configuration, the following actions occur:
 - a. The architectural mode of the CPU (and of all other CPUs in the configuration because
- b. The current PSW is saved for subsequent use by a SIGNAL PROCESSOR set-architecture order that restores the z/Architecture mode.
- c. The current PSW is changed from 16 bytes to eight bytes. The bits of the eight-byte PSW are set as follows: bits 0-11 and 13-32 are set equal to the same bits of the 16-byte PSW, bit 12 is set to one, and bits 33-63 are set equal to bits 97-127 of the 16-byte PSW.

A CPU reset caused by activation of the system-reset-normal key or by the SIGNAL PROCESSOR CPU-reset order, and any CPU reset in the ESA/390 mode, do not affect the captured-z/Architecture-PSW register.

7. The CPU is placed in the stopped state after actions 1-6 have been completed. When the CCW-type IPL sequence follows the reset function on that CPU, the CPU enters the load state at the completion of the reset function and does not necessarily enter the stopped state during the execution of the reset operation. When the list-directed IPL sequence follows the reset function on that CPU, the CPU enters the operating state and does not necessarily enter the stopped state during the execution of the reset operation.

Registers, storage contents, and the state of conditions external to the CPU remain unchanged by CPU

reset. However, the subsequent contents of the register, location, or state are unpredictable if an operation is in progress that changes the contents at the time of the reset. A lock held by the CPU when executing PERFORM LOCKED OPERATION is not released by CPU reset.

When the reset function in the CPU is initiated at the time the CPU is executing an I/O instruction or is performing an I/O interruption, the current operation between the CPU and the channel subsystem may or may not be completed, and the resultant state of the associated channel-subsystem facility may be unpredictable.

Programming Notes:

1. Most operations which would change a state, a condition, or the contents of a field cannot occur when the CPU is in the stopped state. However, some signal-processor functions and some operator functions may change these fields. To eliminate the possibility of losing a field when CPU reset is issued, the CPU should be stopped, and no operator functions should be in progress.
2. If the architectural mode is changed to the ESA/390 mode and bit 31 of the current PSW is one, the PSW is invalid.

Initial CPU Reset

Initial CPU reset combines the CPU reset functions with the following clearing and initializing functions:

1. If the reset is caused by activation of the load-normal key, the architectural mode of the CPU (and of all other CPUs in the configuration) is set to the ESA/390 mode.
2. The contents of the current PSW, captured-z/Architecture-PSW, prefix, CPU timer, clock comparator, and TOD programmable register are set to zero. When the IPL sequence follows the reset function on that CPU, the contents of the PSW are not necessarily set to zero.
3. The contents of the control registers are set to their initial z/Architecture values. All 64 bits of the control registers are set regardless of whether the CPU is in the ESA/390 or the z/Architecture architectural mode.
4. The contents of the floating-point-control register are set to zero.

5. The contents of the breaking-event-address register are initialized to 0000000000000001 hex.

These clearing and initializing functions include validation.

Setting the current PSW to zero when the CPU is in the ESA/390 architectural mode at the end of the operation causes the PSW to be invalid, since PSW bit 12 must be one in that mode. Thus, in this case if the CPU is placed in the operating state after a reset without first introducing a new PSW, a specification exception is recognized.

Subsystem Reset

Subsystem reset operates only on those elements in the configuration which are not CPUs. It performs the following actions:

1. I/O-system reset is performed by the channel subsystem (see "I/O-System Reset" on page 17-13).
2. All floating interruption conditions in the configuration are cleared.

As part of I/O-system reset, pending I/O-interruption conditions are cleared, and system reset is signaled to all control units and devices attached to the channel subsystem (see "I/O-System Reset" on page 17-13). The effect of system reset on I/O control units and devices and the resultant control-unit and device state are described in the appropriate System Library publication for the control unit or device. A system reset, in general, resets only those functions in a shared control unit or device that are associated with the particular channel path signaling the reset.

Clear Reset

Clear reset combines the initial-CPU-reset function with an initializing function which causes the following actions:

1. The architectural mode of all CPUs in the configuration is set to the ESA/390 mode.
2. The access, general, and floating-point registers of all CPUs in the configuration are set to zero. All 64 bits of the general registers are set to zero regardless of whether the CPU was in the ESA/390 or z/Architecture architectural mode when the clear-reset function was initiated.

3. The contents of the main storage in the configuration and the associated storage keys are set to zero with valid checking-block code.
4. The locks used by any CPU in the configuration when executing the PERFORM LOCKED OPERATION instruction are released.
5. A subsystem reset is performed.

Validation is included in setting registers and in clearing storage and storage keys.

Programming Notes:

1. The architectural mode is not changed by activation of the system-reset-normal key or by execution of a SIGNAL PROCESSOR CPU-reset or initial-CPU-reset order. All CPUs in the configuration are always in the same architectural mode.
2. For the CPU-reset operation not to affect the contents of fields that are to be left unchanged, the CPU must not be executing instructions and must be disabled for all interruptions at the time of the reset. Except for the operation of the CPU timer and for the possibility of a machine-check interruption occurring, all CPU activity can be stopped by placing the CPU in the wait state and by disabling it for I/O and external interruptions. To avoid the possibility of causing a reset at the time that the CPU timer is being updated or a machine-check interruption occurs, the CPU must be in the stopped state.
3. CPU reset, initial CPU reset, subsystem reset, and clear reset do not affect the value and state of the TOD clock.
4. The conditions under which the CPU enters the check-stop state are model-dependent and include malfunctions that preclude the completion of the current operation. Hence, if CPU reset or initial CPU reset is executed while the CPU is in the check-stop state, the contents of the PSW, registers, and storage locations, including the storage keys and the storage location accessed at the time of the error, may have unpredictable values, and, in some cases, the contents may still be in error after the check-stop state is cleared by these resets. In this situation, a clear reset is required to clear the error.

Power-On Reset

The power-on-reset function for a component of the machine is performed as part of the power-on sequence for that component.

The power-on sequences for the TOD clock, main storage, expanded storage, and channel subsystem may be included as part of the CPU power-on sequence, or the power-on sequence for these units may be initiated separately. The following sections describe the power-on resets for the CPU, TOD clock, TOD-clock-steering registers, main storage, expanded storage, and channel subsystem. See also "I/O Support Functions" on page 17-1, and the appropriate System Library publication for the channel subsystem, control units, and I/O devices.

CPU Power-On Reset: The power-on reset causes initial CPU reset to be performed and may or may not cause I/O-system reset to be performed in the channel subsystem. The contents of general registers, access registers, and floating-point registers are cleared to zeros with valid checking-block code. Locks used by PERFORM LOCKED OPERATION and associated with the CPU are released unless they are held by a CPU already powered on. If the reset is associated with establishing a configuration, the CPU is placed in the ESA/390 mode; otherwise, the CPU is placed in the architectural mode of the CPUs already in the configuration.

TOD-Clock Power-On Reset: The power-on reset causes the value of the TOD clock to be set to zero with valid checking-block code and causes the clock to enter the not-set state. When the TOD-clock-steering facility is installed, the TOD clock is never reported to be in the not-set state, as the TOD clock is placed in the set state as part of the initial-machine-loading (IML) process.

TOD-Clock-Steering-Registers Power-On Reset: The power-on reset causes the value of the TOD-clock-steering registers to be set to zero with valid checking-block code.

Main-Storage Power-On Reset: For volatile main storage (one that does not preserve its contents when power is off) and for storage keys, power-on reset causes zeros with valid checking-block code to be placed in these fields. The contents of nonvolatile main storage, including the checking-block code, remain unchanged.

Expanded-Storage Power-On Reset: The contents of expanded storage are cleared to zeros with valid checking-block code.

Channel-Subsystem Power-On Reset: The channel-subsystem power-on reset causes I/O-system reset to be performed in the channel subsystem. (See “I/O-System Reset” on page 17-13.)

Initial Program Loading

Initial program loading (IPL) provides a manual means for causing a program to be read from a designated device and for initiating execution of that program.

Some models may provide additional controls and indications relating to IPL; this additional information is specified in the System Library publication for the model.

There are two types of IPL: CCW-type IPL and list-directed IPL. CCW-type IPL is provided by all machine configurations. List-directed IPL may be provided, depending on the model.

CCW-type IPL is described below. List-directed IPL is described in “List-Directed IPL” on page 17-19.

CCW-Type IPL

CCW-type IPL is initiated manually by setting the load-unit-address controls to a four-digit number to designate an input device and by subsequently activating the load-clear or load-normal key for a particular CPU. In the description which follows, the term “this CPU” refers to the CPU in the configuration for which the load-clear or load-normal key was activated.

Activating the load-clear key causes a clear reset to be performed on the configuration.

Activating the load-normal key causes an initial CPU reset to be performed on this CPU, a CPU reset to be propagated to all other CPUs in the configuration, and a subsystem reset to be performed on the remainder of the configuration.

Activating the load-clear key or the load-normal key sets the architectural mode to the ESA/390 mode. For ease of reference, the additional elements of the description of ESA/390 initial program loading are given below.

In the loading part of the operation, after the resets have been performed, this CPU then enters the load state. This CPU does not necessarily enter the stopped state during the execution of the reset operations. The load indicator is on while the CPU is in the load state.

Subsequently, a channel-program read operation is initiated from the I/O device designated by the load-unit-address controls. The effect of executing the channel program is as if a format-0 CCW beginning at absolute storage location 0 specified a read command with the modifier bits zeros, a data address of zero, a byte count of 24, the chain-command and SLI flags ones, and all other flags zeros.

The details of the channel-subsystem portion of the IPL operation are defined in “Initial Program Loading” on page 17-16.

When the IPL I/O operation is completed successfully, the subsystem-identification word for the IPL device is stored in absolute storage locations 184-187, zeros are stored in absolute storage locations 188-191, and a new PSW is loaded from absolute storage locations 0-7. If the PSW loading is successful and no machine malfunctions are detected, this CPU leaves the load state, and the load indicator is turned off. If the rate control is set to the process position, the CPU enters the operating state, and the CPU operation proceeds under control of the new PSW. If the rate control is set to the instruction-step position, the CPU enters the stopped state, with the manual indicator on, after the new PSW is loaded.

If the IPL I/O operation or the PSW loading is not completed successfully, the CPU remains in the load state, and the load indicator remains on. The contents of absolute storage locations 0-7 are unpredictable.

Store Status

The store-status operation places an architectural-mode identification and the contents of the CPU registers, except for the TOD clock, in assigned storage locations.

Figure 4-14 on page 4-57 lists the fields that are stored, their length, and their location in main storage.

Field	Length in Bytes	Absolute Address
Architectural-mode id	1	163
Floating-point registers 0-15	128	4608
General registers 0-15	128	4736
Current PSW	16	4864
Prefix	4	4888
Floating-point control register	4	4892
TOD programmable register	4	4900
CPU timer	8	4904
Zeros	1	4912
Bits 0-55 of clock comparator	7	4913
Access registers 0-15	64	4928
Control registers 0-15	128	4992

Figure 4-14. Assigned Storage Locations for Store Status

During the execution of the store-status operation, zeros are stored in bit positions 0-6, and a one is stored in bit position 7, of absolute location 163, the store-status architectural-mode identification.

When the CPU is in the ESA/390 architectural mode, the store-status operation stores all zeros at absolute location 163.

When bits 0-55 of the clock comparator are stored beginning at absolute location 4913, zeros are stored at absolute location 4912.

The contents of the registers are not changed. If an error is encountered during the operation, the CPU enters the check-stop state.

The store-status operation can be initiated manually by use of the store-status key (see “Store-Status Key” on page 12-5). The store-status operation can also be initiated at the addressed CPU by executing SIGNAL PROCESSOR, specifying the stop-and-store-status order. Execution of SIGNAL PROCESSOR specifying the store-status-at-address order permits the same status information, except for the store-status architectural-mode identification, to be stored at a designated address (see “Signal-Processor Orders” on page 4-58).

Multiprocessing

The multiprocessing facility provides for the interconnection of CPUs, via a common main storage, in

order to enhance system availability and to share data and resources. The multiprocessing facility includes the following facilities:

- Shared main storage
- CPU-to-CPU interconnection
- TOD-clock synchronization

Associated with these facilities is an external-interruption condition (malfunction alert), which is described in “Malfunction Alert” on page 6-12; and control-register positions for the TOD-clock-sync-control bit and for the mask for the external-interruption condition, which are listed in “Control Registers” on page 4-7.

The channel subsystem, including all subchannels, in a multiprocessing configuration can be accessed by all CPUs in the configuration. I/O-interruption conditions are floating and can be accepted by any CPU in the configuration.

Shared Main Storage

The shared-main-storage facility permits more than one CPU to have access to common main-storage locations. All CPUs having access to a common main-storage location have access to the entire 4K-byte block containing that location and to the associated storage key. The channel subsystem and all CPUs in the configuration refer to a shared main-storage location using the same absolute address.

CPU-Address Identification

Each CPU has a number assigned, called its CPU address. A CPU address uniquely identifies one CPU within a configuration. The CPU is designated by specifying this address in the CPU-address field of SIGNAL PROCESSOR. The CPU signaling a malfunction alert, emergency signal, or external call is identified by storing this address in the CPU-address field with the interruption. The CPU address is assigned during system installation and is not changed as a result of reconfiguration changes. The program can determine the address of the CPU by using STORE CPU ADDRESS.

CPU Signaling and Response

The CPU-signaling-and-response facility consists of SIGNAL PROCESSOR and a mechanism to interpret and act on several order codes. The facility provides for communications among CPUs, including transmitting, receiving, and decoding a set of assigned order codes; initiating the specified operation; and responding to the signaling CPU. A CPU can address SIGNAL PROCESSOR to itself. SIGNAL PROCESSOR is described in “SIGNAL PROCESSOR” on page 10-124.

Signal-Processor Orders

The signal-processor orders are specified in bit positions 56-63 of the second-operand address of SIGNAL PROCESSOR and are encoded as shown in Figure 4-15.

Code		Order
(Dec)	(Hex)	
0	00	Unassigned
1	01	Sense
2	02	External call
3	03	Emergency signal
4	04	4
5	05	Stop
6	06	Restart
7	07	Unassigned
8	08	Unassigned
9	09	Stop and store status
10	0A	Unassigned
11	0B	Initial CPU reset
12	0C	CPU reset
13	0D	Set prefix
14	0E	Store status at address
0F-11	0F-11	Unassigned
18	12	Set architecture
19	13	Conditional Emergency Signal
14	14	Unassigned
21	15	Sense Running Status
22-255	16-FF	Unassigned

Figure 4-15. Encoding of Orders

The orders are defined as follows:

Sense

The addressed CPU presents its status to the issuing CPU (see “Status Bits” on page 4-63 for a definition of the bits). No other action is caused at the

addressed CPU. The status, if not all zeros, is stored in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction, and condition code 1 is set; if all status bits are zeros, condition code 0 is set.

External Call

An external-call external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of SIGNAL PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. Only one external-call condition can be kept pending in a CPU at a time. The order is accepted only when the addressed CPU is in the stopped or the operating state.

Emergency Signal

An emergency-signal external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of SIGNAL PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. At any one time the receiving CPU can keep pending one emergency-signal condition for each CPU in the configuration, including the receiving CPU itself. The order is accepted only when the addressed CPU is in the stopped or the operating state.

Start

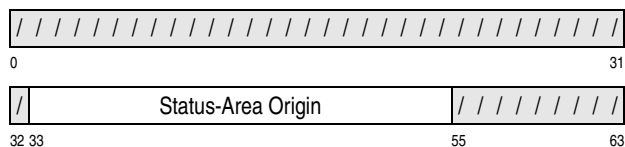
The addressed CPU performs the start function (see “Stopped, Operating, Load, and Check-Stop States” on page 4-2). The CPU does not necessarily enter the operating state during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped state. The effect of performing the start function is unpredictable when the stopped state has been entered by reset.

Stop

The addressed CPU performs the stop function (see “Stopped, Operating, Load, and Check-Stop States” on page 4-2). The CPU does not necessarily enter the stopped state during the execution of SIGNAL

The order is accepted only if the addressed CPU is in the stopped state, the status-area origin designates a location which is available in the configuration, and no other condition precludes accepting the order. Verification of the stopped state of the addressed CPU and of the availability of the designated storage is performed during execution of SIGNAL PROCESSOR. If accepted, the order is not necessarily completed during the execution of SIGNAL PROCESSOR.

The parameter register has the following format:



The store-status-at-address order is completed as follows:

- If the addressed CPU is not in the stopped state, the order is not accepted. Instead, bit 54 (incorrect state) of the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- The address of the area into which status is to be stored is tested for availability. The absolute address of a 512-byte area of storage is formed by appending 9 zeros to the right and 33 zeros to the left of bits 33-54 of the parameter value. This address is treated as a 64-bit absolute address regardless of whether the sending and receiving CPUs are in the 24-bit, 31-bit, or 64-bit addressing mode. The 512-byte block of storage at this address is accessed. The access is not subject to protection, and the associated reference bit may or may not be set to one. If the block is not available in the configuration, the order is not accepted by the addressed CPU, bit 55 (invalid parameter) of the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- The status of the addressed CPU is placed in the designated area. The information stored, and the format of the area receiving the information, are the same as for the stop-and-store-status order, except that each field, rather than being stored at an offset from the beginning of absolute storage, is stored in the designated area at the offsets

listed in Figure 4-16, and except that an architectural-mode identification is not stored. Bytes 288-291 and 312-319 of the designated area remain unchanged.

- A serialization and checkpoint-synchronization function is performed on the addressed CPU following storing of the status.

Field	Length in Bytes	Offset in Bytes
Floating-point registers 0-15	128	0
General registers 0-15	128	128
Current PSW	16	256
Prefix	4	280
Floating-point-control register	4	284
TOD programmable register	4	292
CPU timer	8	296
Zeros	1	304
Bits 0-55 of clock comparator	7	305
Access registers 0-15	64	320
Control registers 0-15	128	384

Figure 4-16. Location of Status Fields in Designated Area

Programming Note: The architectural mode of the CPU that stored status in a designated area normally is indicated by bit 12 of the PSW stored at offset 256 in the area. The PSW is stored at the same offset, 256, in both the ESA/390 mode and the z/Architecture mode. Bit 12 is one in an ESA/390 PSW and zero in a z/Architecture PSW. The store-status-at-address order does not store the architectural-mode identification that is stored at absolute location 163 by the store-status operation and the stop-and-store-status order.

Set Architecture

The contents of bit positions 56-63 of the parameter register are used as a code specifying an architectural mode to which all CPUs in the configuration are to be set: code 0 specifies the ESA/390 mode, and codes 1 and 2 specify the z/Architecture mode. Code 1 specifies that, for each of all CPUs in the configuration, the current ESA/390 PSW is to be transformed to a z/Architecture PSW. Code 2 specifies that the PSW of the CPU executing SIGNAL PROCESSOR is to be transformed to a z/Architecture PSW and that, for each of all other CPUs in the configuration, the PSW is to be set with the value of the captured-z/Architecture-PSW register for that CPU. The setting of the PSW with the value of the captured-z/Architecture-PSW register will restore the PSW that existed when the CPU was last in the z/Architecture mode,

provided that the captured-z/Architecture-PSW register has not been set to all zeros by a reset.

Bits 0-55 of the parameter register are ignored. The contents of the CPU-address register of the SIGNAL PROCESSOR instruction are ignored; all other CPUs in the configuration are considered to be addressed.

The order is accepted only if the code is 0, 1, or 2, the CPU is not already in the mode specified by the code, each of all other CPUs is in either the stopped or the check-stop state, and no other condition precludes accepting the order. If accepted, the order is completed by all CPUs during the execution of SIGNAL PROCESSOR. In no case can different CPUs be in different architectural modes.

The set-architecture order is completed as follows:

- If the code in the parameter register is not 0, 1, or 2, or if the CPU is already in the architectural mode specified by the code, the order is not accepted. Instead, bit 55 (invalid parameter) of the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- If it is not true that all other CPUs in the configuration are in the stopped or check-stop state, the order is not accepted. Instead, bit 54 (incorrect state) of the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction is set to one, and condition code 1 is set.
- The architectural mode of all CPUs in the configuration is set as specified by the code.
- If the order changes the architectural mode from ESA/390 to z/Architecture and the code is 1, then, for each CPU in the configuration, the eight-byte current PSW is changed to a 16-byte PSW, and the bits of the 16-byte PSW are set as follows: bits 0-11 and 13-32 are set equal to the same bits of the eight-byte PSW, bit 12 and bits 33-96 are set to zeros, and bits 97-127 are set equal to bits 33-63 of the eight-byte PSW. Also, bit 19 of the ESA/390 prefix, which becomes bit 51 of the z/Architecture prefix, is set to zero.

If the code is 2, the PSW of the CPU executing SIGNAL PROCESSOR and the prefix values of all CPUs are set as in the code-1 case. For each of all other CPUs in the configuration, the PSW is set with the value of the captured-z/Architecture-PSW register — a PSW saved when the CPU

last went from the z/Architecture mode to the ESA/390 mode because of a set-architecture order with code 0 or a CPU reset due to activation of the load-normal key. However, the captured-z/Architecture-PSW register has been set to all zeros if the CPU performed a reset, other than CPU reset, either at the time of the architectural-mode transition or subsequently.

- If the order changes the architectural mode from z/Architecture to ESA/390, then, for each CPU in the configuration, (1) the current PSW, which is the updated PSW in the case of the CPU executing SIGNAL PROCESSOR, is saved in the captured-z/Architecture-PSW register, and (2) the 16-byte current PSW is changed to an eight-byte PSW by setting the bits of the eight-byte PSW as follows: bits 0-11 and 13-32 are set equal to the same bits of the 16-byte PSW, bit 12 is set to one, and bits 33-63 are set equal to bits 97-127 of the 16-byte PSW. Bit 51 of the z/Architecture prefix, which becomes bit 19 of the ESA/390 prefix, remains unchanged.
- The ALBs and TLBs of all CPUs in the configuration are cleared of their contents.
- A serialization and checkpoint-synchronization function is performed on all CPUs in the configuration.

If the order changes the architectural mode from z/Architecture to ESA/390 and the SIGNAL PROCESSOR instruction causes occurrence of an instruction-fetching PER event, only the rightmost 31 bits of the address of the instruction are stored in the ESA/390 PER-address field.

Conditional Emergency Signal

An emergency-signal external-interruption condition may be generated at the addressed CPU. The contents of bit positions 48-63 of the parameter register of the SIGNAL PROCESSOR instruction are treated as a check-ASN value which is used as part of the condition checking of the addressed CPU. If signaling is permitted, the interruption condition becomes pending during the execution of SIGNAL PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. At any one time the receiving CPU can keep pending one emergency-signal condi-

tion for each CPU in the configuration, including the receiving CPU itself.

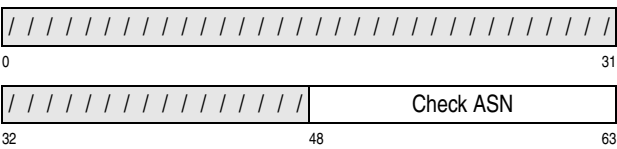
The order is accepted only when the addressed CPU is in the stopped or the operating state and one of the following conditions of the addressed CPU can be determined:

- The PSW is disabled for external interruptions, I/O interruptions, or both.
- The CPU is in the wait state and the instruction address in the PSW is not zero.
- The CPU is not in the wait state, and the specified check-ASN value equals an ASN of the CPU (primary, secondary, both), regardless of the setting of bit positions 16-17 of the PSW.

When making the above determination of possible conditions is precluded, the conditional nature of the order is not effective, and instead the order is accepted as if an emergency-signal order had been specified.

If the order is accepted, the interruption is made pending, and condition code 0 is set. Otherwise, the interruption is not made pending, bit 54 (incorrect state) of the status register is set to one, and condition code 1 is set.

The parameter register has the following format:



Sense Running Status:

The addressed CPU presents its running status to the issuing CPU. No other action is caused at the addressed CPU. If all status bits are zeros, condition code 0 is set. If the status is not all zeros, only status bit 53 is returned as one, no other status bits are one, the non-zero status is stored in the general register designated by the R₁ field of the SIGNAL PROCESSOR instruction, and condition code 1 is set.

Programming Notes:

1. If the set-architecture order changes the architectural mode from z/Architecture to ESA/390 and bit 31 of the PSW is one, the PSW is invalid.

2. For a discussion of the relative performance of the SIGNAL PROCESSOR orders, see the programming note following the instruction “SIGNAL PROCESSOR” on page 10-124.

Conditions Determining Response

Conditions Precluding Interpretation of the Order Code

The following situations preclude the initiation of the order. The sequence in which the situations are listed is the order of priority for indicating concurrently existing situations:

1. The access path to the addressed CPU is busy because a concurrently executed SIGNAL PROCESSOR is using the CPU-signaling-and-response facility. The CPU which is concurrently executing the instruction can be any CPU in the configuration other than this CPU, and the CPU address can be any address, including that of this CPU or an invalid address. The order is rejected. Condition code 2 is set.
2. The addressed CPU is not operational; that is, it is not provided in the installation, it is not in the configuration, it is in any of certain customer-engineer test modes, or its power is off. The order is rejected. Condition code 3 is set. This condition cannot arise as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.
3. One of the following conditions exists at the addressed CPU:
 - a. A previously issued start, stop, restart, stop-and-store-status, set-prefix, or store-status-at-address order has been accepted by the addressed CPU, and execution of the function requested by the order has not yet been completed.
 - b. A manual start, stop, restart, or store-status function has been initiated at the addressed CPU, and the function has not yet been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

If the currently specified order is sense, external call, emergency signal, start, stop, restart, stop and store status, set prefix, store status at address, or set architecture, then the order is

rejected, and condition code 2 is set. If the currently specified order is one of the reset orders, or an unassigned or not-implemented order, the order code is interpreted as described in “Status Bits” on page 4-63.

4. One of the following conditions exists at the addressed CPU:
 - a. A previously issued initial-CPU-reset or CPU-reset order has been accepted by the addressed CPU, and execution of the function requested by the order has not yet been completed.
 - b. A manual-reset function has been initiated at the addressed CPU, and the function has not yet been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

If the currently specified order is sense, external call, emergency signal, start, stop, restart, stop and store status, set prefix, store status at address, or set architecture, then the order is rejected, and condition code 2 is set. If the currently specified order is one of the reset orders, either the order is rejected and condition code 2 is set or the order code is interpreted as described in “Status Bits” on page 4-63.

When any of the conditions described in items 3 and 4 exists, the addressed CPU is referred to as “busy.” Busy is not indicated if the addressed CPU is in the check-stop state or when the operator-intervening condition exists. A CPU-busy condition is normally of short duration; however, the conditions described in item 3 may last indefinitely because of a string of interruptions. In this situation, however, the CPU does not appear busy to any of the reset orders.

When the conditions described in items 1 and 2 above do not apply and operator-intervening and receiver-check status conditions do not exist at the addressed CPU, reset orders may be accepted regardless of whether the addressed CPU has completed a previously accepted order. This may cause the previous order to be lost when it is only partially completed, making unpredictable whether the results defined for the lost order are obtained.

Status Bits

Various status conditions are defined whereby the issuing and addressed CPUs can indicate their responses to the specified order. The status conditions and their bit positions in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction are shown in Figure 4-17.

Bit Position	Status Condition
32	Equipment check
33-52	Unassigned; zeros stored
53	Not running
54	Incorrect state
55	Invalid parameter
56	External-call pending
57	Stopped
58	Operator intervening
59	Check stop
60	Unassigned; zero stored
61	Inoperative
62	Invalid order
63	Receiver check

Figure 4-17. Status Conditions

The status condition assigned to bit position 32, and to bit position 55 when the order is set architecture, are generated by the CPU executing SIGNAL PROCESSOR. The remaining status conditions are generated by the addressed CPU.

When the invalid-parameter condition exists for the set-architecture order, bit 55 of the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction is set to one, all other bits in bit positions 32-63 are set to zeros, bits 0-31 of the register remain unchanged, and condition code 1 is set. No other action is taken.

When the equipment-check condition exists, except when the invalid-parameter condition exists for the set-architecture order, bit 32 of the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction is set to one, unassigned bits in bit positions 32-63 of the status register are set to zeros, the other status bits are unpredictable, and bits 0-31 of the register remain unchanged. In this case, condition code 1 is set independent of whether the access path to the addressed CPU is busy and independent of whether the addressed CPU is not operational, is busy, or has presented zero status.

When the access path to the addressed CPU is not busy and the addressed CPU is operational and does not indicate busy to the currently specified order, the addressed CPU presents its status to the issuing CPU. These status bits are of two types:

1. Status bits 53, 54, and 55 when the order is not set architecture, 56-59, and 61 indicate the presence of the corresponding conditions in the addressed CPU at the time the order code is received. Except in response to the sense order and the sense-running-status order, each condition is indicated only when the condition precludes the successful execution of the specified order, although invalid parameter is not necessarily indicated when any other precluding condition exists. In the case of sense, all existing status conditions except not-running are indicated; the operator-intervening condition is indicated if it precludes the execution of any installed order.
2. Status bits 62 and 63 indicate that the corresponding conditions were detected by the addressed CPU during reception of the order.

If the presented status is all zeros, the addressed CPU has accepted the order, and condition code 0 is set at the issuing CPU; if the presented status is not all zeros, the order has been rejected, the status is stored at the issuing CPU in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction, zeros are stored in the unassigned positions in bit positions 32-63 of the register, bits 0-31 of the register remain unchanged, and condition code 1 is set.

When the order is set architecture, "the addressed CPU" refers to each of the other CPUs in the configuration. Those CPUs, in an unpredictable order, are tested for a condition that causes setting of condition code 1, 2, or 3. Conditions are prioritized for a single CPU as if it were the only CPU addressed, but there is no prioritization across CPUs. If a condition is recognized, no further CPUs are tested, the condition code corresponding to the condition is set, and the execution of SIGNAL PROCESSOR is completed.

The status conditions are defined as follows:

Equipment Check: This condition exists when the CPU executing the instruction detects equipment malfunctioning that has affected only the execution of this instruction and the associated order. The order

code may or may not have been transmitted and may or may not have been accepted, and the status bits provided by the addressed CPU may be in error. This condition is not detected if the invalid-parameter condition for the set-architecture order is detected.

Not Running: This condition exists when the addressed CPU is not running. The condition, when present, is indicated only in response to the sense-running-status order. This condition is not reported as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself; condition code 0 is always set in this case. A CPU is running when it is assigned to a physical CPU.

Incorrect State: A set-prefix or store-status-at-address order has been rejected because the addressed CPU is not stopped, or a set-architecture order has been rejected because not all other CPUs are stopped or in the check-stop state, or a conditional emergency-signal order has been rejected because the addressed CPU is not in a required state. When applicable, this status is generated during execution of SIGNAL PROCESSOR and is indicated concurrently with other indications of conditions which preclude execution of the order, except that this status is not generated if an invalid-parameter condition exists for a set-architecture order.

Invalid Parameter: This condition exists in two cases:

1. The parameter value supplied with a set-prefix or store-status-at-address order designates a storage location which is not available in the configuration. When applicable, this status is generated during execution of SIGNAL PROCESSOR, except that it is not necessarily generated when another condition precluding execution of the order also exists.
2. The parameter value supplied with a set-architecture order either is not 0 or 1 or specifies the current architectural mode. When applicable, this status is generated during execution of SIGNAL PROCESSOR, and no other status is generated.

External Call Pending: This condition exists when an external-call interruption condition is pending in the addressed CPU because of a previously issued SIGNAL PROCESSOR order. The condition exists from the time an external-call order is accepted until the resultant external interruption has been com-

pleted or a CPU reset occurs. The condition may be due to the issuing CPU or another CPU. The condition, when present, is indicated only in response to sense and to external call.

Stopped: This condition exists when the addressed CPU is in the stopped state. The condition, when present, is indicated only in response to sense. This condition cannot be reported as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

Operator Intervening: This condition exists when the addressed CPU is executing certain operations initiated from local or remote operator facilities. The particular manually initiated operations that cause this condition to be present depend on the model and on the order specified. The operator-intervening condition may exist when the addressed CPU uses reloadable control storage to perform an order and the required licensed internal code has not been loaded by the IML function. The operator-intervening condition, when present, can be indicated in response to all orders. Operator intervening is indicated in response to sense if the condition is present and precludes the acceptance of any of the installed orders. The condition may also be indicated in response to unassigned or uninstalled orders. This condition cannot arise as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

Check Stop: This condition exists when the addressed CPU is in the check-stop state. The condition, when present, is indicated only in response to sense, external call, emergency signal, conditional emergency signal, start, stop, restart, set prefix, store status at address, and stop and store status. The condition may also be indicated in response to unassigned or uninstalled orders. This condition cannot be reported as a result of a SIGNAL PROCESSOR instruction executed by a CPU addressing itself.

Inoperative: This condition indicates that the execution of the operation specified by the order code requires the use of a service processor which is inoperative. The failure of the service processor may have been previously reported by a service-processor-damage machine-check condition. The inoperative condition cannot occur for the conditional emergency signal, emergency-signal, external-call, or sense order code.

Invalid Order: This condition exists during the communications associated with the execution of SIGNAL PROCESSOR when an unassigned or uninstalled order code is decoded.

Receiver Check: This condition exists when the addressed CPU detects malfunctioning of equipment during the communications associated with the execution of SIGNAL PROCESSOR. When this condition is indicated, the order has not been initiated, and, since the malfunction may have affected the generation of the remaining receiver status bits, these bits are not necessarily valid. A machine-check condition may or may not have been generated at the addressed CPU.

The following chart summarizes which status conditions are presented to the issuing CPU in response to each order code. _

Order	Status Condition									
	53 – Not running	54 – Incorrect state	55 – Invalid parameter	56 – External call pending	57 – Stopped	58 – Operator intervening#	59 – Check stop	61 – Inoperative	62 – Invalid order	63 – Receiver check#
Sense	0	0	0	X	X	X	X	0	0	X
External call	0	0	0	X	0	X	X	0	0	X
Emergency signal	0	0	0	0	0	X	X	0	0	X
Start	0	0	0	0	0	X	X	X	0	X
Stop	0	0	0	0	0	X	X	X	0	X
Restart	0	0	0	0	0	X	X	X	0	X
Stop and store status	0	0	0	0	0	X	X	X	0	X
Initial CPU reset	0	0	0	0	0	X	0	X	0	X
CPU reset	0	0	0	0	0	X	0	X	0	X
Set prefix	0	X	X	0	0	X	X	X	0	X
Store status at address	0	X	X	0	0	X	X	X	0	X
Set architecture	0	X	X	0	0	X	0	X	0	X
Conditional emergency signal	0	X	0	0	0	X	X	0	0	X
Sense running status	X	0	0	0	0	0	0	0	0	0
Unassigned order	0	0	0	0	0	X	E	X	1	X

Figure 4-18. Status Conditions in Response to Order Code

Order	Status Condition							
	53	54	55	56	57	58	59	60
	Not running	Incorrect state	Invalid parameter	External call pending	Stopped	Operator intervening#	Check stop	Inoperative
								Invalid order
								Receiver check#
Explanation								
#	The current state of the operator-intervening condition may depend on the order code that is being interpreted.							
≠	If a one is presented in the receiver-check bit position, the values presented in the other bit positions are not necessarily valid.							
0	A zero is presented in this bit position regardless of the current state of this condition.							
1	A one is presented in this bit position.							
X	A zero or a one is presented in this bit position, reflecting the current state of the corresponding condition.							
E	Either a zero or the current state of the corresponding condition is indicated.							

Figure 4-18. Status Conditions in Response to Order Code

If the presented status bits are all zeros, the order has been accepted, and the issuing CPU sets condition code 0. If one or more ones are presented, the order has been rejected, and the issuing CPU stores the status in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction and sets condition code 1.

Programming Notes:

- All SIGNAL PROCESSOR orders except set architecture (which in effect is addressed to all other CPUs and affects all CPUs) can be addressed to this same CPU. The following are examples of functions obtained by a CPU addressing SIGNAL PROCESSOR to itself:
 - Sense* indicates whether an external-call condition is pending.
 - External call* and *emergency signal* cause the corresponding interruption conditions to be generated. *External call* can be rejected because of a previously generated external-call condition.
 - Start* sets condition code 0 and has no other effect.

- Stop* causes the CPU to set condition code 0, take pending interruptions for which it is enabled, and enter the stopped state.
- Restart* provides a means to store the current PSW.
- Stop and store status* causes the machine to stop and store all current status.

- Two CPUs can simultaneously execute SIGNAL PROCESSOR, with each CPU addressing the other. When this occurs, one CPU, but not both, can find the access path busy because of the transmission of the order code or status bits associated with SIGNAL PROCESSOR that is being executed by the other CPU. Alternatively, both CPUs can find the access path available and transmit the order codes to each other. In particular, two CPUs can simultaneously stop, restart, or reset each other.
- To obtain status from another CPU which is in the check-stop state by means of the store-status-at-address order, a CPU reset operation should first be used to bring the CPU to the stopped state. This reset order does not alter the status, and, depending on the nature of the malfunction, provides the best chance of establishing conditions in the addressed CPU which allow status to be obtained.

Facility Indications

Facilities installed in a configuration are indicated by facility bits stored by the STORE FACILITY LIST (STFL) and STORE FACILITY LIST EXTENDED (STFLE) instructions.

STORE FACILITY LIST stores an indication of up to 32 facilities in the word at real location 200. STFL is described in Chapter 10, "Control Instructions".

STORE FACILITY LIST EXTENDED stores a variable number of doublewords containing facility bits in a program-specified location. STFLE is described in Chapter 7, "General Instructions".

Figure 4-19 shows the meanings of the assigned facility bits.

A bit is set to one regardless of the current architectural mode if its meaning is true. A meaning applies

Bit	Meaning When Bit Is One	Bit	Meaning When Bit Is One
0	The instructions marked “N3” in the instruction-summary figures in Chapters 7 and 10 are installed.	18	The long-displacement facility is installed in the z/Architecture architectural mode.
1	The z/Architecture architectural mode is installed.	19	The long-displacement facility has high performance. Bit 18 is one if bit 19 is one.
2	The z/Architecture architectural mode is active. When this bit is zero, the ESA/390 architectural mode is active.	20	The HFP-multiply-and-add/subtract facility is installed.
3	The DAT-enhancement facility is installed in the z/Architecture architectural mode. The DAT-enhancement facility includes the INVALIDATE DAT TABLE ENTRY (IDTE) and COMPARE AND SWAP AND PURGE (CSPG) instructions.	21	The extended-immediate facility is installed in the z/Architecture architectural mode.
4	INVALIDATE DAT TABLE ENTRY (IDTE) performs the invalidation-and-clearing operation by selectively clearing combined region-and-segment-table entries when a segment-table entry or entries are invalidated. IDTE also performs the clearing-by-ASCE operation. Unless bit 4 is one, IDTE simply purges all TLBs. Bit 3 is one if bit 4 is one.	22	The extended-translation facility 3 is installed in the z/Architecture architectural mode.
5	INVALIDATE DAT TABLE ENTRY (IDTE) performs the invalidation-and-clearing operation by selectively clearing combined region-and-segment-table entries when a region-table entry or entries are invalidated. Bits 3 and 4 are ones if bit 5 is one.	23	The HFP-unnormalized-extension facility is installed in the z/Architecture architectural mode.
6	The ASN-and-LX reuse facility is installed in the z/Architecture architectural mode.	24	The ETF2-enhancement facility is installed.
7	The store-facility-list-extended facility is installed.	25	The store-clock-fast facility is installed in the z/Architecture architectural mode.
8	The enhanced-DAT facility is installed in the z/Architecture architectural mode.	26	The parsing-enhancement facility is installed in the z/Architecture architectural mode.
9	The sense-running-status facility is installed in the z/Architecture architectural mode.	27	The move-with-optional-specifications facility is installed in the z/Architecture architectural mode.
10	The conditional-SSKE facility is installed in the z/Architecture architectural mode.	28	The TOD-clock-steering facility is installed in the z/Architecture architectural mode.
11	The configuration-topology facility is installed in the z/Architecture architectural mode.	30	The ETF3-enhancement facility is installed in the z/Architecture architectural mode.
13	The IPTE-range facility is installed in the z/Architecture architectural mode.	31	The extract-CPU-time facility is installed in the z/Architecture architectural mode.
14	The nonquiescing key-setting facility is installed in the z/Architecture architectural mode.	32	The compare-and-swap-and-store facility is installed in the z/Architecture architectural mode.
16	The extended-translation facility 2 is installed.	33	The compare-and-swap-and-store facility 2 is installed in the z/Architecture architectural mode.
17	The message-security assist is installed.	34	The general-instructions-extension facility is installed in the z/Architecture architectural mode.
		35	The execute-extensions facility is installed in the z/Architecture architectural mode.
		36	The enhanced-monitor facility is installed in the z/Architecture architectural mode.
		37	The floating-point extension facility is installed in the z/Architecture architectural mode.

Figure 4-19. Assigned Facility Bits (Part 1 of 3)

Figure 4-19. Assigned Facility Bits (Part 2 of 3)

Bit	Meaning When Bit Is One
39	Assigned to IBM internal use.
40	The set-program-parameters facility is installed in the z/Architecture architectural mode.
41	The floating-point-support-enhancement facilities (DFP-rounding, FPR-GR-transfer, FPS-sign-handling, and IEEE-exception-simulation) are installed in the z/Architecture architectural mode.
42	The DFP (decimal-floating-point) facility is installed in the z/Architecture architectural mode.
43	The DFP (decimal-floating-point) facility has high performance. Bit 42 is one if bit 43 is one.
44	The PFPO instruction is installed in the z/Architecture architectural mode.
45	The distinct-operands, fast-BCR-serialization, high-word, interlocked-access, load/store-on-condition, and population-count facilities are installed in the z/Architecture architectural mode.
46	Assigned to IBM internal use.
47	The CMPSC-enhancement facility is installed in the z/Architecture architectural mode.
62	Assigned to IBM internal use.
63	Assigned to IBM internal use.
66	The reset-reference-bits-multiple facility is installed in the z/Architecture architectural mode.
67	The CPU-measurement counter facility is installed in the z/Architecture architectural mode.
68	The CPU-measurement sampling facility is installed in the z/Architecture architectural mode.
75	The access-exception-fetch/store-indication facility is installed in the z/Architecture architectural mode.
76	The message-security-assist-extension-3 facility is installed in the z/Architecture architectural mode.
77	The message-security-assist-extension-4 facility is installed in the z/Architecture architectural mode.

Figure 4-19. Assigned Facility Bits (Part 3 of 3)

to the current architectural mode unless it is said to apply to a specific architectural mode.

Unassigned bits are reserved for indication of new facilities; these bits may be stored as ones in the future.

STORE FACILITY LIST and STORE FACILITY LIST EXTENDED may report the absence of a facility, even though the execution of the facility's instructions may appear to indicate that the facility is installed. The under-reporting of facility indications may occur because one or more systems in the computing environment (to which the application may be relocated) do not have the facility installed.

Programming Note: Prior to the introduction of z/Architecture, determination of the presence of a facility was often accomplished by means of a trial execution of an instruction. If an operation exception was not recognized during the trial execution, then it could be assumed that the facility was present.

Similarly, certain instructions provide a query function to determine the availability of other functions of the instruction or related instructions. However, a program might attempt to determine the availability of a particular function by trial execution of the function.

With the advent of facility and function indications in z/Architecture, the technique of trial execution should be avoided — particularly if a workload may be relocated to another system in which a facility's instructions may not be present. In such an environment, STORE FACILITY LIST EXTENDED (STFLE) or an instruction's query or test function may provide a more accurate indication of facilities and functions that are available on all systems in the computing environment.

Chapter 5. Program Execution

Instructions	5-2	PC-Number Translation Control	5-31
Operands	5-3	Control Register 0	5-31
Instruction Formats	5-3	Control Register 5	5-32
Register Operands	5-6	PC-Number Translation Tables	5-32
Immediate Operands	5-6	Linkage-Table Entries	5-33
Storage Operands	5-7	Linkage-First-Table Entries	5-33
Address Generation	5-8	Linkage-Second-Table Entries	5-33
Trimodal Addressing	5-8	Entry-Table Entries	5-34
Sequential Instruction-Address Generation	5-8	Table Summary	5-35
Operand-Address Generation	5-8	PC-Number-Translation Process	5-35
Formation of the Intermediate Value	5-8	Obtaining the Linkage-Table or Linkage-	
Formation of the Operand Address	5-9	First-Table Designation	5-39
Branch-Address Generation	5-10	Linkage-Table Lookup	5-39
Formation of the Intermediate Value	5-10	Linkage-First-Table Lookup	5-39
Formation of the Branch Address	5-10	Linkage-Second-Table Lookup	5-40
Instruction Execution and Sequencing	5-11	Linkage-Second-Table-Entry-Sequence-	
Decision Making	5-11	Number Comparison	5-40
Loop Control	5-11	Entry-Table Lookup	5-40
Subroutine Linkage without the Linkage Stack	5-11	Recognition of Exceptions during PC-	
Simple Branch Instructions	5-11	Number Translation	5-41
Other Linkage Instructions	5-14	Home Address Space	5-41
Interruptions	5-19	Access-Register Introduction	5-42
Types of Instruction Ending	5-19	Summary	5-42
Completion	5-19	Access-Register Functions	5-43
Suppression	5-19	Access-Register-Specified Address	
Nullification	5-20	Spaces	5-43
Termination	5-20	Access-Register Instructions	5-49
Interruptible Instructions	5-20	Access-Register Translation	5-50
Point of Interruption	5-20	Access-Register-Translation Control	5-50
Unit of Operation	5-20	Control Register 2	5-50
Execution of Interruptible Instructions	5-21	Control Register 5	5-50
Condition-Code Alternative to		Control Register 8	5-51
Interruptibility	5-22	Access Registers	5-51
Exceptions to Nullification and Suppression	5-23	Access-Register-Translation Tables	5-52
Storage Change and Restoration for DAT-		Dispatchable-Unit Control Table and	
Associated Access Exceptions	5-23	Access-List Designations	5-52
Modification of DAT-Table Entries	5-24	Access-List Entries	5-53
Trial Execution for Editing Instructions and		ASN-Second-Table Entries	5-54
Translate Instruction	5-24	Access-Register-Translation Process	5-55
Authorization Mechanisms	5-24	Selecting the Access-List-Entry Token	5-58
Mode Requirements	5-24	Obtaining the Primary or Secondary	
Extraction-Authority Control	5-25	Address-Space-Control Element	5-58
PSW-Key Mask	5-25	Checking the First Byte of the ALET	5-58
Secondary-Space Control	5-25	Obtaining the Effective Access-List	
Subsystem-Linkage Control	5-26	Designation	5-58
ASN-Translation Control	5-26	Access-List Lookup	5-58
Authorization Index	5-26	Locating the ASN-Second-Table Entry	5-59
Instructions and Controls Related to ASN-		Authorizing the Use of the Access-List	
and-LX Reuse	5-27	Entry	5-59
PC-Number Translation	5-30		

Checking for Access-List-Controlled Protection	5-60	Updating Control Register 15	5-82
Obtaining the Address-Space-Control Element from the ASN-Second-Table Entry.	5-60	Recognition of Exceptions during the Stacking Process.	5-82
Recognition of Exceptions during Access-Register Translation.	5-60	Unstacking Process.	5-82
ART-Lookaside Buffer	5-60	Locating the Current Entry and Processing a Header Entry	5-83
ALB Structure.	5-60	Checking for a State Entry.	5-84
Formation of ALB Entries.	5-60	Restoring Information	5-84
Use of ALB Entries.	5-61	Updating the Preceding Entry	5-85
Modification of ART Tables	5-61	Updating Control Register 15	5-85
Subspace Groups	5-62	Recognition of Exceptions during the Unstacking Process.	5-85
Subspace-Group Tables	5-62	Sequence of Storage References	5-85
Subspace-Group Dispatchable-Unit Control Table	5-62	Conceptual Sequence	5-85
Subspace-Group ASN-Second-Table Entries	5-64	Overlapped Operation of Instruction Execution.	5-86
Subspace-Replacement Operations	5-65	Divisible Instruction Execution.	5-86
Linkage-Stack Introduction	5-66	Interlocks for Virtual-Storage References	5-86
Summary	5-66	Interlocks between Instructions	5-87
Linkage-Stack Functions	5-67	Interlocks within a Single Instruction	5-87
Transferring Program Control.	5-67	Instruction Fetching	5-89
Branching Using the Linkage Stack	5-69	ART-Table and DAT-Table Fetches	5-90
Adding and Retrieving Information.	5-70	Storage-Key Accesses	5-91
Testing Authorization	5-70	Storage-Operand References	5-93
Program-Problem Analysis.	5-71	Storage-Operand Fetch References	5-93
Linkage-Stack Entry-Table Entries	5-71	Storage-Operand Store References	5-94
Linkage-Stack Operations	5-72	Storage-Operand Update References.	5-94
Linkage-Stack-Operations Control.	5-74	Storage-Operand Consistency	5-95
Control Register 0	5-74	Single-Access References	5-95
Control Register 15	5-74	Multiple-Access References	5-96
Linkage Stack.	5-75	Block-Concurrent References	5-97
Entry Descriptors	5-75	Relation between Operand Accesses	5-98
Header Entries	5-76	Other Storage References	5-99
Trailer Entries.	5-77	Relation between Storage-Key Accesses	5-99
State Entries.	5-77	Serialization.	5-99
Stacking Process	5-80	CPU Serialization	5-100
Locating Space for a New Entry.	5-80	Specific-Operand Serialization	5-101
Forming the New Entry.	5-81	Channel-Program Serialization	5-102
Updating the Current Entry.	5-82	Quiescing	5-102
		Monitor-Event Counting.	5-102

Normally, operation of the CPU is controlled by instructions in storage that are executed sequentially, one at a time, left to right in an ascending sequence of storage addresses. A change in the sequential operation may be caused by branching, LOAD PSW, interruptions, SIGNAL PROCESSOR orders, or manual intervention.

Instructions

Each instruction consists of two major parts:

- An operation code (op code), which specifies the operation to be performed
- The designation of the operands that participate.

Operands

Operands can be grouped in three classes: operands located in registers, immediate operands, and operands in storage. Operands may be either explicitly or implicitly designated.

Register operands can be located in general, floating-point, access, or control registers, with the type of register identified by the op code. The register containing the operand is specified by identifying the register in a four-bit field, called the R field, in the instruction. For some instructions, an operand is located in an implicitly designated register, the register being implied by the op code.

Immediate operands are contained within the instruction, and the 8-bit, 16-bit, or 32-bit field containing the immediate operand is called the I field.

Operands in storage may have an implied length; be specified by a bitmask; be specified by a four-bit or eight-bit length specification, called the L field, in the instruction; or have a length specified by the contents of a general register. The addresses of operands in storage are specified by means of a format that uses the contents of a general register as part of the address. This makes it possible to:

1. Specify a complete address by using an abbreviated notation
2. Perform address manipulation using instructions which employ general registers for operands
3. Modify addresses by program means without alteration of the instruction stream
4. Operate independent of the location of data areas by directly using addresses received from other programs

The address used to refer to storage either is contained in a register designated by the R field in the instruction or is calculated from a base address, index, and displacement, specified by the B, X, and D fields, respectively, in the instruction.

When the CPU is in the access-register mode, a B or R field may designate an access register in addition to being used to specify an address.

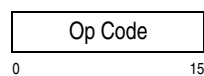
To describe the execution of instructions, operands are designated as first and second operands and, in some cases, third operands.

In general, two operands participate in an instruction execution, and the result replaces the first operand. However, CONVERT TO DECIMAL, TEST BLOCK, and instructions with “store” in the instruction name (other than STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK) use the second-operand address to designate a location in which to store. TEST AND SET, COMPARE AND SWAP, COMPARE AND SWAP AND STORE, and COMPARE DOUBLE AND SWAP may perform an update on the second operand. Except when otherwise stated, the contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

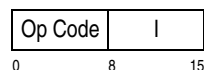
Instruction Formats

An instruction is one, two, or three halfwords in length and must be located in storage on a halfword boundary. Each instruction is in one of 26 basic formats: E, I, RI, RIE, RIL, RIS, RR, RRD, RRE, RRF, RRS, RS, RSI, RSL, RSY, RX, RXE, RXF, RXY, S, SI, SIL, SIY, SS, SSE, and SSF, with two variations of RRF, RS, RSY, RX, and RXY, three of RI and RIL, and six of RIE and SS. Figure 5-1 illustrates the various instruction formats and variations. Where a letter appears to the left of a format variant, it represents one of the variations in assembler syntax as illustrated in the description of the instructions in chapters 7-10, 14, and 18-20.

E Format



I Format



RI Formats

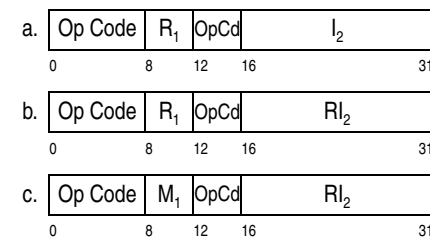
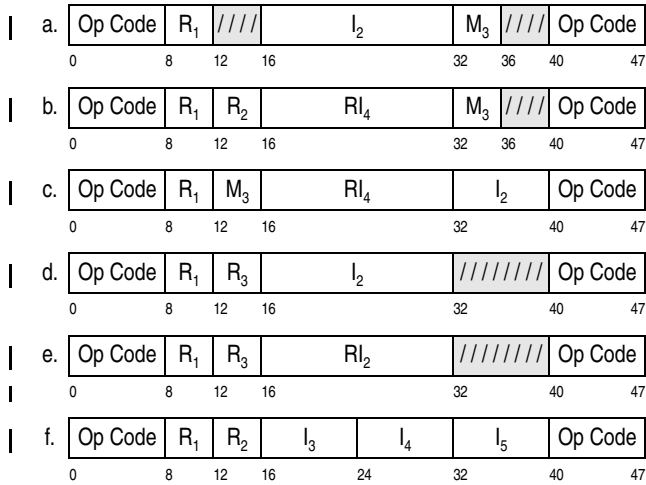
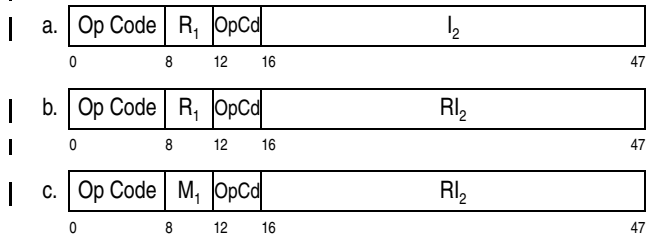


Figure 5-1. Basic Instruction Formats (Part 1 of 4)

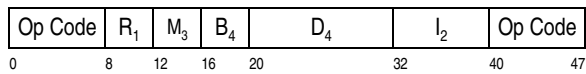
RIE Formats



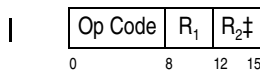
RIL Formats



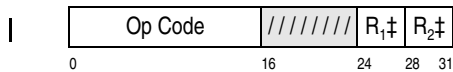
RIS Format



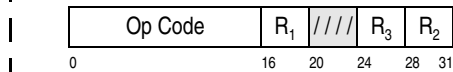
RR Format



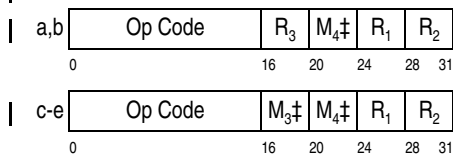
RRE Format



RRD Format



RRF Formats



RRS Format

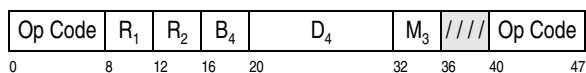
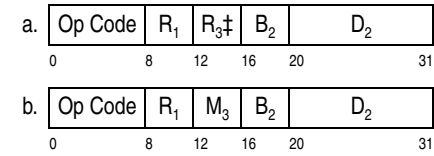
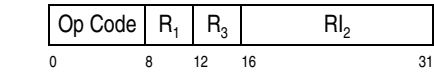


Figure 5-1. Basic Instruction Formats (Part 2 of 4)

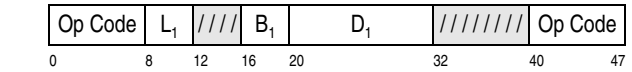
RS Formats



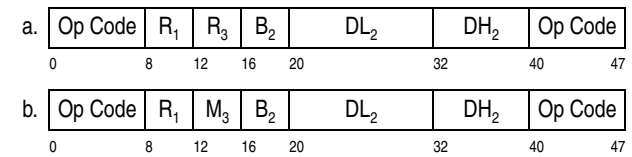
RSI Format



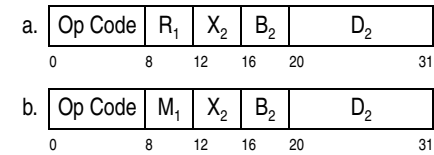
RSL Format



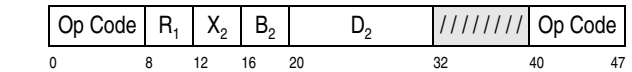
RSY Formats



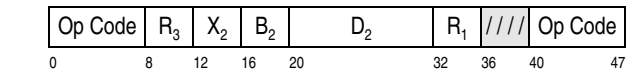
RX Formats



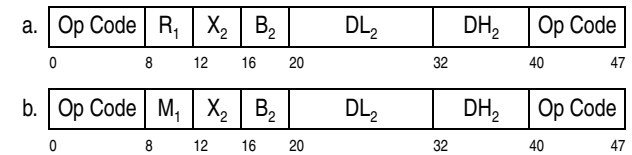
RXE Format



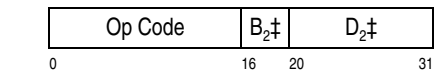
RXF Format



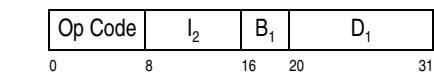
RXY Formats



S Format



SI Format

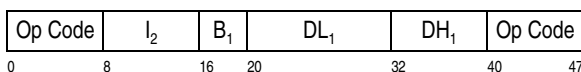


SIL Format

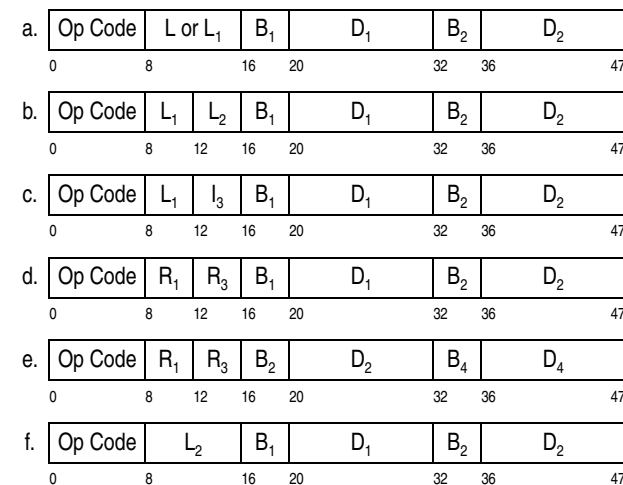


Figure 5-1. Basic Instruction Formats (Part 3 of 4)

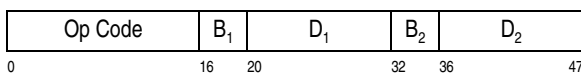
SIY Format



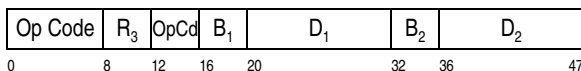
SS Formats



SSE Format



SSF Format



Explanation:

‡ For certain instructions, this operand is not defined.

Figure 5-1. Basic Instruction Formats (Part 4 of 4)

Instruction fields shown in Figure 5-1 on page 5-3 as containing slashes (/) are currently unassigned. These fields in an instruction should contain zeros; otherwise, the program may not operate compatibly in the future.

Some instructions contain fields that vary slightly from the basic format, and in some instructions the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

The format names indicate, in general terms, the classes of operands which participate in the operation and some details about fields:

- E denotes an operation using implied operands and an extended op-code field.
- I denotes an immediate operation.

- RI denotes a register-and-immediate operation and an extended op-code field.
- RIE denotes a register-and-immediate operation and a longer extended op-code field.
- RIL denotes a register-and-immediate operation, an extended op-code field, and a longer immediate field.
- RIS denotes a register-and-immediate operation and a storage operation.
- RR denotes a register-and-register operation.
- RRD denotes a register-and-register operation, an extended op-code field, and an additional R field.
- RRE denotes a register-and-register operation and an extended op-code field.
- RRF denotes a register-and-register operation, an extended op-code field, and an additional R field, M field, or both.
- RRS denotes a register-and-register operation and a storage operation.
- RS denotes a register-and-storage operation.
- RSI denotes a register-and-immediate operation.
- RSL denotes a storage operation (with an instruction format derived from the ESA/390 RSE format).
- RSY denotes a register-and-storage operation, an extended op-code field, and a long displacement field.
- RX denotes a register-and-indexed-storage operation.
- RXE denotes a register-and-indexed-storage operation and an extended op-code field.
- RXF denotes a register-and-indexed-storage operation, an extended op-code field, and an additional R field.
- RXY denotes a register-and-indexed-storage operation, an extended op-code field, and a long displacement field.
- S denotes an operation using an implied operand and storage.
- SI denotes a storage-and-immediate operation.
- SIL denotes a storage-and-immediate operation, with a 16-bit immediate field.
- SIY denotes a storage-and-immediate operation and a long displacement field.
- SS denotes a storage-and-storage operation.
- SSE denotes a storage-and-storage operation and an extended op-code field.
- SSF denotes a storage-and-storage operation and an extended op-code field.

In the I, RR, RS, RSI, RX, SI, and SS formats, the first byte of an instruction contains the op code. In the E, RRE, RRF, S, SIL, and SSE formats, the first two

bytes of an instruction contain the op code, except that for some instructions in the S format, the op code is in only the first byte. In the RI, RIL, and SSF formats, the op code is in the first byte and bit positions 12-15 of an instruction. In the RIE, RIS, RRS, RSL, RSY, RXE, RXF, RXY, and SIY formats, the op code is in the first byte and the sixth byte of an instruction.

The first two bits of the first or only byte of the op code specify the length and format of the instruction, as follows:

Bit Positions 0-1	Instruction Length (in Halfwords)	Instruction Format
00	One	I / E / RR
01	Two	RX
10	Two	RI / RRD / RRE / RRF / RS / RSI / RX / S / SI
11	Three	RIE / RIL / RIS / RRS / RSL / RSY / RXE / RXF / RXY / SIL / SIY / SS / SSE / SSF

In the format illustration for each individual instruction description, the op-code field or fields show the op code as hexadecimal digits within single quotes. The hexadecimal representation uses 0-9 for the binary codes 0000-1001 and A-F for the binary codes 1010-1111.

The remaining fields in the format illustration for each instruction are designated by code names, consisting of one or two letters and possibly a subscript number. The subscript number denotes the operand to which the field applies.

Operation code 00 hex will never be assigned to an instruction implemented in the CPU.

Register Operands

In the RR, RRD, RRE, RRF, RX, RXE, RXF, RXY, RS, RSY, RSI, RI, RIE, and RIL formats, the contents of the register designated by the R_1 field are called the first operand. The register containing the first operand is sometimes referred to as the “first-operand location,” and sometimes as “register R_1 .” In the RR, RRD, RRE, and RRF formats, the R_2 field designates the register containing the second operand, and the R_2 field may designate the same register as R_1 . In the RRD, RRF, RXF, RS, RSY, RSI, and RIE formats, the use of the R_3 field depends on the instruction. In the

RRF, RS, and RSY formats, the R_3 field may instead be an M_3 field specifying a mask. Certain forms of the RRF format include an M_4 field specifying a mask.

The R field designates a general or access register in the general instructions, a general register in the control instructions, and a floating-point register or a general register in the floating-point instructions. However, in the instructions EXTRACT STACKED REGISTERS and LOAD ADDRESS EXTENDED, the R field designates both a general register and an access register, and, in the instructions LOAD CONTROL and STORE CONTROL, the R field designates a control register. (This paragraph refers only to register operands, not to the use of access registers in addressing storage operands.)

For access and floating-point registers, unless otherwise indicated in the individual instruction description, the register operand is one register in length (32 bits for an access register and 64 bits for a floating-point register), and the second operand is the same length as the first. For general and control registers, the register operand is in bit positions 32-63 of the 64-bit register or occupies the entire register, depending on the instruction.

Programming Note: Although only two RRF formats are shown in Figure 5-1, five variations are noted. This is because there are five variations in assembler-language syntax for the RRF-format instructions, as shown below.

Variation	Assembler Syntax
a	mnemonic $R_1, R_2, [R_3, [M_4]]$
b	mnemonic $R_1, R_3, R_2, [M_4]$
c	mnemonic $R_1, R_2, [M_3]$
d	mnemonic R_1, R_2, M_4
e	mnemonic $R_1, M_3, R_2, [M_4]$

Immediate Operands

In the I format, the contents of the eight-bit immediate-data field, the I field of the instruction, are directly used as the operand.

In the SI format, the contents of the eight-bit immediate-data field, the I_2 field of the instruction, are used directly as the second operand. The B_1 and D_1 fields specify the first operand, which is one byte in length. In the SIY format, the operation is the same except that DH_1 and DL_1 fields are used instead of a D_1 field.

In the RI format for the instructions ADD HALFWORD IMMEDIATE, COMPARE HALFWORD IMMEDIATE, LOAD HALFWORD IMMEDIATE, and MULTIPLY HALFWORD IMMEDIATE, the contents of the 16-bit I_2 field of the instruction are used directly as a signed binary integer, and the R_1 field specifies the first operand, which is 32 or 64 bits in length, depending on the instruction. For the instruction TEST UNDER MASK (TMHH, TMHL, TMLH, TMLL), the contents of the I_2 field are used as a mask, and the R_1 field specifies the first operand, which is 64 bits in length.

For the instructions INSERT IMMEDIATE, AND IMMEDIATE, OR IMMEDIATE, and LOAD LOGICAL IMMEDIATE, the contents of the I_2 field are used as an unsigned binary integer or a logical value, and the R_1 field specifies the first operand, which is 64 bits in length.

For instructions in the RI, RIE, and RSI formats having an RI_2 field, or for instructions in the RIE format having an RI_4 field, the contents of the 16-bit RI_2 or RI_4 field are used as a signed binary integer designating a number of halfwords that are added to the address of the instruction to form the operand address. For instructions in the RIL format, the RI_2 field is 32 bits and is used in the same way.

For the RIE-format instructions COMPARE IMMEDIATE AND BRANCH RELATIVE and COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE, the contents of the 8-bit I_2 field are used directly as the second operand. For the RIE-format instructions COMPARE IMMEDIATE AND BRANCH, COMPARE IMMEDIATE AND TRAP, COMPARE LOGICAL IMMEDIATE AND BRANCH, and COMPARE LOGICAL IMMEDIATE AND TRAP, the contents of the 16-bit I_2 field are used directly as the second operand. For the RIE-format instructions COMPARE AND BRANCH RELATIVE, COMPARE IMMEDIATE AND BRANCH RELATIVE, COMPARE LOGICAL AND BRANCH RELATIVE, and COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE, the contents of the 16-bit RI_4 field are used as a signed binary integer designating a number of halfwords that are added to the address of the instruction to form the branch address.

For the RIL-format instructions ADD IMMEDIATE, ADD LOGICAL IMMEDIATE, ADD LOGICAL WITH SIGNED IMMEDIATE, COMPARE IMMEDIATE, COMPARE LOGICAL IMMEDIATE, LOAD IMMEDIATE, and MULTIPLY SINGLE IMMEDIATE, the con-

tents of the 32-bit I_2 field are used directly as the second operand.

For the RIS-format instructions, the contents of the 8-bit I_2 field are used directly as the second operand.

In the SIL format, the contents of the 16-bit I_2 field are used directly as the second operand. The B_1 and D_1 fields specify the first operand, as described below.

Storage Operands

The use of B and R fields to designate access registers to refer to storage operands is described in “Access-Register-Specified Address Spaces” on page 5-43.

In the RSL, SI, SIL, SSE, and most SS formats, the contents of the general register designated by the B_1 field are added to the contents of the D_1 field to form the first-operand address. In the RS, RSY, S, SIY, SS, SSE, and SSF formats, the contents of the general register designated by the B_2 field are added to the contents of the D_2 field or DH_2 and DL_2 fields to form the second-operand address. In the RX, RXE, RXF, and RXY formats, the contents of the general registers designated by the X_2 and B_2 fields are added to the contents of the D_2 field or DH_2 and DL_2 fields to form the second-operand address. In the RIS and RRS formats, and in one SS format, the contents of the general register designated by the B_4 field are added to the contents of the D_4 field to form the fourth-operand address.

When a general register contains a 24-bit or 32-bit length of a storage operand, the length is an unsigned binary integer, except that it is signed for COMPARE UNTIL SUBSTRING EQUAL, with a negative value treated as zero. Similarly, the contents of an L, L_1 , or L_2 field of an instruction are an unsigned binary integer.

In the SS format with a single, eight-bit length field, for the instructions AND (NC), EXCLUSIVE OR (XC), MOVE (MVC), MOVE NUMERICS, MOVE ZONES, and OR (OC), L specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-256, corresponding to a length code in L of 0-255. Storage results replace the first operand and are never stored outside the field specified by the address and length. In this format, the second operand has the same length as the first operand. There are variations of the preceding defini-

tion that apply to EDIT, EDIT AND MARK, PACK ASCII, PACK UNICODE, TRANSLATE, TRANSLATE AND TEST, UNPACK ASCII, and UNPACK UNICODE.

In the SS format with two length fields, and in the RSL format, L_1 specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-16, corresponding to a length code in L_1 of 0-15. Similarly, L_2 specifies the number of additional operand bytes to the right of the location designated by the second-operand address. Results replace the first operand and are never stored outside the field specified by the address and length. If the first operand is longer than the second, the second operand is extended on the left with zeros up to the length of the first operand. This extension does not modify the second operand in storage.

In the SS format with two R fields, as used by the MOVE TO PRIMARY, MOVE TO SECONDARY, and MOVE WITH KEY instructions, the contents of the general register specified by the R_1 field are a 32-bit unsigned value called the true length. The operands are both of a length called the effective length. The effective length is equal to the true length or 256, whichever is less. The instructions set the condition code to facilitate programming a loop to move the total number of bytes specified by the true length. The SS format with two R fields is also used to specify a range of registers and two storage operands for the LOAD MULTIPLE DISJOINT instruction and to specify one or two registers and one or two storage operands for the PERFORM LOCKED OPERATION instruction.

Address Generation

Trimodal Addressing

Bits 31 and 32 of the current PSW are the addressing-mode bits. Bit 31 is the extended-addressing-mode bit, and bit 32 is the basic-addressing-mode bit. These bits control the size of the effective address produced by address generation. When bits 31 and 32 of the current PSW both are zeros, the CPU is in the 24-bit addressing mode, and 24-bit instruction and operand effective addresses are generated. When bit 31 of the current PSW is zero and bit 32 is one, the CPU is in the 31-bit addressing

mode, and 31-bit instruction and operand effective addresses are generated. When bits 31 and 32 of the current PSW are both one, the CPU is in the 64-bit addressing mode, and 64-bit instruction and operand effective addresses are generated.

Execution of instructions by the CPU involves generation of the addresses of instructions and operands. This section describes address generation as it applies to most instructions. In some instructions, the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

Sequential Instruction-Address Generation

When an instruction is fetched from the location designated by the current PSW, the instruction address is increased by the number of bytes in the instruction, and the instruction is executed. The same steps are then repeated by using the new value of the instruction address to fetch the next instruction in the sequence.

In the 24-bit addressing mode, instruction addresses wrap around, with the halfword at instruction address $2^{24} - 2$ being followed by the halfword at instruction address 0. Thus, in the 24-bit addressing mode, any carry out of PSW bit position 104, as a result of updating the instruction address, is lost. In the 31-bit or 64-bit addressing mode, instruction addresses similarly wrap around, with the halfword at instruction address $2^{31} - 2$ or $2^{64} - 2$, respectively, followed by the halfword at instruction address 0. A carry out of PSW bit position 97 or 64, respectively, is lost.

Operand-Address Generation

Formation of the Intermediate Value

An operand address that refers to storage is derived from an intermediate value, which either is contained in a register designated by an R field in the instruction or is calculated from the sum of three binary numbers: base address, index, and displacement.

The base address (B) is a 64-bit number contained in a general register specified by the program in a four-bit field, called the B field, in the instruction. Base addresses can be used as a means of independently

addressing each program and data area. In array-type calculations, it can designate the location of an array, and, in record-type processing, it can identify the record. The base address provides for addressing the entire storage. The base address may also be used for indexing.

The index (X) is a 64-bit number contained in a general register designated by the program in a four-bit field, called the X field, in the instruction. It is included only in the address specified by the RX-, RXE-, RXF-, and RXY-format instructions. The RX-, RXE-, RXF-, and RXY-format instructions permit double indexing; that is, the index can be used to provide the address of an element within an array.

The displacement (D) is a 12-bit or 20-bit number contained in a field, called the D field, in the instruction. A 12-bit displacement is unsigned and provides for relative addressing of up to 4,095 bytes beyond the location designated by the base address. A 20-bit displacement is signed and provides for relative addressing of up to 524,287 bytes beyond the base-address location or of up to 524,288 bytes before it. In array-type calculations, the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

A 12-bit displacement is in bit positions 20-31 of instructions of certain formats (see Figure 5-1 on page 5-3). In instructions of some formats, a second 12-bit displacement also is in the instruction, in bit positions 36-47.

A 20-bit displacement is in instructions of only the RSY, RXY, or SIY format. In these instructions, the D field consists of a DL (low) field in bit positions 20-31 and of a DH (high) field in bit positions 32-39. When the long-displacement facility is installed, the numeric value of the displacement is formed by appending the contents of the DH field on the left of the contents of the DL field. When the long-displacement facility is not installed, the numeric value of the displacement is formed by appending eight zero bits on the left of the contents of the DL field, and the contents of the DH field are ignored.

In forming the intermediate sum, the base address and index are treated as 64-bit binary integers. A 12-bit displacement is treated as a 12-bit unsigned binary integer, and 52 zero bits are appended on the left. A 20-bit displacement is treated as a 20-bit

signed binary integer, and 44 bits equal to the sign bit are appended on the left. The three are added as 64-bit binary numbers, ignoring overflow. The sum is always 64 bits long and is used as an intermediate value to form the generated address. The bits of the intermediate value are numbered 0-63.

A zero in any of the B₁, B₂, X₂, or B₄ fields indicates the absence of the corresponding address component. For the absent component, a zero is used in forming the intermediate sum, regardless of the contents of general register 0. A displacement of zero has no special significance.

When an instruction description specifies that the contents of a general register designated by an R field are used to address an operand in storage, the register contents are used as the 64-bit intermediate value.

An instruction can designate the same general register both for address computation and as the location of an operand. Address computation is completed before registers, if any, are changed by the operation.

Unless otherwise indicated in an individual instruction definition, the generated operand address designates the leftmost byte of an operand in storage.

Formation of the Operand Address

The generated operand address is always 64 bits long, and the bits are numbered 0-63. The manner in which the generated address is obtained from the intermediate value depends on the current addressing mode. In the 24-bit addressing mode, bits 0-39 of the intermediate value are ignored, bits 0-39 of the generated address are forced to be zeros, and bits 40-63 of the intermediate value become bits 40-63 of the generated address. In the 31-bit addressing mode, bits 0-32 of the intermediate value are ignored, bits 0-32 of the generated address are forced to be zero, and bits 33-63 of the intermediate value become bits 33-63 of the generated address. In the 64-bit addressing mode, bits 0-63 of the intermediate value become bits 0-63 of the generated address.

Programming Note: Negative values may be used in index and base-address registers. Bits 0-32 of these values are ignored in the 31-bit addressing mode, and bits 0-39 are ignored in the 24-bit addressing mode.

Branch-Address Generation

Formation of the Intermediate Value

For branch instructions, the address of the next instruction to be executed when the branch is taken is called the branch address. Depending on the branch instruction, the instruction format may be RR, RRE, RX, RXY, RS, RSY, RSI, RI, RIE, or RIL.

In the RS, RSY, RX, and RXY formats, the branch address is specified by a base address, a displacement, and, in the RX and RXY formats, an index. In these formats, the generation of the intermediate value follows the same rules as for the generation of the operand-address intermediate value.

In the RR and RRE formats, the contents of the general register designated by the R_2 field are used as the intermediate value from which the branch address is formed. Unless otherwise specified, general register 0 cannot be designated as containing a branch address, and a value of zero in the R_2 field causes the instruction to be executed without branching.

The relative-branch instructions are in the RSI, RI, RIE, and RIL formats. In the RSI, RI, and RIE formats for the relative-branch instructions, the contents of the I_2 field are treated as a 16-bit signed binary integer designating a number of halfwords. In the RIL format, the contents of the I_2 field are treated as a 32-bit signed binary integer designating a number of halfwords. The branch address is the number of halfwords designated by the I_2 field added to the address of the relative-branch instruction.

The 64-bit intermediate value for a relative branch instruction in the RSI, RI, RIE, or RIL format is the sum of two addends, with overflow from bit position 0 ignored. In the RSI, RI, or RIE format, the first addend is the contents of the I_2 field with one zero bit appended on the right and 47 bits equal to the sign bit of the contents appended on the left, except that for COMPARE AND BRANCH RELATIVE, COMPARE IMMEDIATE AND BRANCH RELATIVE, COMPARE LOGICAL AND BRANCH RELATIVE and COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE, the first addend is the contents of the I_4 field, with bits appended as described above for the I_2 field. In the RIL format, the first addend is the contents of the I_2 field with one zero bit appended on the right and 31 bits equal to the sign bit of the contents appended on the left. In all formats, the second

addend is the 64-bit address of the branch instruction. The address of the branch instruction is the instruction address in the PSW before that address is updated to address the next sequential instruction, or it is the address of the target of the execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG) if an execute-type instruction is used. If an execute-type instruction is used in the 24-bit or 31-bit addressing mode, the address of the branch instruction is the target address with 40 or 33 zeros, respectively, appended on the left.

Formation of the Branch Address

The branch address is always 64 bits long, with the bits numbered 0-63. The branch address replaces bits 64-127 of the current PSW.

The manner in which the branch address is obtained from the intermediate value depends on the addressing mode. For those branch instructions which change the addressing mode, the new addressing mode is used. In the 24-bit addressing mode, bits 0-39 of the intermediate value are ignored, bits 0-39 of the branch address are made zeros, and bits 40-63 of the intermediate value become bits 40-63 of the branch address. In the 31-bit addressing mode, bits 0-32 of the intermediate value are ignored, bits 0-32 of the branch address are made zeros, and bits 33-63 of the intermediate value become bits 33-63 of the branch address. In the 64-bit addressing mode, bits 0-63 of the intermediate value become bits 0-63 of the branch address.

For several branch instructions, branching depends on satisfying a specified condition. When the condition is not satisfied, the branch is not taken, normal sequential instruction execution continues, and the branch address is not used. When a branch is taken, bits 0-63 of the branch address replace bits 64-127 of the current PSW. The branch address is not used to access storage as part of the branch operation.

A specification exception due to an odd branch address and access exceptions due to fetching of the instruction at the branch location are not recognized as part of the branch operation but instead are recognized as exceptions associated with the execution of the instruction at the branch location.

A branch instruction, such as BRANCH AND SAVE, can designate the same general register for branch-address computation and as the location of an oper-

and. Branch-address computation is completed before the remainder of the operation is performed.

Instruction Execution and Sequencing

The program-status word (PSW), described in Chapter 4 “Control” contains information required for proper program execution. The PSW is used to control instruction sequencing and to hold and indicate the status of the CPU in relation to the program currently being executed. The active or controlling PSW is called the current PSW.

Branch instructions perform the functions of decision making, loop control, and subroutine linkage. A branch instruction affects instruction sequencing by introducing a new instruction address into the current PSW. The relative-branch instructions with a 16-bit I_2 field allow branching to a location at an offset of up to plus 64K - 2 bytes or minus 64K bytes relative to the location of the branch instruction, without the use of a base register. The relative-branch instructions with a 32-bit I_2 field allow branching to a location at an offset of up to plus 4G - 2 bytes or minus 4G bytes relative to the location of the branch instruction, without the use of a base register.

Decision Making

Facilities for decision making are provided by the BRANCH ON CONDITION, BRANCH RELATIVE ON CONDITION, and BRANCH RELATIVE ON CONDITION LONG instructions. These instructions inspect a condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. The condition code, which consists of two bits, provides for four possible condition-code settings: 0, 1, 2, and 3.

The specific meaning of any setting depends on the operation that sets the condition code. For example, the condition code reflects such conditions as zero, nonzero, first operand high, equal, overflow, and sub-channel busy. Once set, the condition code remains unchanged until modified by an instruction that causes a different condition code to be set. See Appendix C, “Condition-Code Settings” for a summary of the instructions which set the condition code.

Loop Control

Loop control can be performed by the use of BRANCH ON CONDITION, BRANCH RELATIVE ON CONDITION, and BRANCH RELATIVE ON CONDITION LONG to test the outcome of address arithmetic and counting operations. For some particularly frequent combinations of arithmetic and tests, BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL are provided, and relative-branch equivalents of these instructions are also provided. These branches, being specialized, provide increased performance for these tasks.

Subroutine Linkage without the Linkage Stack

This section describes only the methods for subroutine linkage that do not use the linkage stack. For the linkage extensions provided by the linkage stack, see “Linkage-Stack Introduction” on page 5-66. (Those extensions include a different method of operation of the PROGRAM CALL instruction and also the BRANCH AND STACK and PROGRAM RETURN instructions.)

Simple Branch Instructions

Subroutine linkage when a change of the addressing mode is not required is provided by the BRANCH AND LINK and BRANCH AND SAVE instructions. (This discussion of BRANCH AND SAVE applies also to BRANCH RELATIVE AND SAVE and BRANCH RELATIVE AND SAVE LONG.) Both of these instructions permit not only the introduction of a new instruction address but also the preservation of a return address and associated information. The return address is the address of the instruction following the branch instruction in storage, except that it is the address of the instruction following an execute-type instruction that has the branch instruction as its target.

Both BRANCH AND LINK and BRANCH AND SAVE have an R_1 field. They form a branch address by means of fields that depend on the instruction. The operations of the instructions are summarized as follows:

- In the 24-bit addressing mode, both instructions place the return address in bit positions 40-63 of general register R_1 and leave bits 0-31 of that

register unchanged. **BRANCH AND LINK** places the instruction-length code for the instruction and also the condition code and program mask from the current PSW in bit positions 32-39 of general register R_1 . **BRANCH AND SAVE** places zeros in those bit positions.

- In the 31-bit addressing mode, both instructions place the return address in bit positions 33-63 and a one in bit position 32 of general register R_1 , and they leave bits 0-31 of the register unchanged.
- In the 64-bit addressing mode, both instructions place the return address in bit positions 0-63 of general register R_1 .
- In any addressing mode, both instructions generate the branch address under the control of the current addressing mode. The instructions place bits 0-63 of the branch address in bit positions 64-127 of the PSW. In the RR format, both instructions do not perform branching if the R_2 field of the instruction is zero.

It can be seen that, in the 24-bit or 31-bit addressing mode, **BRANCH AND SAVE** places the basic-addressing-mode bit, bit 32 of the PSW, in bit position 32 of general register R_1 . **BRANCH AND LINK** does so in the 31-bit addressing mode.

The instructions **BRANCH AND SAVE AND SET MODE** and **BRANCH AND SET MODE** are for use when a change of the addressing mode is required during linkage. These instructions have R_1 and R_2 fields. The operations of the instructions are summarized as follows:

- **BRANCH AND SAVE AND SET MODE** sets the contents of general register R_1 the same as **BRANCH AND SAVE**. In addition, the instruction places the extended-addressing-mode bit, bit 31 of the PSW, in bit position 63 of the register.
- **BRANCH AND SET MODE**, if R_1 is nonzero, performs as follows. In the 24- or 31-bit mode, it places bit 32 of the PSW in bit position 32 of general register R_1 , and it leaves bits 0-31 and 33-63 of the register unchanged. Note that bit 63 of the register should be zero if the register contains an instruction address. In the 64-bit mode, the instruction places bit 31 of the PSW (a one) in bit position 63 of general register R_1 , and it leaves bits 0-62 of the register unchanged.

- When R_2 is nonzero, both instructions set the addressing mode and perform branching as follows. Bit 63 of general register R_2 is placed in bit position 31 of the PSW. If bit 63 is zero, bit 32 of the register is placed in bit position 32 of the PSW. If bit 63 is one, PSW bit 32 is set to one. Then the branch address is generated from the contents of the register, except with bit 63 of the register treated as a zero, under the control of the new addressing mode. The instructions place bits 0-63 of the branch address in bit positions 64-127 of the PSW. Bit 63 of general register R_2 remains unchanged and, therefore, may be one upon entry to the called program. If R_2 is the same as R_1 , the results in the designated general register are as specified for the R_1 register.

The operations of the simple branch instructions are summarized in Figure 5-2 on page 5-13. For contrast, the figure also shows the **BRANCH ON COUNT** instruction, which is not for use in linkage, and the **LOAD ADDRESS** and **LOAD ADDRESS EXTENDED** instructions.

Programming Notes:

1. A called program that is entered in the 64-bit addressing mode can use bit 63 of the entry-point register to determine the instruction used to perform the call and, thus, the instruction that must be used to perform the return linkage. If bit 63 is zero, **BRANCH AND SAVE** (BAS or BASR) (or possibly BAL, BALR, BRAS, or BRASL) was used, the addressing mode of the caller is not indicated in the return register, and **BRANCH ON CONDITION** (BCR) must be used to return without changing the addressing mode during the return. If bit 63 of the entry-point register is one, BASSM or BSM was used, the addressing mode of the caller is indicated in the return register (or at least can be, in the case of BSM), and BSM must be used to return and restore the addressing mode of the caller.
2. When BSM is executed in the 24-bit or 31-bit addressing mode and used in a forward linkage to set the 64-bit mode, and the R_1 and R_2 of the instruction are the same value, bit 63 of the designated general register does not, upon entry to the called program, correctly indicate the mode of the calling program. (The bit is one, instead of zero, because the program set bit 63 of the R_2 register to one and the instruction does not change bit 63 of the R_1 register in the 24- or

Instruction	Format	In Mode	Address Placed in GR R ₁				Branch or 2nd-Op Address		R ₂ Bit 63	PSW Bit 31 Set to	PSW Bit 32 Set to
			Bits 0-31	Bit 32	Bits 33-62	Bits 63	Bits 0-32	Bits 33-63			
BALR*/BAL	RR/RX	24	U	***	***	IA	SIA	SIA	LSExc	U	U
		31	U	BAM	IA	IA	SIA	SIA	LSExc	U	U
		64	IA	IA	IA	IA	SIA	SIA	LSExc	U	U
BASR*/BAS/ BRAS/BRASL	RR/RX/ RI/RIL	24/31	U	BAM	IA	IA	SIA	SIA	LSExc	U	U
		64	IA	IA	IA	IA	SIA	SIA	LSExc	U	U
BASSM*	RR	24/31	U	BAM	IA	IA	SIA	SIA	0	0	R ₂ 32
		24/31	U	BAM	IA	IA	SIA	SIA	1G0	1	1
		64	IA	IA	IA	1	SIA	SIA	0	0	R ₂ 32
BSM**	RR	64	IA	IA	IA	1	SIA	SIA	1G0	1	1
		24/31	U	BAM	U	U	SIA	SIA	0	0	R ₂ 32
		24/31	U	BAM	U	U	SIA	SIA	1G0	1	1
BCTR*/BCT/ BCTGR*/BCTG LA/LAE	RR/RX/ RRE/RXY RX/RX	64	U	U	U	1	SIA	SIA	0	0	R ₂ 32
		24/31	NLA	NLA	NLA	NLA	SIA	SIA	1G0	1	1
		24/31	NLA	NLA	NLA	NLA	SIA	SIA	LSExc	U	U
		64	NLA	NLA	NLA	NLA	SIA	SIA	LSExc	U	U
		24/31	U	0	Op2Ad	Op2Ad	FZ	SR1	0/1	U	U
		64	Op2Ad	Op2Ad	Op2Ad	Op2Ad	SR1	SR1	0/1	U	U

Explanation:

—	The address does not exist, or the bit has no special effect.
*	The action associated with the R ₂ field is not performed if the field is zero.
**	The action associated with the R ₁ or R ₂ field is not performed if the field is zero.
***	The instruction-length code, condition code, and program mask are saved in bit positions 32-39 of the link address, and bits 40-63 of the updated instruction address are saved in bit positions 40-63.
0/1	Bit 63 can be zero or one.
1G0	Bit 63 is one and is left one, but the branch address is generated as if the bit is zero.
BAM	Bit 32 of the link address is set with the basic-addressing-mode bit, bit 32 of the PSW.
FZ	Bits 0-32 of the second-operand address are forced to zeros in the 24-bit or 31-bit addressing mode.
IA	Bits of the link address are set with the updated instruction address as shown.
LSExc	A late specification exception is recognized if the bit is one.
NLA	The instruction does not produce a link address. (The instruction is shown simply as an example of a non-linkage branch instruction.)
Op2Ad	Bits of the address in general register R ₁ are set with the corresponding bits of the second-operand address as shown.
R ₂ 32	The basic-addressing-mode bit, bit 32 of the PSW, is set with bit 32 of general register R ₂ .
SIA	Bits 0-63 of the branch address are used to set the instruction address in the PSW. Bits 0-39 of the branch address are forced to zeros in the 24-bit addressing mode. Bits 0-32 are forced to zeros in the 31-bit addressing mode.
SR1	Bits of the second-operand address are used to set the corresponding bits of the address in the R ₁ general register as shown. Bits 0-39 of the second-operand address are forced to zeros in the 24-bit addressing mode. Bits 0-32 are forced to zeros in the 31-bit addressing mode.
U	Unchanged.

Figure 5-2. Summary of Simple Branch Linkage Instructions and Other Instructions

31-bit mode.) BASSM always correctly indicates the addressing mode of the calling program.

- If an entry point can be branched to in the 64-bit addressing mode either by BAS or BASR or by BASSM or BSM, and a USING statement pro-

vides addressability to the entry point, bit 63 of the entry-point register must be zeroed to ensure compatible operation regardless of whether the linkage instruction changes the addressing mode. For example, if general register 15 is the

entry-point register, the following may be used to set bit 63 to zero.

```
NILL 15,X'FFFE'
```

If the entry point can be branched to in the 64-bit addressing mode only by BASSM or BSM, the entry-point USING statement can accommodate bit 63 being one – without having to zero the bit – as shown below.

```
USING *+1,15
```

Other Linkage Instructions

Note: The discussion in this section of PROGRAM TRANSFER applies equally to PROGRAM TRANSFER WITH INSTANCE, which differs from PROGRAM TRANSFER only as described in “ASN-Second-Table-Entry Instance Number and ASN Reuse” on page 3-21.

PROGRAM TRANSFER WITH INSTANCE is not further referred to in this section.

Linkage between a problem-state program and the supervisor or monitoring program is provided by means of the SUPERVISOR CALL and MONITOR CALL instructions.

The instructions PROGRAM CALL and PROGRAM TRANSFER provide the facility for linkage between programs of different authority and in different address spaces. PROGRAM CALL permits linkage to a number of preassigned programs that may be in either the problem or the supervisor state and may be in either the same address space or an address space different from that of the caller. It permits a change between the 24-bit and 31-bit addressing modes, and it permits an increase of PSW-key-mask authority, which authorizes the execution of the SET PSW KEY FROM ADDRESS instruction and also other functions. In general, PROGRAM CALL is used to transfer control to a program of higher authority. PROGRAM TRANSFER permits a change of the instruction address and address space and a change between the 24-bit and 31-bit addressing modes. PROGRAM TRANSFER also permits a reduction of PSW-key-mask authority and a change from the supervisor to the problem state. In general, it is used to transfer control from one program to another of equal or lower authority.

When a calling linkage is to increase authority, the calling linkage can be performed by PROGRAM

CALL and the return linkage by PROGRAM TRANSFER. Alternatively, when the calling linkage is to decrease authority, the calling linkage can be performed by PROGRAM TRANSFER and the return linkage by PROGRAM CALL.

The operation of PROGRAM CALL is controlled by means of an entry-table entry, which is located as part of a table-lookup process during the execution of the instruction. The entry-table entry specifies either a basic (nonstacking) operation or the stacking operation described in “Linkage-Stack Introduction” on page 5-66. The instruction causes the primary address space to be changed only when the ASN in the entry-table entry is nonzero. When the primary address space is changed, the operation is called PROGRAM CALL with space switching (PC-ss). When the primary address space is not changed, the operation is called PROGRAM CALL to current primary (PC-cp).

PROGRAM TRANSFER specifies the address space which is to become the new primary address space. When the primary address space is changed, the operation is called PROGRAM TRANSFER with space switching (PT-ss). When the primary address space is not changed, the operation is called PROGRAM TRANSFER to current primary (PT-cp).

Basic PROGRAM CALL, and PROGRAM TRANSFER, can be executed successfully in either a basic (24-bit or 31-bit) addressing mode or the extended (64-bit) addressing mode. They do not provide a change between a basic addressing mode and the extended addressing mode.

The BRANCH AND SET AUTHORITY instruction can improve performance by replacing a PT-cp instruction used to perform a calling linkage in which PSW-key-mask authority is reduced, and by replacing a PC-cp instruction used to perform the associated return linkage in which PSW-key-mask authority is restored. BRANCH AND SET AUTHORITY also permits changes between the supervisor and problem states, and it can replace SET PSW KEY FROM ADDRESS by changing the PSW key during the linkage. The calling-linkage operation is called BRANCH AND SET AUTHORITY in the base-authority state (BSA-ba), and the return-linkage operation is called BRANCH AND SET AUTHORITY in the reduced-authority state (BSA-ra).

The BRANCH IN SUBSPACE GROUP instruction allows linkage within a group of address spaces

called a subspace group, where one address space in the group is called the base space and the others are called subspaces. It is intended that each subspace contain a different subset of the storage in the base space, that the base space and each subspace contain a subsystem control program, such as CICS, and application programs, and that each subspace contain the data for a single transaction being processed under the subsystem control program. The placement of the data for each transaction in a different subspace prevents a program that is being executed to process one particular transaction from erroneously damaging the data of other transactions. It is intended that the primary address space be the base space when the control program is being executed, and that it be the subspace for a transaction when an application program is being executed to process that transaction. **BRANCH IN SUBSPACE GROUP** changes not only the instruction address in the PSW but also the primary address-space-control element in control register 1. **BRANCH IN SUBSPACE GROUP** does not change the primary ASN in control register 4 or the primary-ASN-second-table-entry origin in control register 5, and, therefore, the base space and the subspaces all are associated with the same ASN, and the programs in those address spaces all are of equal authority.

Although a subspace is intended to be a subset of the base space as described above, **BRANCH IN SUBSPACE GROUP** does not require this, and the instruction may be useful in ways other than as described above.

BRANCH IN SUBSPACE GROUP uses an access-list-entry token (ALET) in an access register as an identifier of the address space that is to receive control. The instruction saves the updated instruction address to permit a return linkage, but it does not save an identifier of the address space from which control was transferred. However, an ALET equal to 00000000 hex, called ALET 0, can be used to return from a subspace to the base space, and an ALET equal to 00000001 hex, called ALET 1, can be used to return from the base space to the subspace that last had control.

The **SET ADDRESSING MODE** (SAM24, SAM31, SAM64) instruction can assist in linkage by setting the 24-bit, 31-bit, or 64-bit addressing mode either before or after a linkage operation.

The **RESUME PROGRAM** instruction is intended for use by a problem-state interruption-handling program

to return to the interrupted program. The interruption-handling program can use **LOAD ACCESS MULTIPLE** and **LOAD MULTIPLE** instructions to restore the contents of the interrupted program's access and general registers from a save area, except for the contents of one access-and-general register pair. The interruption-handling program then can use **RESUME PROGRAM** to restore the contents of certain PSW fields, including the instruction address, and also the contents of the remaining access-and-general pair from the save area, with that pair first being used by **RESUME PROGRAM** to address the save area.

The **TRAP** instruction (**TRAP2**, **TRAP4**) can overlay instructions in an application program and give control to a trap program for performing fix-ups of data used by the application program. The **RESUME PROGRAM** instruction can be used to return control from the trap program to the application program.

The linkage instructions provided and the functions performed by each are summarized in Figure 5-3 on page 5-16.

Programming Note: This note describes the simple branch-type linkage instructions that were included in 370-XA and carried forward to ESA/370, ESA/390, and z/Architecture. To give the reader a better understanding of the utility and intended usage of these linkage instructions, the following paragraphs in this note describe various program linkages and conventions and the use of the linkage instructions in these situations.

The linkage instructions were originally provided to permit System/370 programs to operate with no modification or only slight modification on 370/XA (and successor) systems and also to provide additional function for those programs which were designed to take advantage of the 31-bit addressing of 370/XA. The instructions provided the capability for both old and new programs to coexist in storage and to communicate with each other. The instructions now have been enhanced to permit usage of the 64-bit addressing of z/Architecture.

With respect to System/370 programs, it is assumed that old, unmodified programs operate in the 24-bit addressing mode and call, or directly communicate with, other programs operating in the 24-bit addressing mode only. Modified programs normally operate in the 24-bit addressing mode but may have called programs which operate in either the 24-bit or 31-bit

Instruction	Format	Instruction Address PSW Bits 64-127		Basic Addr. Mode PSW Bit 32		Extended Addr. Mode PSW Bit 31		Problem State PSW Bit 15		PASN CR4 Bits 48-63		PSW-Key Mask Changed in CR3	Trace
		Save	Set	Save	Set	Save	Set	Save	Set	Save	Set		
BALR	RR	Yes*	R ₂ ¹	BAM31	-	-	-	-	-	-	-	-	R ₂ ¹
BAL	RX	Yes*	Yes*	BAM31	-	-	-	-	-	-	-	-	-
BASR	RR	Yes	R ₂ ¹	BAM	-	-	-	-	-	-	-	-	R ₂ ¹
BAS	RX	Yes	Yes	BAM	-	-	-	-	-	-	-	-	-
BASSM	RR	Yes	R ₂ ¹	BAM	R ₂ ¹	Yes	R ₂ ¹	-	-	-	-	-	-
BRAS	RI	Yes	Yes	BAM	-	-	-	-	-	-	-	-	-
BRASL	RIL	Yes	Yes	BAM	-	-	-	-	-	-	-	-	-
BSA-ba	RRE	Yes	Yes	BAM	BAM	-	-	Yes	Yes ⁴	-	-	"AND" R ₁ ⁵	Yes
BSA-ra	RRE	R ₁ ¹	Yes	R ₁ ¹ BAM	BAM	-	-	-	Yes	-	-	Yes	Yes
BSG	RRE	Yes	Yes	R ₁ ¹ BAM	BAM	-	-	-	-	-	- ³	-	Yes
BSM	RR	-	R ₂ ¹	R ₁ ¹ BAM	R ₂ ¹	R ₁ ¹ EAM64	R ₂ ¹	-	-	-	-	-	-
MC# ²	SI	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-	-	-	-
PC-cp	S	Yes	Yes	BAM	BAM	-	-	Yes	Yes	-	-	"OR" EKM	Yes
PC-ss	S	Yes	Yes	BAM	BAM	-	-	Yes	Yes	Yes	Yes	"OR" EKM	Yes
PT-cp or PTI-cp	RRE	-	R ₂	-	R ₂ BAM	-	-	-	R ₂ ^{**}	-	-	"AND" R ₁	Yes
PT-ss or PTI-ss	RRE	-	R ₂	-	R ₂ BAM	-	-	-	R ₂ ^{**}	-	Yes	"AND" R ₁	Yes
RP	S	-	Yes	-	Yes	-	Yes	-	-	-	-	-	Yes
SAM24	E	-	-	-	Yes 0	-	Yes 0	-	-	-	-	-	Yes
SAM31	E	-	-	-	Yes 1	-	Yes 0	-	-	-	-	-	Yes
SAM64	E	-	-	-	Yes 1	-	Yes 1	-	-	-	-	-	Yes
SVC2	RR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-	-	-	-
TRAP2	E	Yes	Yes	Yes	Yes	Yes	-	Yes	-	-	-	-	Yes
TRAP4	S	Yes	Yes	Yes	Yes	Yes	-	Yes	-	-	-	-	Yes

Figure 5-3. Summary of Linkage Instructions without the Linkage Stack

Instruction	Format	Instruction Address PSW Bits 64-127		Basic Addr. Mode PSW Bit 32		Extended Addr. Mode PSW Bit 31		Problem State PSW Bit 15		PASN CR4 Bits 48-63		PSW-Key Mask Changed in CR3	Trace
		Save	Set	Save	Set	Save	Set	Save	Set	Save	Set		
Explanation:													
-	No												
*	In the 24-bit addressing mode, the instruction-length code, condition code, and program mask are saved in bit positions 32-39 of the R ₁ general register.												
**	A change from the supervisor to the problem state is allowed; a privileged-operation exception is recognized when a change from the problem to the supervisor state is specified.												
#	Monitor-mask bits provide a means of disallowing linkage, or enabling linkage, for selected classes of events. When the enhanced-monitor facility is installed, the enhanced-monitor-mask bits are also used in controlling linkage (by means of a monitor-event program interruption).												
1	The action takes place only if the associated R field in the instruction is nonzero.												
2	MC and SVC, as part of the interruption, save the entire current PSW and load a new PSW.												
3	The primary address-space-control element is set even though the PASN is not set.												
4	The problem state is set												
5	The PSW key also is set from general register R ₁												
BAM	The basic-addressing-mode bit is saved or set only in the 24-bit or 31-bit addressing mode.												
BAM31	The basic-addressing-mode bit is saved only in the 31-bit addressing mode.												
BAM64	The extended-addressing-mode bit is saved only in the 64-bit addressing mode.												
R ₁	The field or bit is saved in general register R ₁ .												
R ₂	The field or bit is set from general register R ₂ .												

Figure 5-3. Summary of Linkage Instructions without the Linkage Stack

addressing mode. They and also modified 370-XA, ESA/370, and ESA/390 programs now may call programs that operate in the 24-bit, 31-bit, or 64-bit addressing mode. New programs may be written to operate in any addressing mode, and, in some cases, a program may be written such that it can be invoked in any addressing mode.

BRANCH AND SAVE AND SET MODE (BASSM) is intended to be the principal calling instruction to sub-routines outside of an assembler/linkage-editor control section (CSECT), for use by all new programs and particularly by programs that must change the addressing mode during the linkage. The calling sequence has normally been:

```

L      15,ACON
BASSM 14,15
...
EXTRN SUB
ACON DC A(X'80000000'+SUB)

```

where ACON is an A-type address constant, and the X'80000000' should be present to give control in the 31-bit addressing mode or should be omitted to give control in the 24-bit addressing mode.

The return from such a routine normally is:

```
BSM 0,14
```

It is assumed that the A-type address constant will be extended so it may be an eight-byte field containing a 64-bit entry-point address, with bit 63 of the address indicating, when one, that the entry is in the 64-bit addressing mode. This extended constant is shown here as "ACONE". The calling sequence would normally be:

```

LG      15,ACONE
BASSM 14,15
...
EXTRN SUB
ACONE DC AD(X'1'+SUB)

```

The return from such a routine would normally be:

```
BSM 0,14
```

When a change of the addressing mode is not required, BRANCH AND LINK or BRANCH AND SAVE should be used instead of BASSM.

The BRANCH AND LINK (BAL, BALR) instruction is provided primarily for compatibility with System/370.

It is defined to operate in the 31-bit and 64-bit addressing modes to increase the probability that an old, straightforward program can be modified to operate in those addressing modes with minimal or no change. It is recommended, however, that **BRANCH AND SAVE (BAS and BASR)** be used instead and that **BRANCH AND LINK** be avoided since it places nonzero information in bit positions 32-39 of the general register in the 24-bit addressing mode, which may lead to problems and may decrease performance.

BRANCH RELATIVE AND SAVE and **BRANCH RELATIVE AND SAVE LONG** may be used instead of **BRANCH AND SAVE**.

It is assumed that the normal return from a subroutine called in the 24-bit or 31-bit addressing mode by **BRANCH AND LINK (BAL or BALR)** will be:

BCR 15,14

However, the standard “return instruction”:

BSM 0,14

operates correctly for all cases except for a calling **BAL** executed in the 24-bit addressing mode. In the 24-bit addressing mode, **BAL** causes an ILC of 10 to be placed in bit positions 32 and 33 of the link register. Thus, a **BSM** would return in the 31-bit addressing mode. Note that an **EXECUTE** of **BALR** in the 24-bit addressing mode also causes the same ILC effect; an **EXECUTE RELATIVE LONG** of **BALR** in the 24-bit addressing mode causes a similar ILC effect (the ILC is 11 binary).

The **BRANCH AND SAVE (BAS, BASR)** instruction is provided to be used for subroutine linkage to any program either within the same CSECT or known to be in the same addressing mode. **BASR** with the R_2 field 0 is also useful for obtaining addressability to the instruction stream by getting a 31-bit address, uncluttered by leftmost fields, in the 24-bit addressing mode.

The instruction for returning from a routine called in the 24-bit or 31-bit addressing mode by **BRANCH AND SAVE (BAS or BASR)** may be either:

BCR 15,14

or:

BSM 0,14

The instruction for returning from a routine called in the 64-bit addressing mode by **BAS** or **BASR** must be **BCR**; **BSM** would set the 24-bit or 31-bit addressing mode, depending on bit 32 of the link register (an address bit), because bit 63 of the link register (the rightmost bit of an instruction address) is zero. **BSM** can always be used as the return instruction if **BASSM** is used as the calling instruction.

In some cases, it may be desirable to rewrite a program that is called by an old program which has not been rewritten. In such a case, the old program, which operates in the 24-bit or 31-bit addressing mode, will be given the address of an intermediate program that will set up the correct entry and return modes and then call the rewritten program. Such an intermediate program is sometimes referred to as a *glue module*. The instruction **BRANCH AND SET MODE (BSM)** with a nonzero R_1 field provides the function necessary to perform this operation efficiently. This is shown in Figure 5-4 on page 5-19 for a linkage from a 24-bit-mode program to a 31-bit-mode program.

Note that the “**BSM 14,15**” in the glue module causes either an indication of the 64-bit addressing mode to be saved in bit position 63 of general register 14 or an indication of one of the 24-bit and 31-bit addressing modes to be saved in bit position 32 of the register, and that the other bits of the register are unchanged. Thus, when “**BSM 0,14**” is executed in the new program, control passes directly back to the old program without passing through the glue module again. The glue module could give control to a program in the 64-bit addressing mode and possibly above the 2G-byte boundary by loading an eight-byte

A-type address constant, with bit 63 set to one, instead of a four-byte A-type address constant.

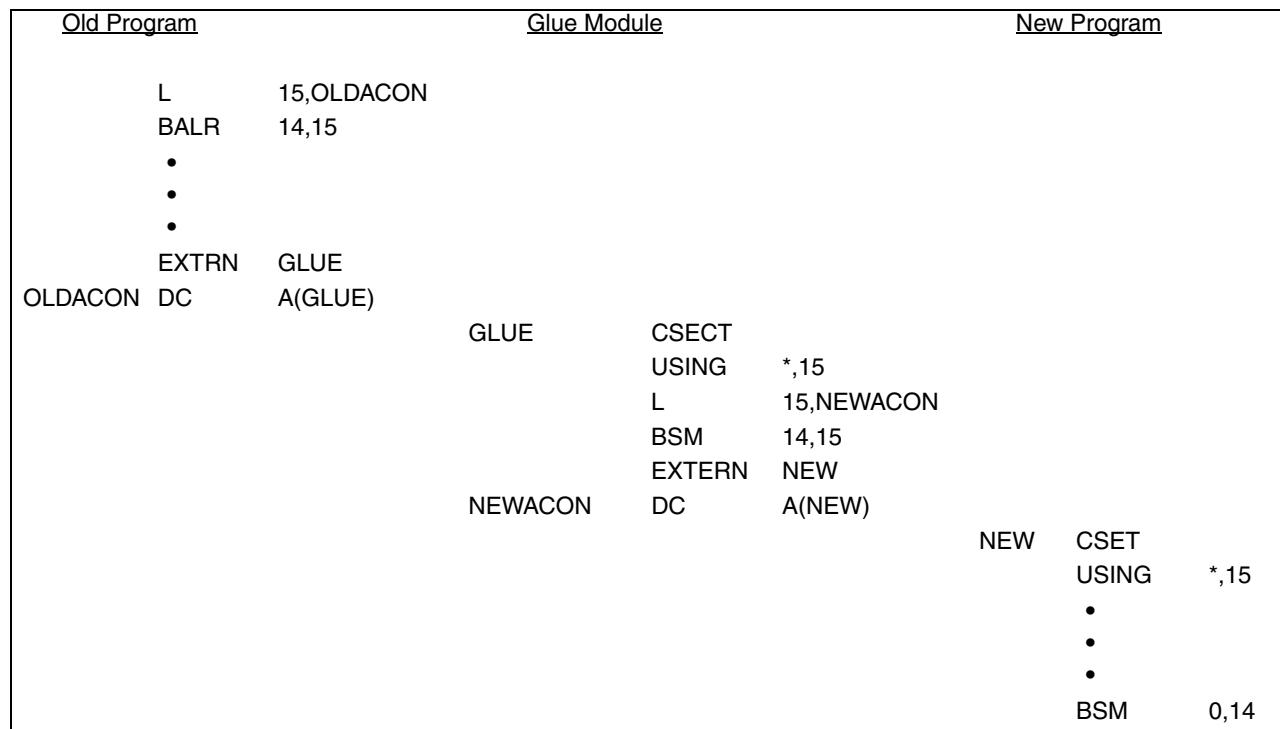


Figure 5-4. Glue Module for Linkage from the 24-Bit Mode to the 31-Bit Mode

Interruptions

Interruptions permit the CPU to change state as a result of conditions external to the system, in sub-channels or input/output (I/O) devices, in other CPUs, or in the CPU itself. Details are to be found in Chapter 6, “interruptions”.

Six classes of interruption conditions are provided: external, I/O, machine check, program, restart, and supervisor call. Each class has two related PSWs, called old and new, in permanently assigned real storage locations. In all classes, an interruption involves storing information identifying the cause of the interruption, storing the current PSW at the old-PSW location, and fetching the PSW at the new-PSW location, which becomes the current PSW.

The old PSW contains CPU-status information necessary for resumption of the interrupted program. At the conclusion of the program invoked by the interruption, the instruction LOAD PSW EXTENDED may be used to restore the current PSW to the value of the old PSW.

Types of Instruction Ending

Instruction execution ends in one of five ways: completion, nullification, suppression, termination, and partial completion.

Partial completion of instruction execution occurs only for interruptible instructions; it is described in “Interruptible Instructions” on page 5-20.

Completion

Completion of instruction execution provides results as called for in the definition of the instruction. When an interruption occurs after the completion of the execution of an instruction, the instruction address in the old PSW designates the next sequential instruction.

Suppression

Except as noted below, suppression of instruction execution causes the instruction to be executed as if it specified “no operation.” The contents of any result fields, including the condition code, are not changed. The instruction address in the old PSW on an inter-

ruption after suppression designates the next sequential instruction.

When (a) the AFP-register (additional floating-point register) control bit, bit 45 of control register 0, is one, (b) either an IEEE-invalid-operation condition or an IEEE-division-by-zero condition is recognized, and (c) the respective IEEE mask bit in the floating-point-control (FPC) register is one, then the resulting data-exception program interruption is considered to be suppressing, even though the FPC is altered. See “Data-Exception Code (DXC)” on page 6-14 for details.

Nullification

Nullification of instruction execution has the same effect as suppression, except that when an interruption occurs after the execution of an instruction has been nullified, the instruction address in the old PSW designates the instruction whose execution was nullified (or an execute-type instruction, as appropriate) instead of the next sequential instruction.

Termination

Termination of instruction execution causes the contents of any fields due to be changed by the instruction to be unpredictable. The operation may replace all, part, or none of the contents of the designated result fields and may change the condition code if such change is called for by the instruction. Unless the interruption is caused by a machine-check condition, the validity of the instruction address in the PSW, the interruption code, and the ILC are not affected, and the state or the operation of the machine is not affected in any other way. The instruction address in the old PSW on an interruption after termination designates the next sequential instruction.

Programming Note: Although the execution of an instruction is treated as a no-operation when suppression or nullification occurs, stores may be performed as the result of the implicit tracing action associated with some instructions. See “Tracing” on page 4-10.

Interruptible Instructions

Point of Interruption

For most instructions, the entire execution of an instruction is one operation. An interruption is permitted between operations; that is, an interruption can

occur after the performance of one operation and before the start of a subsequent operation.

For the following instructions, referred to as interruptible instructions, an interruption is permitted also after partial completion of the instruction:

- COMPARE AND FORM CODEWORD
- COMPARE LOGICAL LONG
- COMPARE UNTIL SUBSTRING EQUAL
- COMPRESSION CALL
- INVALIDATE PAGE TABLE ENTRY (when the IPTe-range facility is installed, and the R_3 field is nonzero)
- MOVE LONG
- PERFORM FRAME MANAGEMENT FUNCTION (when the enhanced-DAT facility is installed, and the frame-size code designates a 1M-byte frame)
- SET STORAGE KEY EXTENDED (when the enhanced-DAT facility is installed, and the multiple-block control is one)
- TEST BLOCK
- UPDATE TREE

Unit of Operation

Whenever points of interruption that include those occurring within the execution of an interruptible instruction are discussed, the term “unit of operation” is used. For a noninterruptible instruction, the entire execution consists, in effect, in the execution of one unit of operation.

The execution of an interruptible instruction is considered to consist in the execution of a number of units of operation, and an interruption is permitted between units of operation. The amount of data processed in a unit of operation depends on the particular instruction and may depend on the model and on the particular condition that causes the execution of the instruction to be interrupted.

When an instruction execution consists of a number of units of operation and an interruption occurs after some, but not all, units of operation have been completed, the instruction is said to be partially completed. In this case, the type of ending (completion, nullification, or suppression) is associated with the unit of operation. In the case of termination, the entire instruction is terminated, not just the unit of operation.

An exception may exist that causes the first unit of operation of an interruptible instruction not to be

completed. In this case when the ending is nullification or suppression, all operand parameters and result locations remain unchanged, except that the condition code is unpredictable if the instruction is defined to set the condition code.

When a storage-alteration PER event is recognized, fewer than 4K additional bytes are stored before the event is indicated by an interruption.

Execution of Interruptible Instructions

The execution of an interruptible instruction is completed when all units of operation associated with that instruction are completed. When an interruption occurs after completion, nullification, or suppression of a unit of operation, all preceding units of operation have been completed, and subsequent units of operation and instructions have not been started. The main difference between these types of ending is the handling of the current unit of operation and whether the instruction address stored in the old PSW identifies the current instruction or the next sequential instruction.

At the time of an interruption, changes to storage locations or register contents which are due to be made by instructions following the interrupted instruction have not yet been made.

Completion: On completion of the last unit of operation of an interruptible instruction, the instruction address in the old PSW designates the next sequential instruction. The result location for the current unit of operation has been updated. It depends on the particular instruction how the operand parameters are adjusted. On completion of a unit of operation other than the last one, the instruction address in the old PSW designates the interrupted instruction or an execute-type instruction, as appropriate. The result location for the current unit of operation has been updated. The operand parameters are adjusted such that the execution of the interrupted instruction is resumed from the point of interruption when the old PSW stored during the interruption is made the current PSW.

Nullification: When a unit of operation is nullified, the instruction address in the old PSW designates the interrupted instruction or an execute-type instruction, as appropriate. The result location for the current unit of operation remains unchanged. The

operand parameters are adjusted such that, if the instruction is reexecuted, execution of the interrupted instruction is resumed with the current unit of operation.

Suppression: When a unit of operation is suppressed, the instruction address in the old PSW designates the next sequential instruction. The operand parameters, however, are adjusted so as to indicate the extent to which instruction execution has been completed. If the instruction is reexecuted after the conditions causing the suppression have been removed, the execution is resumed with the current unit of operation.

Termination: When an exception which causes termination occurs as part of a unit of operation of an interruptible instruction, the entire operation is terminated, and the contents, in general, of any fields due to be changed by the instruction are unpredictable. On such an interruption, the instruction address in the old PSW designates the next sequential instruction.

The differences among the four types of ending for a unit of operation are summarized in Figure 5-5 on page 5-21.

Unit of Operation Is	Instruction Address	Operand Parameters	Current Result Location
Completed			
Last unit of operation	Next instruction	Depends on the instruction	Changed
Any other unit of operation	Current instruction	Next unit of operation	Changed
Nullified	Current Instruction	Current unit of operation	Unchanged
Suppressed	Next Instruction	Current unit of operation	Unchanged
Terminated	Next instruction	Unpredictable	Unpredictable

Figure 5-5. Types of Ending for a Unit of Operation

If an instruction is defined to set the condition code, the execution of the instruction makes the condition code unpredictable except when the last unit of operation has been completed.

Condition-Code Alternative to Interruptibility

The following instructions are not interruptible instructions but instead may be completed after performing a CPU-determined subportion of the processing specified by the parameters of the instructions:

- CHECKSUM
- CIPHER MESSAGE
- CIPHER MESSAGE WITH CFB
- CIPHER MESSAGE WITH CHAINING
- CIPHER MESSAGE WITH COUNTER
- CIPHER MESSAGE WITH OFB
- COMPARE LOGICAL LONG EXTENDED
- COMPARE LOGICAL LONG UNICODE
- COMPARE LOGICAL STRING
- COMPUTE INTERMEDIATE MESSAGE DIGEST
- COMPUTE LAST MESSAGE DIGEST
- COMPUTE MESSAGE AUTHENTICATION CODE
- CONVERT UTF-16 TO UTF-32
- CONVERT UTF-16 TO UTF-8
- CONVERT UTF-32 TO UTF-16
- CONVERT UTF-32 TO UTF-8
- CONVERT UTF-8 TO UTF-16
- CONVERT UTF-8 TO UTF-32
- MOVE LONG EXTENDED
- MOVE LONG UNICODE
- MOVE STRING
- PERFORM CRYPTOGRAPHIC COMPUTATION
- SEARCH STRING
- SEARCH STRING UNICODE
- TRANSLATE AND TEST EXTENDED
- TRANSLATE AND TEST REVERSE EXTENDED
- TRANSLATE EXTENDED
- TRANSLATE ONE TO ONE
- TRANSLATE ONE TO TWO
- TRANSLATE TWO TO ONE
- TRANSLATE TWO TO TWO

When any of the above instructions is completed after performing only a CPU-determined amount of processing instead of all specified processing, the instruction sets condition code 3. On such completion, the instruction address in the PSW designates the next sequential instruction, and the operand parameters of the instruction have been adjusted so that the processing of the instruction can be resumed

simply by branching back to the instruction to execute it again. When the instruction has performed all specified processing, it sets a condition code other than 3.

The points at which any of the above instructions may set condition code 3 are comparable to the points of interruption of an interruptible instruction, and the amount of processing between adjacent points is comparable to a unit of operation of an interruptible instruction. However, since the instruction is not interruptible, each execution is considered the execution of one unit of operation.

Completion with the setting of condition code 3 permits interruptions to occur. Depending on the model and the instruction, condition code 3 may or may not be set when there is not a need for an interruption.

When a storage-alteration PER event is recognized, fewer than 4K additional bytes are stored before the instruction is completed (with condition code 3, if no other condition takes precedence), and the event is indicated by an interruption.

The COMPARE UNTIL SUBSTRING EQUAL and COMPRESSION CALL instructions both are interruptible instructions and ones that may set condition code 3 after performing a CPU-determined amount of processing.

Programming Notes:

1. Any interruption, other than supervisor call and some program interruptions, can occur after a partial execution of an interruptible instruction. In particular, interruptions for external, I/O, machine-check, restart, and program interruptions for access exceptions and PER events can occur between units of operation.
2. The amount of data processed in a unit of operation of an interruptible instruction depends on the model and may depend on the type of condition which causes the execution of the instruction to be interrupted or stopped. Thus, when an interruption occurs at the end of the current unit of operation, the length of the unit of operation may be different for different types of interruptions. Also, when the stop function is requested during the execution of an interruptible instruction, the CPU enters the stopped state at the completion of the execution of the current unit of operation. Similarly, in the instruction-step mode, only a sin-

gle unit of operation is performed, but the unit of operation for the various cases of stopping may be different.

Exceptions to Nullification and Suppression

In certain unusual situations, the result fields of an instruction having a store-type operand are changed in spite of the occurrence of an exception which would normally result in nullification or suppression. These situations are exceptions to the general rule that the operation is treated as a no-operation when an exception requiring nullification or suppression is recognized. Each of these situations may result in the turning on of the change bit associated with the store-type operand, even though the final result in storage may appear unchanged. Depending on the particular situation, additional effects may be observable. The extent of these effects is described along with each of the situations.

All of these situations are limited to the extent that a store access does not occur and the change bit is not set when the store access is prohibited. For the CPU, a store access is prohibited whenever an access exception exists for that access, or whenever an exception exists which is of higher priority than the priority of an access exception for that access.

When, in these situations, an interruption for an exception requiring suppression occurs, the instruction address in the old PSW designates the next sequential instruction. When an interruption for an exception requiring nullification occurs, the instruction address in the old PSW designates the instruction causing the exception even though partial results may have been stored.

Storage Change and Restoration for DAT-Associated Access Exceptions

In this section, the term “DAT-associated access exceptions” is used to refer to those exceptions which may occur as part of the dynamic-address-translation process. These exceptions are ASCE-type, region-first translation, region-second translation, region-third translation, segment translation, page translation, translation specification, and addressing due to a DAT-table entry being designated at a location that is not available in the configuration. The first six of these exceptions normally cause nullification, and the last two normally cause

suppression. Protection exceptions, including those due to DAT protection, are not considered to be DAT-associated access exceptions.

For DAT-associated access exceptions, on some models, channel programs may observe the effects on storage as described in the following case.

When, for an instruction having a store-type operand, a DAT-associated access exception is recognized for any operand of the instruction, that portion, if any, of the store-type operand which would not cause an exception may be changed to an intermediate value but is then restored to the original value.

The accesses associated with storage change and restoration for DAT-associated access exceptions are only observable by channel programs and are not observable by other CPUs in a multiprocessing configuration. Except for instructions which are defined to have multiple-access operands, the intermediate value, if any, is always equal to what would have been the final value if the DAT-associated access exception had not occurred.

Programming Notes:

1. Storage change and restoration for DAT-associated access exceptions occur in two main situations:
 - a. The exception is recognized for a portion of a store-type operand which crosses a page boundary, and the other portion has no access exception.
 - b. The exception is recognized for one operand of an instruction having two storage operands (for example, an SS-format instruction or MOVE LONG), and the other operand, which is a store-type operand, has no access exception.
2. To avoid letting a channel program observe intermediate operand values due to storage change and restoration for DAT-associated access exceptions (especially when a CCW chain is modified), the CPU program should do one of the following:
 - a. Operate on one storage page at a time
 - b. Perform preliminary testing to ensure that no exceptions occur for any of the required pages

- c. Operate with DAT off

Modification of DAT-Table Entries

When a valid and attached DAT-table entry is changed to a value which would cause an exception, and when, before the TLB is cleared of entries which qualify for substitution for that entry, an attempt is made to refer to storage by using a virtual address requiring that entry for translation, the contents of any fields due to be changed by the instruction are unpredictable. Results, if any, associated with the virtual address whose DAT-table entry was changed may be placed in those real locations originally associated with the address. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable. On some machines, this situation may be reported by means of an instruction-processing-damage machine check with the delayed-access-exception bit also indicated.

Trial Execution for Editing Instructions and Translate Instruction

For the instructions EDIT, EDIT AND MARK, and TRANSLATE, the portions of the operands that are actually used in the operation may be established in a trial execution for operand accessibility that is performed before the execution of the instruction is started. This trial execution consists in an execution of the instruction in which results are not stored. If the first operand of TRANSLATE or either operand of EDIT or EDIT AND MARK is changed by another CPU or by a channel program, after the initial trial execution but before completion of execution, the contents of any fields due to be changed by the instruction are unpredictable.

Authorization Mechanisms

Most of the authorization mechanisms that are described in this section permit the control program to establish the degree of function provided to a particular semiprivileged program. These authorization mechanisms are intended for use by programs considered to be semiprivileged, that is, programs that are executed in the problem state but which may be authorized to use additional capabilities. With these authorization controls, a hierarchy of programs may be established, with programs at a higher level having a greater degree of privilege or authority than programs at a lower level. The range of functions

available at each level, and the ability to transfer control from a lower to a higher level, are specified in tables which are managed by the control program. When the linkage stack is used, a nonhierarchical transfer of control also can be specified.

A semiprivileged instruction is one which can be executed in the problem state, but which is subject to the control of one or more of the authorization mechanisms described in this section. There are 23 semiprivileged instructions and also the privileged LOAD ADDRESS SPACE PARAMETERS instruction that are controlled by the authorization mechanisms. All of these semiprivileged and privileged instructions are described in Chapter 10, "Control Instructions".

This section also describes, or refers to descriptions of, authorization mechanisms that are available when the ASN-and-LX-reuse facility is installed. These mechanisms do not apply to particular programs or provide for a hierarchy of programs.

The instructions controlled by the authorization mechanisms are listed in Figure 5-6 on page 5-28. The figure also shows additional authorization mechanisms that do not control specifically semiprivileged instructions; they control implicit access-register translation (access-register translation as part of an instruction making a storage reference) and also access-register translation in the LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, STORE REAL ADDRESS, TEST ACCESS, and TEST PROTECTION instructions and a special form of access-register translation in the BRANCH IN SUBSPACE GROUP instruction. These additional mechanisms (the extended authorization index, ALE sequence number, and ASTE sequence number) index and ALE sequence number) are described in "Access-Register-Specified Address Spaces" on page 5-43, and the ASTE instance number is further described in "ASN-Second-Table-Entry Sequence Number" on page 3-20. The figure also shows the LSTE sequence number, which may be used in PC-number translation in the PROGRAM CALL instruction and is described in "ASN-and-LX-Reuse Control (R)." on page 5-31.

Mode Requirements

Most of the semiprivileged instructions can be executed only with DAT on. Basic PROGRAM CALL, PROGRAM TRANSFER, and PROGRAM TRANSFER WITH INSTANCE, are valid only in the primary-space mode. (Basic PROGRAM CALL is the PRO-

GRAM CALL operation when the linkage stack is not used. When the linkage stack is used, the PROGRAM CALL operation is called stacking PROGRAM CALL). MOVE TO PRIMARY and MOVE TO SECONDARY are valid only in the primary-space and secondary-space modes. BRANCH AND STACK, stacking PROGRAM CALL, and PROGRAM RETURN are valid only in the primary-space and access-register modes. EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE are valid only in the primary-space, access-register, and home-space modes. When a semiprivileged instruction is executed in an invalid translation mode, a special-operation exception is recognized.

PROGRAM TRANSFER and PROGRAM TRANSFER WITH INSTANCE specify a new value for the problem-state bit in the PSW. If a program in the problem state attempts to execute PROGRAM TRANSFER or PROGRAM TRANSFER WITH INSTANCE and set the supervisor state, a privileged-operation exception is recognized. A privileged-operation exception is also recognized on an attempt to use RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST to set the home-space mode in the problem state.

Extraction-Authority Control

The extraction-authority-control bit is located in bit position 36 of control register 0. In the problem state, bit 36 must be one to allow completion of these instructions:

- EXTRACT PRIMARY ASN
- EXTRACT PRIMARY ASN AND INSTANCE
- EXTRACT SECONDARY ASN
- EXTRACT SECONDARY ASN AND INSTANCE
- INSERT ADDRESS SPACE CONTROL
- INSERT PSW KEY
- INSERT VIRTUAL STORAGE KEY

Otherwise, a privileged-operation exception is recognized. The extraction-authority control is not examined in the supervisor state.

PSW-Key Mask

The PSW-key mask consists of bits 32-47 in control register 3, with the bits corresponding to the values 0-15, respectively, of the PSW key. These bits are used in the problem state to control which keys and entry points are authorized for the program. The

PSW-key mask is modified by PROGRAM TRANSFER, and PROGRAM TRANSFER WITH INSTANCE, is modified or loaded by BRANCH AND SET AUTHORITY and PROGRAM CALL, and is loaded by LOAD ADDRESS SPACE PARAMETERS and PROGRAM RETURN. The PSW-key mask is used in the problem state to control the following:

- The PSW-key values that can be set by means of the instruction SET PSW KEY FROM ADDRESS.
- The PSW-key values that are valid for the six move instructions that specify a second access key: MOVE PAGE, MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH DESTINATION KEY, MOVE WITH KEY, and MOVE WITH SOURCE KEY.
- The PSW-key values that are valid for the MOVE WITH OPTIONAL SPECIFICATIONS instruction when a key is explicitly specified for the first, second, or both operands.
- The entry points which can be called by means of PROGRAM CALL. In this case, the PSW-key mask is ANDed with the authorization key mask in the entry-table entry, and, if the result is zero, the program is not authorized.

When an instruction in the problem state attempts to use a key not authorized by the PSW-key mask, a privileged-operation exception is recognized. The same action is taken when an instruction in the problem state attempts to call an entry not authorized by the PSW-key mask. The PSW-key mask is not examined in the supervisor state, all keys and entry points being valid.

Secondary-Space Control

Bit 37 of control register 0 is the secondary-space-control bit. This bit provides a mechanism whereby the control program can indicate whether or not the secondary region-first table, region-second table, region-third table or segment table has been established. Bit 37 may be required to be one to allow completion of SET ADDRESS SPACE CONTROL FAST and MOVE WITH OPTIONAL SPECIFICATIONS, and it must be one to allow completion of these instructions:

- MOVE TO PRIMARY
- MOVE TO SECONDARY
- SET ADDRESS SPACE CONTROL

Otherwise, a special-operation exception is recognized. The secondary-space control is examined in both the problem and supervisor states.

Subsystem-Linkage Control

Bit 192 of the primary ASN-second-table entry is the subsystem-linkage-control bit. The subsystem-linkage control must be one to allow completion of these instructions:

- PROGRAM CALL
- PROGRAM TRANSFER
- PROGRAM TRANSFER WITH INSTANCE

Otherwise, a special-operation exception is recognized. The subsystem-linkage control is examined in both the problem and supervisor states and controls both the space-switching and current-primary versions of the instructions.

ASN-Translation Control

Bit 44 of control register 14 is the ASN-translation-control bit. This bit provides a mechanism whereby the control program can indicate whether ASN translation may occur while a particular program is being executed. Bit 44 must be one to allow completion of these instructions:

- LOAD ADDRESS SPACE PARAMETERS
- SET SECONDARY ASN
- SET SECONDARY ASN WITH INSTANCE
- PROGRAM CALL with space switching
- PROGRAM RETURN with space switching and also when the restored secondary ASN is not equal to the restored primary ASN
- PROGRAM TRANSFER with space switching
- PROGRAM TRANSFER WITH INSTANCE with space switching

Otherwise, a special-operation exception is recognized. The ASN-translation control is examined in both the problem and supervisor states. The ASN-translation control is examined by PROGRAM CALL even though PROGRAM CALL obtains the address of the ASN-second-table entry directly from the entry-table entry instead of by performing ASN translation.

Authorization Index

The authorization index is contained in bit positions 32-47 of control register 4. The authorization index is associated with the primary address space and is

loaded along with the PASN when PROGRAM CALL with space switching, PROGRAM RETURN with space switching, PROGRAM TRANSFER with space switching, PROGRAM TRANSFER WITH INSTANCE with space switching, or LOAD ADDRESS SPACE PARAMETERS is executed. The authorization index is used to determine whether a program is authorized to establish a particular address space. A program may be authorized to establish the address space as a secondary-address space, as a primary-address space, or both. The authorization index is examined in both the problem and supervisor states.

Associated with each address space is an authority table. The authorization index is used to select an entry in the authority table. Each entry contains two bits, which indicate whether the program with that authorization index is permitted to establish the address space as a primary address space, as a secondary address space, or both.

The instructions SET SECONDARY ASN with space switching, and SET SECONDARY ASN WITH INSTANCE with space switching, and the instruction PROGRAM RETURN when the restored secondary ASN is not equal to the restored primary ASN, use the authorization index to test the secondary-authority bit in the authority-table entry to determine if the address space can be established as a secondary address space. The tested bit must be one; otherwise, a secondary-authority exception is recognized.

The instructions PROGRAM TRANSFER with space switching and PROGRAM TRANSFER WITH INSTANCE with space switching use the authorization index to test the primary-authority bit in the authority-table entry to determine if the address space can be established as a primary address space. The tested bit must be one; otherwise, a primary-authority exception is recognized.

The instruction PROGRAM CALL with space switching causes a new authorization index to be loaded from the ASN-second-table entry. This permits the program which is called to be given an authorization index which authorizes it to access more or different address spaces than those authorized for the calling program. The instructions PROGRAM RETURN with space switching, PROGRAM TRANSFER with space switching, and PROGRAM TRANSFER WITH INSTANCE with space switching restore the authorization index that is associated with the returned-to address space.

The secondary-authority bit in the authority-table entry may also be used, along with the extended authorization index, to determine if the program is authorized to use an access-list entry in access-register translation. This is described in “Access-Register-Specified Address Spaces” on page 5-43.

Instructions and Controls Related to ASN-and-LX Reuse

This section describes instructions and controls that are provided when the ASN-and-LX-reuse facility is installed. The controls apply in both the problem and the supervisor states.

Instructions: The ASN-and-LX-reuse facility provides the following instructions:

- EXTRACT PRIMARY ASN AND INSTANCE
- EXTRACT SECONDARY ASN AND INSTANCE
- PROGRAM TRANSFER WITH INSTANCE
- SET SECONDARY ASN WITH INSTANCE

PROGRAM TRANSFER WITH INSTANCE and SET SECONDARY ASN WITH INSTANCE are similar to PROGRAM TRANSFER and SET SECONDARY ASN, respectively. The space-switching forms of PROGRAM TRANSFER WITH INSTANCE and SET SECONDARY ASN WITH INSTANCE compare an ASTE instance number (ASTEIN) specified by the program in general register 1 to the ASTEIN in the ASN-second-table entry for the address space to which the space switch is to occur. An ASTE-instance exception is recognized if the two ASTEINs are not equal.

ASN-and-LX-Reuse Control: Bit 44 of control register 0 is the ASN-and-LX-reuse control. This bit is used to control both ASN reuse and LX reuse, as described below.

ASN Reuse: When the ASN-and-LX-reuse control is one, PROGRAM RETURN and LOAD ADDRESS SPACE PARAMETERS, when they perform ASN translation of either a primary ASN or a secondary ASN, compare a specified ASTEIN to the ASTEIN in the ASN-second-table entry located by the ASN translation. The two ASTEINs must be equal; otherwise, PROGRAM RETURN recognizes an ASTE-instance exception, and LOAD ADDRESS SPACE PARAMETERS sets condition code 1 or 2. For PROGRAM RETURN, the specified ASTEIN is in the linkage-stack program-call state entry being unstacked.

For LOAD ADDRESS SPACE PARAMETERS, the specified ASTEIN is in the first operand of the instruction.

Bit 31 of word 1 of the ASN-second-table entry is the reusable-ASN bit. When the ASN-and-LX-reuse control is one, the space-switching forms of PROGRAM TRANSFER and SET SECONDARY ASN recognize a special-operation exception if the reusable-ASN bit is one in the ASN-second-table entry located by ASN translation. These instructions do not make a comparison to the ASTEIN in the ASN-second-table entry.

Bit 30 of word 1 of the ASN-second-table entry is the controlled-ASN bit. Regardless of the value of the ASN-and-LX-reuse control, PROGRAM TRANSFER WITH INSTANCE and SET SECONDARY ASN WITH INSTANCE recognize a special-operation exception if the controlled-ASN bit is one in the ASN-second-table entry located by ASN translation and the CPU is in the problem state at the beginning of the operation.

A more detailed description of ASN reuse is given in “ASN-Second-Table-Entry Instance Number and ASN Reuse” on page 3-21.

LX Reuse: When the ASN-and-LX-reuse control is one, the linkage table is replaced by a linkage first table and, for each valid linkage-first-table entry, a linkage second table.

The second word of the linkage-second-table entry used contains an LSTE sequence number (LSTESN), and PROGRAM CALL recognizes an LSTE-sequence exception if this LSTESN is nonzero and not equal to an LSTESN specified by the program in general register 15.

A more detailed description of LX reuse is given in “ASN-and-LX-Reuse Control (R):” on page 5-31.

Explanation for Summary of Authorization Mechanisms:

- 1 The PSW-key mask is ANDed with the authorization key mask in the entry-table entry.
- 2 The exception is recognized on an attempt to set the supervisor state when in the problem state.

Function or Instruction	Mode Requirement		Authorization Mechanism											Space Switch Event Ctl. (1.57, 13.57)
	Pr. Op	Trans. Mode	Subs. Link. Ctl. ⁶	Sec.- Space Ctl. (0.37)	ASN- Trans. Ctl. (14.44)	Extr. Auth. Ctl (0.36)	PSW- Key Mask (3.32- 3.47)	Auth. Index (4.32- 4.47)	Ext.- Auth. Index (8.32- 8.47)	ALE Seq. No. ⁷	ASTE Seq. No. ⁸	ASTE Inst. No. ⁹	LSTE Seq. No. ¹⁰	
Implicit AR translation BAKR BSA-ba BSA-ra BSG EPAR		A SO-PA SO-PSAH SO-PSAH					Q		EA	ALQ	ASQ ASQ			
EPAIR EREG EREGG ESAR ESAIR ESTA		SO-PSAH SO-PAH SO-PAH SO-PSAH SO-PSAH SO-PAH				Q Q Q								
IAC IPK IVSK LASP LPTEA LRA	P P P	SO-PSAH SO-PSAH			SO	Q Q Q		CC				CC CCA CCA	CC CCA CCA	CC
LRAG MSTA MVCDK MVCOS MVCP MVCS	P	SO-PAH SO-PSAH SO-PS SO-PS		SO-MS SO SO			Q Q Q Q		CCA	CCA	CCA			
MVCSK bPC-cp sPC-cp bPC-ss sPC-ss PR-cp		SO-P SO-PA SO-P SO-PA SO-PA	SO SO SO SO		SO SO SO ⁴		Q Q ¹ Q ¹ Q ¹ Q ¹				ASQ ASQ ASQ		LSQ LSQ LSQ	X1 X1
PR-ss PT-cp PT-ss ¹¹ PTI-cp PTI-ss ¹² RP	Q ² Q ² Q ² Q ²	SO-PA SO-P SO-P SO-P SO-P	SO SO SO SO		SO SO SO			PASA ⁵ PA PA			ASQ ASQ ASQ	AIN AIN		X1 X1 X1 X2
SAC SACF SPKA SSAR-cp SSAR-ss ¹¹ SSAIR-cp	Q ³ Q ³	SO-PSAH SO-PSAH SO-PSAH SO-PSAH SO-PSAH		SO SO ¹³			Q				ASQ			X2 X2

Figure 5-6. Summary of Authorization Mechanisms (Part 1 of 2)

Function or Instruction	Mode Requirement		Authorization Mechanism											Space Switch Event Ctl. (1.57, 13.57)
	Pr. Op	Trans. Mode	Subs. Link. Ctl. ⁶	Sec.- Space Ctl. (0.37)	ASN- Trans. Ctl. (14.44)	Extr. Auth. Ctl (0.36)	PSW- Key Mask (3.32- 3.47)	Auth. Index (4.32- 4.47)	Ext.- Auth. Index (8.32- 8.47)	ALE Seq. No. ⁷	ASTE Seq. No. ⁸	ASTE Inst. No. ⁹	LSTE Seq. No. ¹⁰	
SSAIR-ss ¹²		SO-PSAH			SO			SA			ASQ	AIN		
STRAG	P								CC	CC	CC			
TAR									CC	CC	CC			
TPROT	P								CC	CC	CC			

Figure 5-6. Summary of Authorization Mechanisms (Part 2 of 2)

3	The exception is recognized on an attempt to set the home-space mode when in the problem state.	A	Access-register translation occurs only in the access-register mode.
4	ASN translation is performed for the new SASN, and the exception may be recognized, only when the new SASN is not equal to the new PASN.	AIN	ASTE-instance exception.
		ALQ	ALE-sequence exception.
		ASQ	ASTE-sequence exception
5	Secondary authority is checked for the new SASN, and the exception may be recognized, only when the new SASN is not equal to the new PASN.	bPC	Basic (nonstacking) PROGRAM CALL.
		CC	Test results in setting a condition code.
		CCA	Test results in setting a condition code. The test occurs only in the access-register mode.
6	Subsystem-linkage control is bit 92 of the primary ASN-second-table entry.	CRx.y	Control register x, bit position y.
7	ALE sequence number is bits 8-15 of the access-list entry	EA	Extended-authority exception.
8	ASTE sequence number is bits 160-191 of the ASN-second-table entry.	LSQ	LSTE-sequence exception.
9	ASTE instance number is bits 352-383 of the ASN-second-table entry	P	Privileged-operation exception for privileged instruction.
10	LSTE sequence number is bits 32-63 of the linkage-second-table entry.	PA	Primary-authority exception.
		PASA	Primary-authority exception or secondary-authority exception.
11	Special-operation exception is recognized if ASN-and-LX-reuse control, bit 44 of control register 0, is one and reusable-ASN bit, bit 31 of word 1 of the ASN-second-table entry, is one.	Q	Privileged-operation exception for semi-privileged instruction. Authority checked only in the problem state.
		SA	Secondary-authority exception.
12	Special-operation exception is recognized if controlled-ASN bit, bit 30 of word 1 of the ASN-second-table entry, is one and the CPU is in the problem state at the beginning of the operation.	SO	Special-operation exception.
		SO-MS	The secondary-space control, bit 37 of control register 0, must be one when either operand of MOVE WITH OPTIONAL SPECIFICATIONS is accessed in the secondary-space mode; otherwise, a special-operation exception is recognized.
13	Whether the exception is recognized is unpredictable.		

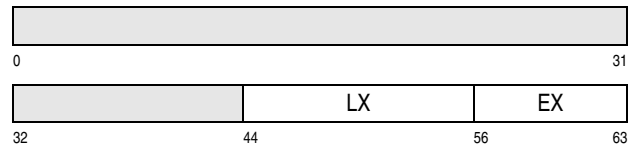
SO-P	CPU must be in the primary-space mode; special-operation exception if the CPU is in the secondary-space, access-register, home-space, or real mode.
SO-PA	CPU must be in the primary-space or access-register mode; special-operation exception if the CPU is in the secondary-space, home-space, or real mode.
SO-PAH	CPU must be in the primary-space, access-register, or home-space mode; special-operation exception if the CPU is in the secondary-space or real mode.
SO-PS	CPU must be in the primary-space or secondary-space mode; special-operation exception if the CPU is in the home-space, access-register, or real mode.
SO-PSAH	CPU must be in the primary-space, secondary-space, access-register, or home-space mode; special-operation exception if the CPU is in the real mode.
sPC	Stacking PROGRAM CALL.
X1	When bit 57 of control register 1 is one, a space-switch event is recognized. The operation is completed.
X2	When bit 57 of control register 1 or 13 is one and the instruction space is changed to or from the home address space, a space-switch event is recognized. The operation is completed.

PC-Number Translation

PC-number translation is the process of translating PC number to locate an entry-table entry as part of the execution of the PROGRAM CALL instruction. When the ASN-and-LX-reuse facility is not installed or is not enabled, the PC number is 20 bits in bit positions 44-63 of the effective address used by PROGRAM CALL. When the ASN-and-LX-reuse facility is installed and enabled, the PC number is 20 bits in bit positions 44-63 of the effective address if bit 44 of the address is zero, or it is 32 bits in bit positions 32-63 of the address if bit 44 of the address is one. The facility is enabled if the ASN-and-LX-reuse control, bit 44 of control register 0, is one. The case when the facility is installed and enabled is referred to simply by saying that ASN-and-LX reuse is enabled.

To perform the translation of a PC number to locate an entry-table entry, the PC number is divided into two fields: a linkage index (LX) and an entry index (EX). The entry index is always the rightmost eight bits of the PC number. When ASN-and-LX reuse is not enabled, the leftmost 12 bits of the 20-bit PC number are the linkage index. In this case, the effective address has the following format:

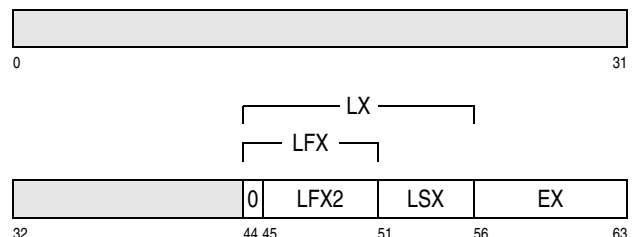
Effective Address when ASN-and-LX Reuse Is Not Enabled



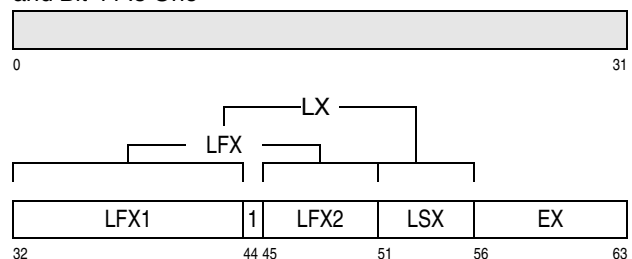
Bit 44 of the effective address has no special meaning and may be zero or one.

When ASN-and-LX reuse is enabled, the linkage index is further divided into a linkage first index (LFX) and a linkage second index (LSX). The linkage second index is always the five bits immediately on the left of the entry index. The size and format of the linkage first index depend on whether the PC number is 20 bits or 32 bits, which in turn depends on whether bit 44 of the effective address is zero or one, respectively. In these cases, the effective address has the following formats:

Effective Address when ASN-and-LX Reuse Is Enabled and Bit 44 Is Zero



Effective Address when ASN-and-LX Reuse Is Enabled and Bit 44 Is One



When ASN-and-LX reuse is enabled and bit 44 of the effective address is zero, the linkage first index is bits 44-50, or bits 45-50 (LFX2) with a zero appended on the left. Thus, the linkage first index is seven bits of which the leftmost bit is always zero. When bit 44 is one, the linkage first index is bits 45-50 (LFX2) with bits 32-43 (LFX1) appended on the left, or 18 bits. However, a linkage first table can contain at most 16,384 entries, and, therefore, the leftmost four bits of the linkage first index, bits 32-35 of the effective address, must always be zeros; otherwise, an LFX-translation exception is recognized.

When ASN-and-LX reuse is not enabled, the translation of a PC number is performed by means of two tables: a linkage table and an entry table. When ASN-and-LX reuse is enabled, the translation is performed by means of three tables: a linkage first table, a linkage second table, and an entry table. All of these tables reside in real storage. The linkage-table designation or linkage-first-table designation resides in another area in storage, called the primary ASN-second-table entry (primary ASTE), whose origin is in control register 5. When there are two levels of tables, The entry table is designated by means of a linkage-table entry. When there are three levels of tables, the linkage second table is designated by means of a linkage-first-table entry, and the entry table is designated by means of a linkage-second-table entry.

Programming Notes:

1. Bit 44 of a PROGRAM CALL effective address specifying a 32-bit PC number is not a numeric part of the PC number.
2. The effective address from which a PC number is derived is subject to the addressing mode in the current PSW. Therefore, bits 0-39 of the effective address in the 24-bit addressing mode, and bits 0-32 in the 31-bit addressing mode, are treated as containing zeros.

PC-Number Translation Control

PC-number translation is controlled by means of the ASN-and-LX-reuse control in control register 0 and a linkage-table designation or linkage-first-table designation

in the primary ASN-second-table entry designated by the contents of control register 5.

Control Register 0



ASN-and-LX-Reuse Control (R): Bit 44 of control register 0 is the ASN-and-LX-reuse-control bit and is assigned if the ASN-and-LX-reuse facility is installed. When the bit is one, (1) the PC number is 20 bits if bit 44 of the effective address is zero or 32 bits if bit 44 is one, (2) the PC number is divided into a seven-bit (bit 44 is zero) or 18 bit (bit 44 is one) linkage first index, five-bit linkage second index, and eight-bit entry index, (3) the primary ASN-second-table entry contains a linkage-first-table designation instead of a linkage-table designation, and (4) the PC number is translated by means of a linkage first table, linkage second table, and entry table.

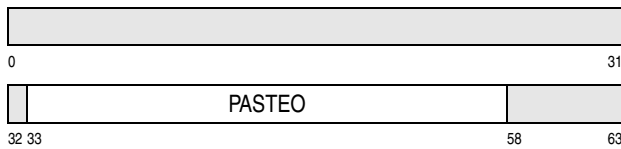
Also when the ASN-and-LX-reuse-control bit is one, if the linkage-second-table entry used during PC-number translation performed by PROGRAM CALL contains a nonzero linkage-second-table-entry sequence number (LSTESN), this LSTESN must be equal to an LSTESN specified in bit positions 0-31 of general register 15; otherwise, the PROGRAM CALL instruction cannot be completed, and an LSTE-sequence exception is recognized.

The ASN-and-LX-reuse control in control register 0 is examined in both the problem and the supervisor states. Other uses of this control, related to the ASN-second-table-entry instance number, are summarized in "ASN-Second-Table-Entry Instance Number and ASN Reuse" on page 3-21.

This use of the LSTESN allows a linkage index associated with a particular LSTESN to be made unusable when the linkage index is reassigned to specify a different (different origin) or conceptually different (different contents, or containing ASNs that designate conceptually different address spaces) entry table. The LX-and-LSTESN combination can be made unusable by changing the LSTESN in the linkage-second-table entry.

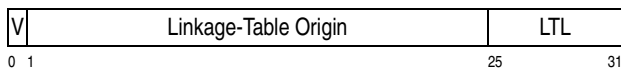
Control Register 5

Control register 5 specifies the location of the primary ASN-second-table entry. The register has the following format:



Primary-ASTE Origin (PASTE0): Bits 33-57 of control register 5, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the primary ASTE. Bytes 24-27 of the primary ASTE contain the linkage-table designation if the ASN-and-LX-reuse facility is not installed or is not enabled by a one value of the ASN-and-LX-reuse control in control register 0, or they contain the linkage-first-table designation if the facility is installed and enabled.

The linkage-table designation has the following format:



Subsystem-Linkage Control (V): Bit 0 of the linkage-table designation is the subsystem-linkage-control bit. Bit 0 must be one to allow completion of these instructions:

- PROGRAM CALL
- PROGRAM TRANSFER
- PROGRAM TRANSFER WITH INSTANCE

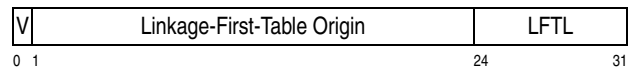
Otherwise, a special-operation exception is recognized. The subsystem-linkage control is examined in both the problem and the supervisor states and controls both the space-switching and current-primary versions of the instructions.

Linkage-Table Origin: Bits 1-24 of the linkage-table designation, with seven zeros appended on the right, form a 31-bit real address that designates the beginning of the linkage table.

Linkage-Table Length (LTL): Bits 25-31 of the linkage-table designation specify the length of the linkage table in units of 128 bytes, thus making the length of the linkage table variable in multiples of 32 four-byte entries. The length of the linkage table, in

units of 128 bytes, is one more than the value in bit positions 25-31. The linkage-table length is compared against the leftmost seven bits of the linkage-index portion of the PC number to determine whether the linkage index designates an entry within the linkage table.

The linkage-first-table designation has the following format:



Subsystem-Linkage Control (V): Bit 0 of the linkage-first-table designation has the same definition as bit 0 of the linkage-table designation.

Linkage-First-Table Origin: Bits 1-23 of the linkage-first-table designation, with eight zeros appended on the right, form a 31-bit real address that designates the beginning of the linkage first table.

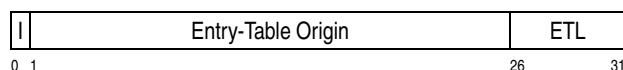
Linkage-First-Table Length (LFTL): Bits 24-31 of the linkage-first-table designation specify the length of the linkage first table in units of 256 bytes, thus making the length of the linkage first table variable in multiples of 64 four-byte entries. The length of the linkage first table, in units of 256 bytes, is one more than the value in bit positions 24-31. When bit 44 of the effective address specifying the PC number is one, the linkage-first-table length is compared against the leftmost 12 bits of the linkage-first-index portion of the PC number, bits 32-43 of the effective address, to determine whether the linkage first index designates an entry within the linkage first table. When bit 44 of the effective address is zero, the linkage-first-table length is ignored.

PC-Number Translation Tables

If the ASN-and-LX-reuse facility is not installed or is not enabled by a one value of the ASN-and-LX-reuse control in control register 0, the PC-number translation process consists in a two-level lookup using two tables: a linkage table and an entry table. If the facility is installed and enabled, the PC-number translation process consists in a three-level lookup using three tables: a linkage first table, a linkage second table, and an entry table. All of these tables reside in real storage.

Linkage-Table Entries

The entry fetched from the linkage table has the following format:



The fields in the linkage-table entry are allocated as follows:

LX-Invalid Bit (I): Bit 0 controls whether the entry table associated with the linkage-table entry is available.

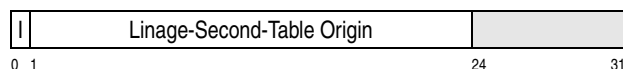
When the bit is zero, PC-number translation proceeds by using the linkage-table entry. When the bit is one, an LX-translation exception is recognized.

Entry-Table Origin: Bits 1-25, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the entry table.

Entry-Table Length (ETL): Bits 26-31 specify the length of the entry table in units of 128 bytes, thus making the table variable in multiples of four 32-byte entries. The length of the entry table, in units of 128 bytes, is one more than the value in bit positions 26-31. The entry-table length is compared against the leftmost six bits of the entry index to determine whether the entry index designates an entry within the entry table.

Linkage-First-Table Entries

The entry fetched from the linkage first table has the following format:



The fields in the linkage-first-table entry are allocated as follows:

LFX-Invalid Bit (I): Bit 0 controls whether the linkage second table associated with the linkage-first-table entry is available.

When the bit is zero, PC-number translation proceeds by using the linkage-first-table entry. When the bit is one, an LFX-translation exception is recognized.

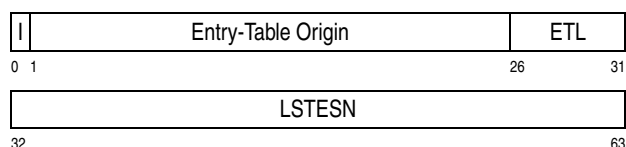
Linkage-Second-Table Origin: Bits 1-23, with eight zeros appended on the right, form a 31-bit real

address that designates the beginning of the linkage second table.

Programming Note: The unused field in the linkage-first-table entry, bits 24-31, should be set to zeros. This field is reserved for future extensions, and programs which place nonzero values in this field may not operate compatibly on future machines.

Linkage-Second-Table Entries

The entry fetched from the linkage second table has the following format:



The fields in the linkage-second-table entry are allocated as follows:

LSX-Invalid Bit (I): Bit 0 controls whether the entry table associated with the linkage-second-table entry is available.

When the bit is zero, PC-number translation proceeds by using the linkage-second-table entry. When the bit is one, an LSX-translation exception is recognized.

Entry-Table Origin: Bits 1-25, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the entry table.

Entry-Table Length (ETL): Bits 26-31 specify the length of the entry table in units of 128 bytes, thus making the table variable in multiples of four 32-byte entries. The length of the entry table, in units of 128 bytes, is one more than the value in bit positions 26-31. The entry-table length is compared against the leftmost six bits of the entry index to determine whether the entry index designates an entry within the entry table.

Linkage-Second-Table-Entry Sequence Number (LSTESN): During PC-number translation in PROGRAM CALL when bits 32-63 are not all zeros, bits 32-63 are compared to an LSTESN specified in bit positions 0-31 of general register 15. The comparison must give an equal result; otherwise, an LSTE-sequence exception is recognized. When bits 32-63 are all zeros, bits 0-31 of general register 15 are ignored.

Programming Note: The format of bits 0-31 of the linkage-second-table entry is the same as the format of the linkage-table entry.

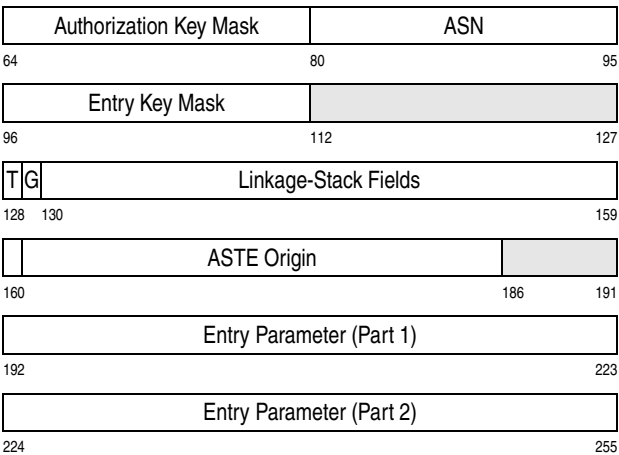
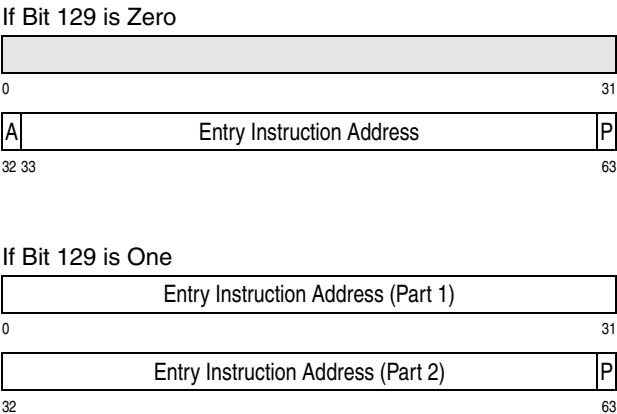
Entry-Table Entries

The format of bits 0-63 of the entry-table entry depends on whether the addressing-mode in effect after the PROGRAM CALL operation is the extended (64-bit) addressing mode or a basic (24-bit or 31-bit) addressing mode. This in turn depends on bits 128 and 129 of the entry-table entry.

Bit 128 of the entry-table entry (T) is the PC-type bit. When bit 128 is zero, PROGRAM CALL is to perform the basic (nonstacking) operation. When bit 128 is one, PROGRAM CALL is to perform the stacking operation.

Bit 129 of the entry-table entry (G) is the entry-extended-addressing-mode bit. In the basic PROGRAM CALL operation, bit 31 of the current PSW, the extended-addressing-mode bit, must equal bit 129; otherwise, a special-operation exception is recognized. In the stacking operation when bit 129 is zero, bit 31 of the current PSW is set to zero, and bit 32 of the PSW, the basic-addressing-mode bit, is set with the value of bit 32 of the entry-table entry (A), the entry-basic-addressing-mode bit. In the stacking operation when bit 129 is one, bits 31 and 32 of the current PSW both are set to one. Thus, the basic PROGRAM CALL operation does not switch between the extended and a basic addressing mode but can switch between the 24-bit and 31-bit modes, and the stacking operation can set any addressing mode.

The 32-byte entry-table entry has the following format:



The fields in the entry-table entry are allocated as follows:

Entry Basic Addressing Mode (A): When bit 129 is zero, bit 32 replaces the basic-addressing-mode bit, bit 32 of the current PSW, as part of the PROGRAM CALL operation. In this case if bit 32 is zero, bits 33-39 must also be zeros; otherwise, a PC-translation-specification exception is recognized. When bit 129 is one, bit 32 is a bit of the entry instruction address, and bit 32 of the PSW remains or is set to one.

Entry Instruction Address: When bit 129 is zero, bits 33-62, with 33 zeros appended on the left and a zero appended on the right, form the instruction address which replaces the instruction address in the PSW as part of the PROGRAM CALL operation. When bit 129 is one, bits 0-62, with a zero appended on the right, form the instruction address.

Entry Problem State (P): Bit 63 replaces the problem-state bit, bit 15 of the current PSW, as part of the PROGRAM CALL operation.

Authorization Key Mask: Bits 64-79 are used to verify whether the program issuing the PROGRAM CALL instruction, when in the problem state, is authorized to call this entry point. The authorization key mask and the current PSW-key mask in control register 3 are ANDed, and the result is checked for all zeros. If the result is all zeros, a privileged-operation exception is recognized. The test is not performed in the supervisor state.

ASN: Bits 80-95 specify whether a space-switching (PC-ss) operation or a to-current-primary (PC-cp) operation is to occur. When bits 80-95 are zeros, PC-

cp is specified. When bits 80-95 are not all zeros, PC-ss is specified, and the bits are the ASN that replaces the primary ASN.

Entry Key Mask: Bits 96-111 may be ORed into or may replace the PSW-key mask in control register 3 as part of the PROGRAM CALL operation, as determined by a bit in bit positions 130-159.

PC-Type Bit (T): Bit 128 specifies the basic PROGRAM CALL operation when the bit is zero or the stacking PROGRAM CALL operation when the bit is one.

Entry Extended Addressing Mode (G): In the basic PROGRAM CALL operation, bit 129 must match the extended-addressing-mode bit, bit 31 of the current PSW; otherwise, a special-operation exception is recognized. In the stacking operation, bit 129 replaces bit 31 of the PSW.

ASTE Origin: When bits 80-95 are not all zeros, bits 161-185, with six zeros appended on the right, form the 31-bit real ASN-second-table-entry address that should result from applying the ASN-translation process to bits 80-95.

Entry Parameter: When bit 129 is zero, bits 224-255 are placed in bit positions 32-63 of general register 4, and bits 0-31 of the register remain unchanged, as part of the PC operation. When bit 129 is one, bits 192-255 are placed in general register 4 as part of the PC operation.

Bits 130-159 are used in connection with the linkage stack and are described in “Linkage-Stack Entry-Table Entries” on page 5-71.

Bits 112-127, 160, and 186-191 are reserved for possible future extensions and should be zeros.

Programming Note: The entry parameter is intended to provide the called program with an address which can be depended upon and used as

the basis of addressability in locating necessary information which may be environment dependent. The parameter may be appropriately changed for each environment by setting up different entry tables. The alternative — obtaining this information from the calling program — may require extensive validity checking or may present an integrity exposure.

Table Summary

Figure 5-7 on page 5-35 presents a summary of information about the tables used in PC-number translation, except for the primary ASN second table.

PC-Number-Translation Process

When the ASN-and-LX-reuse facility is not installed or is not enabled, the translation of the PC number is performed by means of a linkage table and entry table. When the ASN-and-LX-reuse facility is installed and enabled, referred to by saying that ASN-and-LX reuse is enabled, the translation of the PC number is performed by means of a linkage first table, linkage second table and entry table. All of these tables reside in real storage. The translation also requires the use of the primary ASN-second-table entry, which also resides in real storage.

For the purposes of PC-number translation, the PC number is divided into two parts: the leftmost part is called the linkage index (LX), and the rightmost eight bits are called the entry index (EX). When ASN-and-LX reuse is not enabled, the PC number is 20 bits, and the linkage index is 12 bits. When ASN-and-LX reuse is enabled, the linkage index also is divided into two parts: the leftmost part is called the linkage first index (LFX), and the rightmost five bits are called the linkage second index (LSX). When ASN-and LX reuse is enabled and bit 44 of the effective address specifying the PC number is zero, the PC number is 20 bits, and the linkage first index is seven bits, or, when bit 44 is one, the PC number is 32 bits, and the linkage first index is 18 bits.

Table	Boundary (Bytes)	Entry Size (Bytes)	Unit Size		Maximum Number of Units	Maximum Size	
			Bytes	Entries		Bytes	Entries
Linkage	12	4	128	32	128	16384	4096
Linkage First	256	4	256	64	256 ¹	65536	16384

Figure 5-7. Summary of PC-Number-Translation Tables

Table	Boundary (Bytes)	Entry Size (Bytes)	Unit Size		Maximum Number of Units	Maximum Size	
			Bytes	Entries		Bytes	Entries
Linkage Second	256	8	256	32	1	256	32
Entry	64	32	128	4	64	8192	256
Explanation: ¹ A 20-bit PC number can specify an entry only within the first unit. Bits 0-3 of a 32-bit PC number specify entries beyond the end of the largest possible table.							

Figure 5-7. Summary of PC-Number-Translation Tables

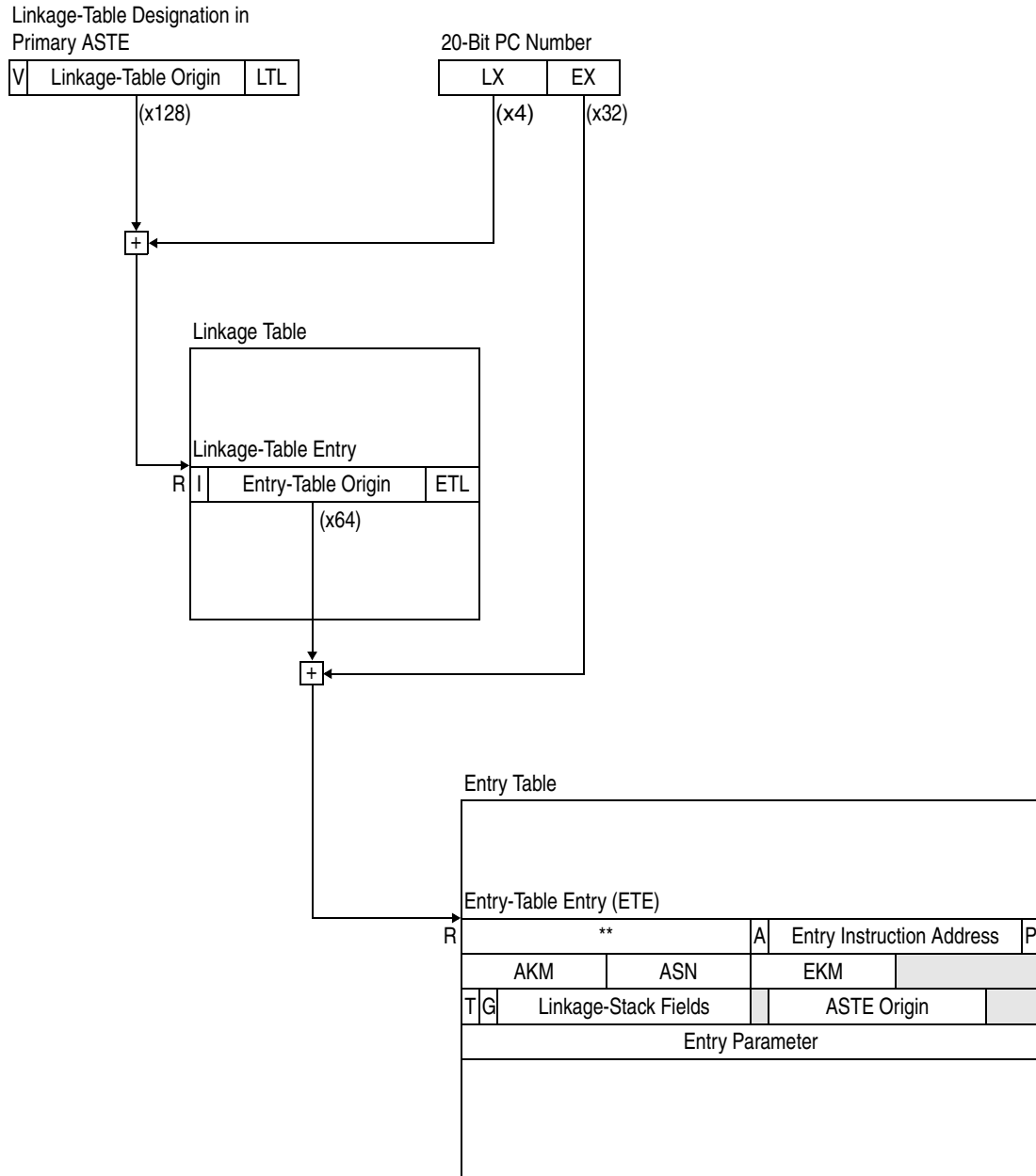
When ASN-and-LX reuse is not enabled, the LX is used to select an entry from the linkage table, the starting address and length of which are specified by the linkage-table designation in the primary ASTE. This entry designates the entry table to be used. The EX field of the PC number is used to select an entry from the entry table.

When ASN-and-LX reuse is enabled, the LFX is used to select an entry from the linkage first table, the starting address and length of which are specified by the linkage-first-table designation in the primary ASTE. This entry designates the linkage second table to be used. The LSX is then used to select an entry from the linkage second table, which entry designates the entry table to be used. The EX field of the PC number is then used to select an entry from the entry table.

When ASN-and-LX reuse is enabled, a linkage-second-table-entry sequence number (LSTESN) in general register 15 is compared to the LSTESN in the linkage-second-table entry if the LSTESN in the entry is nonzero.

When, for the purposes of PC-number translation, accesses are made to main storage to fetch entries from the primary ASTE, linkage table, linkage first table, linkage second table, and entry table, key-controlled protection does not apply.

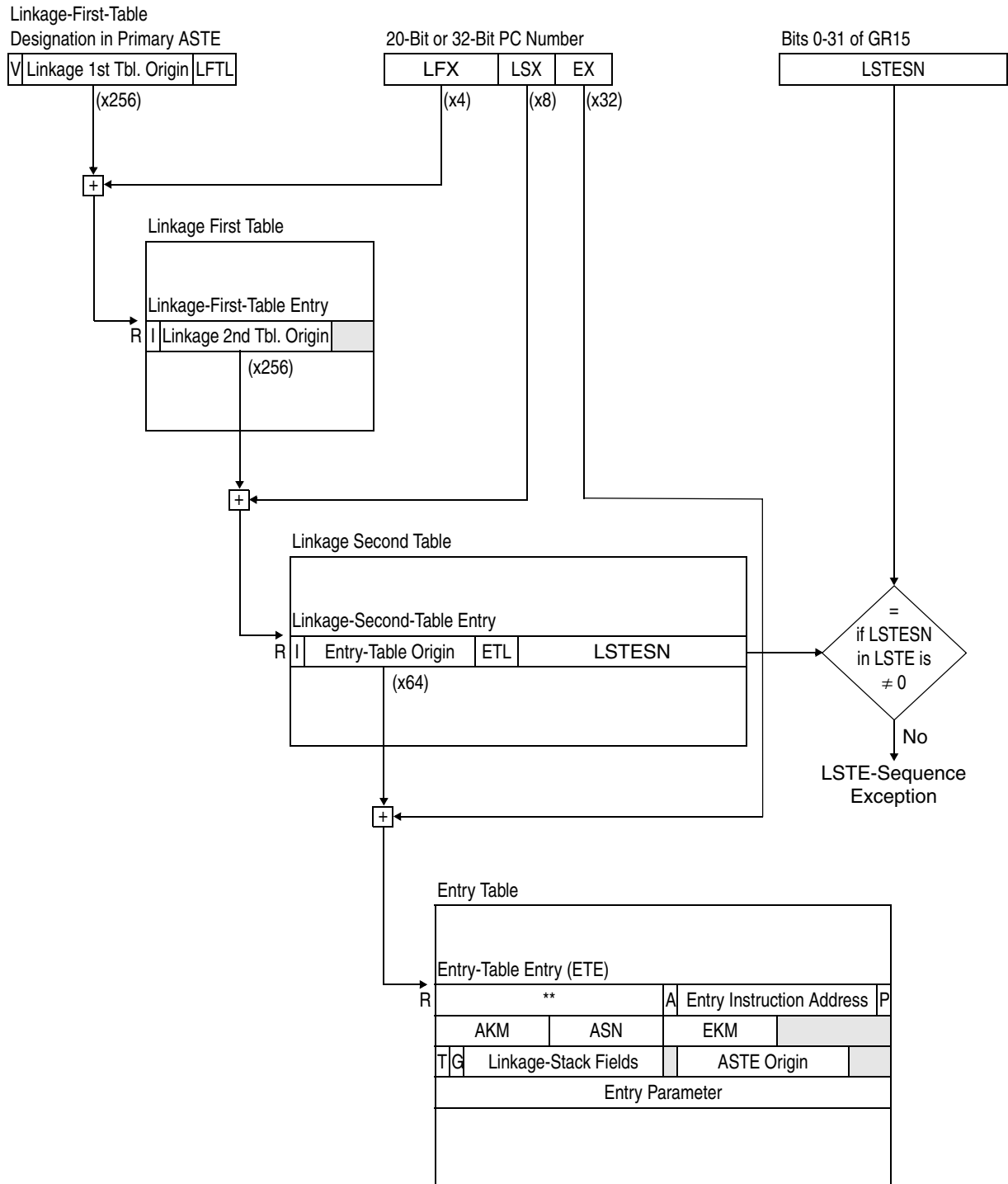
The PC-number-translation process is shown in Figure 5-8 on page 5-37 for when ASN-and-LX-reuse is not enabled and in Figure 5-9 on page 5-38 for when it is enabled.



R: Address is real.

**:: First word and A of ETE are bits 0-32 of entry-instruction address (EIA) if G is one.

Figure 5-8. PC-Number Translation when ASN-and-LX Reuse Is Not Enabled



R: Address is real.

*: PC Number is 32 bits if bit 44 of the effective address is one. Bit 44 is not shown in this case.

**: First word and A of ETE are bits 0-32 of entry-instruction address (EIA) if G is one.

Figure 5-9. PC-Number Translation when ASN-and-LX Reuse Is Enabled

Obtaining the Linkage-Table or Linkage-First-Table Designation

The linkage-table or linkage-first-table designation is obtained from bytes 24-27 of the primary ASN-second-table entry, the starting address of which is specified by the contents of control register 5.

The 31-bit real address of the linkage-table or linkage-first-table designation is obtained by appending six zeros on the right to the primary-ASTE origin, bits 33-57 of control register 5, and adding 24. The addition cannot cause a carry into bit position 0. The 31-bit address is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

All four bytes of the linkage-table or linkage-first-table designation appear to be fetched concurrently from the primary ASTE as observed by other CPUs. The fetch access is not subject to protection. When the storage address which is generated for fetching the linkage-table or linkage-first-table designation designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed. Besides the linkage-table or linkage-first-table designation, no other field in the primary ASTE is examined.

Linkage-Table Lookup

When the ASN-and-LX-reuse facility is not installed or the ASN-and-LX-reuse-control bit, bit 44 of control register 0, is zero, the field fetched from the primary ASTE is a linkage-table designation. The linkage-index (LX) portion of the PC number, in conjunction with the linkage-table origin, is used to select an entry from the linkage table.

The 31-bit real address of the linkage-table entry is obtained by appending seven zeros on the right to the contents of bit positions 1-24 of the linkage-table designation and adding the linkage index, with two rightmost and 17 leftmost zeros appended. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31} - 1$ to 0. The 31-bit address is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

As part of the linkage-table-lookup process, the leftmost seven bits of the linkage index are compared against the linkage-table length, bits 25-31 of the linkage-table designation, to establish whether the

addressed entry is within the linkage table. If the value in the linkage-table-length field is less than the value of the seven leftmost bits of the linkage index, an LX-translation exception is recognized.

All four bytes of the linkage-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address which is generated for fetching the linkage-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the linkage-table entry specifies whether the entry table corresponding to the linkage index is available. This bit is inspected, and, if it is one, an LX-translation exception is recognized.

When no exceptions are recognized in the process of linkage-table lookup, the entry fetched from the linkage table designates the origin and length of the corresponding entry table.

Linkage-First-Table Lookup

When the ASN-and-LX-reuse facility is installed and the ASN-and-LX-reuse-control bit, bit 44 of control register 0, is one, the field fetched from the primary ASTE is a linkage-first-table designation. The linkage-first-index (LFX) portion of the PC number, in conjunction with the linkage-first-table origin, is used to select an entry from the linkage first table.

When bit 44 of the effective address of PROGRAM CALL is zero, the PC number is 20 bits, the linkage first index is bits 44-50 of the effective address, and the 31-bit real address of the linkage-first-table entry is obtained by appending eight zeros on the right to the contents of bit positions 1-23 of the linkage-first-table designation and adding the linkage first index, with two rightmost and 22 leftmost zeros appended.

When bit 44 of the effective address of PROGRAM CALL is one, the PC number is 32 bits, the linkage first index is bits 32-43 of the effective address appended on the left of bits 45-50 of the effective address, and the 31-bit real address of the linkage-first-table entry is obtained by appending eight zeros on the right to the contents of bit positions 1-23 of the linkage-first-table designation and adding the linkage first index, with two rightmost and 11 leftmost zeros appended.

When bit 44 of the effective address is one and a carry into bit position 0 occurs during the addition to form the address of a linkage-first-table entry, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31} - 1$ to 0.

When bit 44 of the effective address is either zero or one, the 31-bit address of the linkage-first-table entry is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

As part of the linkage-first-table-lookup process when bit 44 of the effective address is one, the leftmost 12 bits of the linkage first index are compared against the linkage-first-table length, bits 24-31 of the linkage-first-table designation, to establish whether the addressed entry is within the linkage first table. For this comparison, the linkage-first-table length is extended with four zero bits on the left. If the value of the extended linkage-first-table length is less than the value of the 12 leftmost bits of the linkage first index, an LFX-translation exception is recognized.

All four bytes of the linkage-first-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address which is generated for fetching the linkage-first-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the linkage-first-table entry specifies whether the linkage second table corresponding to the linkage first index is available. This bit is inspected, and, if it is one, an LFX-translation exception is recognized.

When no exceptions are recognized in the process of linkage-first-table lookup, the entry fetched from the linkage first table designates the origin of the corresponding linkage second table.

Linkage-Second-Table Lookup

When the ASN-and-LX-reuse facility is installed and the ASN-and-LX-reuse-control bit, bit 44 of control register 0, is one, a linkage-second-table lookup is performed after a linkage-first-table entry has been fetched. The linkage-second-index (LSX) portion of the PC number, in conjunction with the linkage-second-table origin, is used to select an entry from the linkage second table.

The 31-bit real address of the linkage-second-table entry is obtained by appending eight zeros on the right to the linkage-second-table origin, bits 1-23 of the linkage-first-table entry, and adding the linkage second index, with three rightmost and 23 leftmost zeros appended. A carry into bit position 0 cannot occur during this addition. The 31-bit address is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

All eight bytes of the linkage-second-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address which is generated for fetching the linkage-second-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the linkage-second-table entry specifies whether the entry table corresponding to the linkage second index is available. This bit is inspected, and, if it is one, an LSX-translation exception is recognized.

When no exceptions are recognized in the process of linkage-second-table lookup, the entry fetched from the linkage second table designates the origin of the corresponding entry table. The linkage-second-table entry contains a linkage-second-table-entry sequence number that may be used to test the correctness of the use of the linkage index.

Linkage-Second-Table-Entry-Sequence-Number Comparison

The linkage-second-table entry contains a linkage-second-table-entry sequence number (LSTESN) in bit positions 32-63. If this LSTESN is nonzero, it is compared to an LSTESN in bit positions 0-31 of general register 15, and an LSTE-sequence exception is recognized if the two LSTESNs are not equal.

Entry-Table Lookup

When the ASN-and-LX-reuse facility is not installed or is not enabled, the entry-table entry is located from a linkage-table entry. When the ASN-and-LX-reuse facility is installed and enabled, the entry-table entry is located from a linkage-second-table entry.

The entry-index (EX) portion of the PC number, in conjunction with the entry-table origin contained in the linkage-table or linkage-second-table entry, is used to select an entry from the entry table.

The 31-bit real address of the entry-table entry is obtained by appending six zeros on the right to the entry-table origin bits 1-25 of the linkage-table or linkage-second-table entry, and adding the entry index, with five rightmost and 18 leftmost zeros appended. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the table to wrap from $2^{31} - 1$ to 0. The 31-bit address is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

As part of the entry-table-lookup process, the six leftmost bits of the entry index are compared against the entry-table length, bits 26-31 of the linkage-table or linkage-second-table entry, to establish whether the addressed entry is within the table. If the value in the entry-table length field is less than the value of the six leftmost bits of the entry index, an EX-translation exception is recognized.

The 32-byte entry-table entry is fetched by using the real address. The fetch of the entry appears to be word concurrent, as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining seven words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address which is generated for fetching the entry-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

The use that is made of the information fetched from the entry-table entry is described in the definition of the PROGRAM CALL instruction.

Recognition of Exceptions during PC-Number Translation

The exceptions which can be encountered during the PC-number-translation process and their priority are described in the definition of the PROGRAM CALL instruction.

Programming Note: The linkage-table or linkage-first-table designation is fetched successfully from the primary ASN-second-table entry regardless of the value of bit 0, the ASX-invalid bit, in the primary ASTE. A one value of this bit may cause an exception to be recognized in other circumstances.

Home Address Space

Facilities are provided which a privileged program, such as the control program, can use to obtain control in and access the home address space of a dispatchable unit (for example, a task).

Each dispatchable unit normally has an address space associated with it in which the control program keeps the principal control blocks that represent the dispatchable unit. This address space is called the home address space of the dispatchable unit. Different dispatchable units may have the same or different home address spaces. When the control program initiates a dispatchable unit, it may set the primary and secondary address spaces equal to the home address space of the dispatchable unit. Thereafter, because of the dispatchable unit's possible use of the PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, or SET SECONDARY ASN instruction, the control program normally cannot depend on either the primary address space or the secondary address space being the home address space when the home address space must be accessed, for example, during the processing by the control program of an interruption. Therefore, the control program normally must take some special action to ensure that the home address space is addressed when it must be accessed. The home-address-space facilities provide an efficient means to take this action.

The home-address-space facilities include:

- The home address-space-control element (HASCE) in control register 13. The HASCE is used by DAT in the same way as the primary address-space-control element (PASCE) in control register 1 and the secondary address-space-control element (SASCE) in control register 7.
- Home-space mode, which results when DAT is on and the address-space control, PSW bits 16 and 17, has the value 11 binary. When the CPU is in the home-space mode, instruction and logical addresses are home virtual addresses and are translated by DAT by means of the HASCE.
- The ability of the RESUME PROGRAM, SET ADDRESS SPACE CONTROL, and SET ADDRESS SPACE CONTROL FAST instructions to set the home-space mode in the supervisor state, and the ability of the INSERT ADDRESS

SPACE CONTROL instruction to return an indication of the home-space mode.

- The home space-switch-event control, bit 57 of control register 13.
- Recognition of a space-switch event upon completion of a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction if the CPU was in the home-space mode before or after the operation but not both before and after the operation, if any of the following is true: (1) the primary space-switch-event control, bit 57 of control register 1, is one, (2) the home space-switch-event control is one, or (3) a PER event is to be indicated.

The space-switch event can be used to enable or disable PER or tracing when fetching of instructions begins or ends in particular address spaces.

Access-Register Introduction

Many of the functions related to access registers are described in this section and in “Subroutine Linkage without the Linkage Stack” on page 5-11, “Access-Register Translation” on page 5-50, and “Sequence of Storage References” on page 5-85. Additionally, translation modes and access-list-controlled protection are described in Chapter 3, “Storage”; the PER means of restricting storage-alteration events to designated address spaces and the handling of access registers during resets and during the store-status operation are described in Chapter 4, “Control”; interruptions are described in Chapter 6, “Interruptions”; instructions are described in Chapter 7, “General Instructions”, and Chapter 10, “Control Instructions”; the handling of access registers during a machine-check interruption and the programmed validation of the access registers are described in Chapter 11, “Machine-Check Handling”; and the alter-and-display controls for access registers are described in Chapter 12, “Operator Facilities.”

Summary

These major functions are provided:

- A maximum of 16 address spaces, including the instruction space, for immediate and simultaneous use by a semiprivileged program; the

address spaces are specified by 16 registers called access registers.

- Instructions for examining and changing the contents of the access registers.

In addition, control and authority mechanisms are incorporated to control these functions.

Access registers allow a sequence of instructions, or even a single instruction such as MOVE (MVC) or MOVE LONG (MVCL), to operate on storage operands in multiple address spaces, without the requirement of changing either the translation mode or other control information. Thus, a program residing in one address space can use the complete instruction set to operate on data in that address space and in up to 15 other address spaces, and it can move data between any and all pairs of these address spaces. Furthermore, the program can change the contents of the access registers in order to access still other address spaces.

The instructions for examining and changing access-register contents are unprivileged and are described in Chapter 7, “General Instructions.” They are:

- COPY ACCESS
- EXTRACT ACCESS
- LOAD ACCESS MULTIPLE
- LOAD ADDRESS EXTENDED
- SET ACCESS
- STORE ACCESS MULTIPLE

The privileged PURGE ALB instruction and COMPARE AND SWAP AND PURGE instruction are used in connection with access registers and are described in Chapter 10, “Control Instructions.”

Access registers specify address spaces when the CPU is in the access-register mode. The SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST instructions allow setting of the access-register mode, and the INSERT ADDRESS SPACE CONTROL instruction provides an indication of the access-register mode. The stacking PROGRAM CALL, PROGRAM RETURN, and RESUME PROGRAM instructions also allow setting of the access-register mode. All of these instructions are described in Chapter 10, “Control Instructions.”

Access registers are used in a special way by the BRANCH IN SUBSPACE GROUP instruction. The use of access registers by that instruction is

described in detail only in the definition of the instruction in Chapter 10, “Control Instructions.” However, “Subspace-Group Tables” on page 5-62 describes the use of the dispatchable-unit control table and the extended ASN-second-table entry by BRANCH IN SUBSPACE GROUP.

Access-Register Functions

Access-Register-Specified Address Spaces

The CPU includes sixteen 32-bit access registers numbered 0-15. In the access-register mode, which results when DAT is on and PSW bits 16 and 17 are 01 binary, an instruction B or R field that is used to specify the logical address of a storage operand designates not only a general register but also an access register. The designated general register is used in the ordinary way to form the logical address of the storage operand. The designated access register is used to specify the address space to which the logical address is relative. The access register specifies the address space by specifying an address-space-control element for the address space, and this address-space-control element is used by DAT to translate the logical address. An access register specifies an address-space-control element in an indirect way, not by containing the address-space-control element.

An access register may specify the primary or secondary address-space-control element in control register 1 or 7, respectively, or it may specify an address-space-control element contained in an ASN-second-table entry. In the latter case, the access register designates an entry in a table called an access list, and the designated access-list entry in turn designates the ASN-second-table entry.

The process of using the contents of an access register to obtain an address-space-control element for

use by DAT is called access-register translation (ART). This is depicted in Figure 5-10.

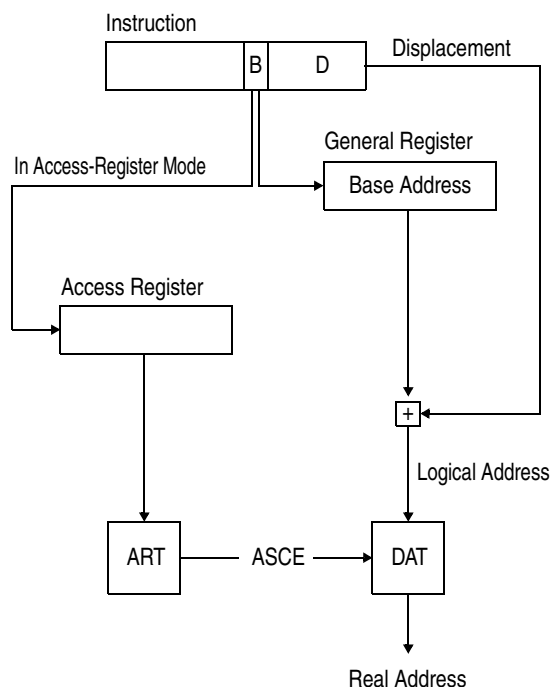


Figure 5-10. Use of Access Registers

An access register is said to specify an AR-specified address space by means of an AR-specified address-space-control element. The virtual addresses in an AR-specified address space are called AR-specified virtual addresses.

In the access-register mode, whereas all storage-operand addresses are AR-specified virtual, instruction addresses are primary virtual.

Designating Access Registers: In the access-register mode, an instruction B or R field designates an access register, for use in access-register translation, under the following conditions:

- The field is a B field which designates a general register containing a base address. The base address is used, along with a displacement (D) and possibly an index (X), to form the logical address of a storage operand.
- The field is an R field which designates a general register containing the logical address of a storage operand.

For example, consider the following instruction:

MVC 0(L,1),0(2)

The second operand, of length L, is to be moved to the first-operand location. The logical address of the second operand is in general register 2, and that of the first-operand location in general register 1. The address space containing the second operand is specified by access register 2, and that containing the first-operand location by access register 1. These two address spaces may be different address spaces, and each may be different from the current instruction address space (the primary address space).

When PSW bits 16 and 17 are 01, the B₂ field of the LOAD REAL ADDRESS and STORE REAL ADDRESS instructions designates an access register, for use in access-register translation, regardless of whether DAT is on or off. When the M₄ field of the LOAD PAGE-TABLE-ENTRY ADDRESS instruction is 0001 binary, or when the M₄ field is 0100 binary and bits 16-17 of the PSW are 01 binary, then the R₂ field designates an access register, for use in access-register translation, regardless of whether DAT is on or off.

The COMPARE AND FORM CODEWORD and UPDATE TREE instructions specify storage operands by means of implicitly designated general registers and access registers.

The MOVE TO PRIMARY and MOVE TO SECONDARY instructions specify storage operands by means of primary virtual and secondary virtual addresses, and access registers do not apply to these instructions. An exception is recognized when either of these instructions is executed in the access-register mode. The MOVE WITH KEY instruction can be used in place of MOVE TO PRIMARY and MOVE TO SECONDARY in the access-register mode. The MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY instructions also can be used.

An instruction R field may designate an access register for other than the purpose of access-register translation.

The fields which may designate access registers, whether or not for access-register translation, are indicated in the summary figure at the beginning of each instruction chapter.

Obtaining the Address-Space-Control Element:

This section and the following ones introduce the access-register-translation process and present the concepts related to access lists.

The address-space-control element specified by an access register is obtained by access-register translation as follows:

- If the access register contains 00000000 hex, the specified address-space-control element is the primary address-space-control element (PASCE), obtained from control register 1.
- If the access register contains 00000001 hex, the specified address-space-control element is the secondary address-space-control element (SASCE), obtained from control register 7.
- If the access register contains any other value, the specified address-space-control element is obtained from an ASN-second-table entry. The contents of the access register designate an access-list entry that contains the real origin of the ASN-second-table entry.

Access register 0 is treated in a special way by access-register translation; it is treated as containing 00000000 hex, and its actual contents are not examined. Thus, a logical address specified by means of a zero B or R field in the access-register mode is always relative to the primary address space, regardless of the contents of access register 0. However, there is one exception to how access register 0 is treated: the TEST ACCESS instruction uses the actual contents of access register 0, instead of treating access register 0 as containing 00000000 hex.

The treatment of an access register containing the value 00000000 hex as designating the current primary address space allows that address space to be addressed, in the access-register mode, without requiring the use of an access-list entry. This is useful when the primary address space is changed by a space-switching PROGRAM CALL (PC-ss), PROGRAM RETURN (PR-ss), or PROGRAM TRANSFER (PT-ss) instruction. Similarly, the treatment of an access register containing the value 00000001 hex as designating the secondary address space allows that space to be addressed after a space-switching

operation, again without requiring the use of an access-list entry.

The contents of the access registers are not changed by the PROGRAM CALL and PROGRAM TRANSFER instructions. Therefore, an access register containing 00000000 or 00000001 hex may specify a different address space after the execution of PROGRAM CALL or PROGRAM TRANSFER than before the execution. For example, if a space-switching PROGRAM CALL instruction is executed, an access register containing 00000000 hex specifies the old primary address space before the execution and the new primary address space after the execution.

Access Lists: The access-list entry that is designated by the contents of an access register can be located in either one of two access lists, the dispatchable-unit access list or the primary-space access list. A bit in the access register specifies which of the two access lists contains the designated entry. Both of the access lists reside in real or absolute storage. The locations of the access lists are specified by means of control registers 2 and 5.

Control register 2 contains the origin of a real-storage area called the dispatchable-unit control table. The dispatchable-unit control table contains the designation — the real or absolute origin, and length — of the dispatchable-unit access list.

Control register 5 contains the origin of a real-storage area called the primary ASN-second-table entry. The primary ASN-second-table entry contains the designation of the primary-space access list.

An access list, either the dispatchable-unit access list or the primary-space access list, contains some multiple of eight 16-byte entries, up to a maximum of 1,024 entries.

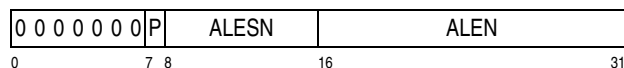
Programs and Dispatchable Units: When discussing access lists, it is necessary to distinguish between the terms “program” and “dispatchable unit.” A program is a sequence of instructions and may be referred to as a program module. A program may be a sequence of calling and called programs. A dispatchable unit, which is sometimes called a process or a task, is a unit of work that is performed through the execution of a program by one CPU at a time. The dispatchable-unit access list is intended to be associated with a dispatchable unit; that is, it is intended that a dispatchable unit have the same dispatchable-unit access list regardless of which pro-

gram is currently being executed to perform the dispatchable unit. There is no mechanism, except for the LOAD CONTROL instruction, that changes the dispatchable-unit-control-table origin in control register 2.

The primary-space access list is associated with the primary address space that is specified by the primary ASN in control register 4 and the primary address-space-control element in control register 1. The primary-space access list that is available for use by a dispatchable unit changes as the primary address space of the dispatchable unit changes, that is, whenever a program in a different primary address space begins to be executed to perform the dispatchable unit. Whenever a LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL, PROGRAM RETURN, or PROGRAM TRANSFER instruction replaces the primary ASN in control register 4 and the primary address-space-control element in control register 1, it also replaces the primary-ASN-second-table-entry origin in control register 5.

Thus, for a dispatchable unit, the dispatchable-unit access list is intended to be constant (although its entries may be changed, as will be described), and the primary-space access list is a function of which program is being executed, through being a function of the primary address space of the program. Also, all dispatchable units and programs in the same primary address space have the same primary-space access list.

Access-List-Entry Token: The contents of an access register are called an access-list-entry token (ALET) since, in the general case, they designate an entry in an access list. An ALET has the following format:



The ALET contains a primary-list bit (P) that specifies which access list contains the designated access-list entry: the dispatchable-unit access list if the bit is zero, or the primary-space access list if the bit is one. The specified access list is called the effective access list.

The ALET also contains an access-list-entry number (ALEN) which, when multiplied by 16, is the number of bytes from the beginning of the effective access list to the designated access-list entry. During access-

register translation, an exception is recognized if the ALEN designates an entry that is outside the effective access list or if the leftmost seven bits in the ALET are not all zeros.

The access-list-entry sequence number (ALESN) in the ALET is described in the next section.

The above format of the ALET does not apply when the ALET is 00000000 or 00000001 hex.

An ALET can exist in an access register, in a general register, or in storage, and it has no special protection from manipulation by the problem program. Any program can transfer ALETs back and forth among access registers, general registers, and storage. A called program can save the contents of the access registers in any storage area available to it, load and use the access registers for its own purposes, and then restore the original contents of the access registers before returning to its caller.

Allocating and Invalidating Access-List Entries: It is intended that access lists be provided by the control program and that they be protected from direct manipulation by any problem program. This protection may be obtained by means of key-controlled protection or by placing the access lists in real storage not accessible by any problem program by means of DAT.

As determined by a bit in the entry, an access-list entry is either valid or invalid. A valid access-list entry specifies an address space and can be used by a suitably authorized program to access that space. An invalid access-list entry is available for allocation as a valid entry. It is intended that the control program provide services that allocate valid access-list entries and that invalidate previously allocated entries.

Allocation of an access-list entry may consist in the following steps. A problem program passes some kind of identification of an address space to the control program, and it passes a specification of either the dispatchable-unit access list or the primary-space access list. The control program checks, by some means, the authority of the problem program to access the address space. If the problem program is authorized, the control program selects an invalid entry in the specified access list, changes it to a valid entry specifying the subject address space, and returns to the problem program an access-list-entry token (ALET) that designates the allocated entry. The

problem program can subsequently place the ALET in an access register in order to access the address space. Later, through the use of the invalidation service of the control program, the access-list entry that was allocated may be made invalid. An exception is recognized during access-register translation if an ALET is used that designates an invalid access-list entry.

It may be that a particular access-list entry is allocated, then invalidated, and then allocated again, this time specifying a different address space than the first time. To guard against erroneous use of an ALET that designates a conceptually wrong address space, an access-list-entry sequence number (ALESN) is provided in both the ALET and the access-list entry. When the control program allocates an access-list entry, it should place the same ALESN in the entry and in the designating ALET that it returns to the problem program. When the control program reallocates an access-list entry, it should change the value of the ALESN. An exception is recognized during access-register translation if the ALESN in the ALET used is not equal to the ALESN in the designated access-list entry.

The ALESN check is a reliability mechanism, not an authority mechanism, because the ALET is not protected from the problem program, and the problem program can change the ALESN in the ALET to any value. Also, this is not a fail-proof reliability mechanism because the ALESN is one byte and its value wraps around after 256 reallocations, assuming that the value is incremented by one for each reallocation.

Authorizing the Use of Access-List Entries: Although an access list is intended to be associated with either a dispatchable unit or a primary address space, the valid entries in the list are intended to be associated with the different programs that are executed, in some order, to perform the work of the dispatchable unit. It is intended that each program be able to have a particular authority that permits the use of only those access-list entries that are associated with the program. The authority being referred to here is represented by a 16-bit extended authorization index (EAX) in control register 8.

Other elements used in the related authorization mechanism are: (1) a private bit in the access-list entry, (2) an access-list-entry authorization index (ALEAX) in the access-list entry, and (3) the authority table.

A program is authorized to use an access-list entry, in access-register translation, if any of the following conditions is met:

1. The private bit in the access-list entry is zero. This condition provides a high-performance means to authorize any and all programs that are executed to perform the dispatchable unit.
2. The ALEAX in the access-list entry is equal to the EAX in control register 8. This condition provides a high-performance means to authorize only particular programs.
3. The EAX selects a secondary bit that is one in the authority table associated with the address space that is specified by the access-list entry. The authority table is locatable in that the access-list entry contains the real origin of the ASN-second-table entry (ASTE) for the address space, and the ASTE contains the real origin of

the authority table. This condition provides another means, less well-performing than condition 2, for authorizing only particular programs. However, providing for condition 3 to be met instead of condition 2 can be advantageous because it permits several programs, each executed with a different EAX, all to use a single access-list entry to access a particular address space.

Access-register translation tests for the three conditions in the order indicated by their numbers, and a higher-numbered condition is not tested for if a lower-numbered condition is met. An exception is recognized if none of the conditions is met.

Figure 5-11 shows an example of how the authorization mechanism can be used. In the figure, “PB=0” means that the private bit is zero, and “PB=1” means that the private bit is one.

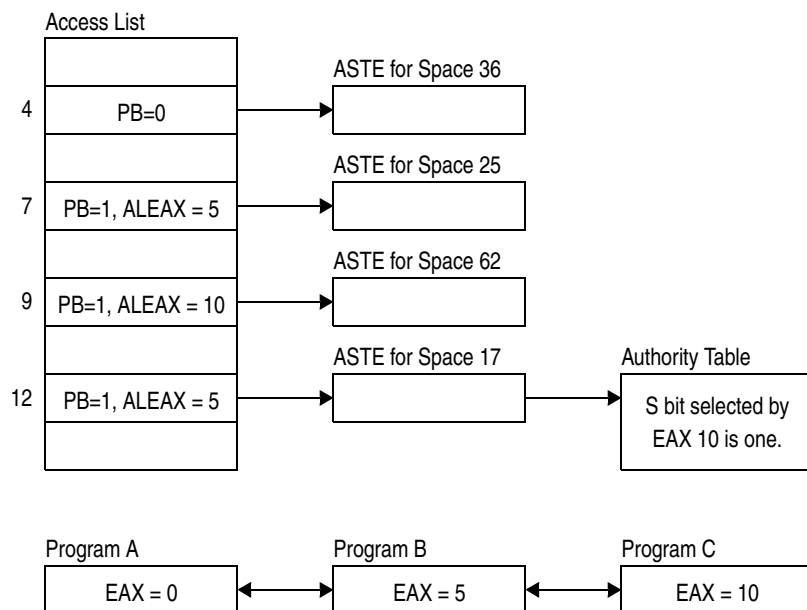


Figure 5-11. Example of Authorizing the Use of Access-List Entries

The figure shows an access list — assume it is a dispatchable-unit access list — in which the entries of interest are entries 4, 7, 9, and 12. Each access-list entry contains a private bit, an ALEAX, and the real origin of the ASTE for an address space. The private bit in entry 4 is zero, and, therefore, the value of the ALEAX in entry 4 is immaterial and is not shown. The private bits in entries 7, 9, and 12 are ones, and the ALEAX values in these entries are as shown. The

numbers used to identify the address spaces (36, 25, 62, and 17) are arbitrary. They may be the ASNs of the address spaces; however, ASNs are in no way used in access-register translation. Only the authority table for address space 17 is shown. In it, the secondary bit selected by EAX 10 is one. Assume that no secondary bits are ones in the authority tables for the other spaces.

The figure also shows a sequence of three programs, named A, B, and C, that is executed to perform the work of the dispatchable unit associated with the access list. These programs may be in the same or different address spaces. The EAX in control register 8 when each of these programs is executed is 0, 5, and 10, respectively.

Each of programs A, B, and C can use access-list entry (ALE) 4 to access address space 36 since the private bit in ALE 4 is zero. Program B can use ALE 7 to access space 25 because the ALEAX in the ALE equals the EAX for the program, and no other program can use this ALE. Similarly, only program C can use ALE 9. Program B can use ALE 12 because the ALEAX and EAX are equal, and program C can use it because A's EAX selects a secondary bit that is one in the authority table for space 17.

The example would be the same if programs A, B, and C were all in the same address space and the access list were the primary-space access list for that space.

An ALE in which the private bit is zero may be called public because the ALE can be used by any program, regardless of the value of the current EAX. An ALE in which the private bit is one may be called private because the ability of a program to use the ALE depends on the current EAX.

Notes on the Authorization Mechanism: An access list is a kind of capability list, in the sense in which the word "capability" is used in computer science. It is up to the control program to formulate the policies that are used to allocate entries in an access list, and the programmed authorization checking required during allocation may be very complex and lengthy. After a valid entry has been made in an access list, the access-register-translation process enforces the control-program policies in a well-performing way by means of the authorization mechanism described above.

Using access lists has an advantage over using only ASNs and authority tables. For example, assume that an access register could contain an ASN and that access-register translation would do ASN translation of the ASN and then use the EAX to test the authority table. This would make the EAX relevant to all existing address spaces, and, therefore, it would make the management of EAXs and their assignment to programs more difficult. With the actual definitions of the ALET and access-register translation, an EAX is

relevant to only the address spaces that are represented in the current dispatchable-unit and primary-space access lists. Also, since ASN translation is not done as a part of access-register translation, the number of concurrently existing address spaces, as represented by ASN-second-table entries, can be greater than the number of available ASNs (64K).

The entry-table entry and linkage stack can be used to assign EAXs to programs and to change the EAX in control register 8 during program linkages. These components are introduced in "Linkage-Stack Introduction" on page 5-66. The privileged EXTRACT AND SET EXTENDED AUTHORITY instruction also is available for saving and changing the EAX in control register 8.

The SET SECONDARY ASN instruction and the authorization index (AX), bits 32-47 of control register 4, can play a role in the use of access registers. The space-switching form of SET SECONDARY ASN (SSAR-ss) establishes a new secondary address space if the secondary bit selected by the AX is one in the authority table associated with the new secondary space. The secondary space can be addressed by means of an ALET having the value 00000001 hex.

Revoking Accessing Capability: Another mechanism, which is a combined authority and integrity mechanism, is part of access-register translation, and it is described in this section.

An access-list entry (ALE) contains an ASN-second-table-entry sequence number (ASTESN), and so does the ASTE designated by the ALE. During access-register translation, the ASTESN in the ALE must equal the ASTESN in the designated ASTE; otherwise, an exception is recognized.

When the control program allocates an ALE, it should copy the ASTESN from the designated ASTE into the ALE. Subsequently, the control program can, in effect, revoke the addressing capability represented by the ALE by changing the ASTESN in the ASTE. Changing the ASTESN in the ASTE makes all previously usable ALEs that designate the ASTE unusable.

Making an ALE unusable may be required in either of two cases:

1. Some element of the control-program policy for determining the authority of a program to have

access to the address space specified by the ASTE has changed. This may mean that some or all of the programs that were authorized to the address space, and for which ALEs have been allocated, are no longer authorized. Changing the ASTESN in the ASTE ends the usability of all ALEs that designate the ASTE. If this revocation of capability is to be selective, then, when an exception is recognized because of unequal ASTESNs, the control program can reapply its programmed procedures for determining authorization, and an ALE which should have remained usable can be made usable again by copying the new ASTESN into it. When the usability of an ALE is restored, the control program normally should cause reexecution of the instruction that encountered the exception.

2. The ASTE has been reassigned to specify a conceptually different address space, and ALEs which specified the old address space must not be allowed to specify the new one. (Bit 0 of the ASTE, the ASX-invalid bit, can be set to one to delete the assignment of the ASTE to an address space, and this prevents the use of the ASTE in access-register translation. But after reassignment, bit 0 normally is set back to zero.)

The ASTESN mechanism may be regarded as an authority mechanism in the first case above and as an integrity mechanism in the second.

The ASTESN mechanism is especially valuable because it avoids the need of the control program to keep track of the access lists that contain the ALEs that designate each ASTE. Furthermore, it avoids the need of searching through these access lists in order to find the ALEs and set them invalid, to prevent the use of the ALEs in access-register translation. The latter activity could be particularly time-consuming, or could present a particularly difficult management problem, because the access lists could be in auxiliary storage, such as a direct-access storage device, when the need arises to invalidate the ALEs.

The ASTESN is a four-byte field. Assuming a reasonable frequency of authorization-policy changes or address-space reassignments, the approximately four billion possible values of the ASTESN provide a fail-proof authority or integrity mechanism over the lifetime of the system.

Preventing Store References: The access-list entry contains a fetch-only bit which, when one,

specifies that the access-list entry cannot be used to perform storage-operand store references. The principal description of the effect of the fetch-only bit is in "Access-List-Controlled Protection" on page 3-12.

Improving Translation Performance: Access-register translation (ART) conceptually occurs each time a logical address is used to reference a storage operand in the access-register mode. To improve performance, ART normally is implemented such that some or all of the information contained in the ART tables (access-list-designation sources, access lists, ASN second tables, and authority tables) is maintained in a special buffer referred to as the ART-lookaside buffer (ALB). The CPU necessarily refers to an ART-table entry in real storage only for the initial access to that entry. The information in the entry may be placed in the ALB, and subsequent translations may be performed using the information in the ALB.

The PURGE ALB instruction and the COMPARE AND SWAP AND PURGE instruction can be used to clear all information from the ALB after a change has been made to an ART-table entry in real storage.

Access-Register Instructions

The following instructions are provided for examining and changing the contents of access registers:

- COPY ACCESS
- EXTRACT ACCESS
- LOAD ACCESS MULTIPLE
- LOAD ADDRESS EXTENDED
- SET ACCESS
- STORE ACCESS MULTIPLE

The SET ACCESS instruction replaces the contents of a specified access register with the contents of a specified general register. Conversely, the EXTRACT ACCESS instruction moves the contents of an access register to a general register. The COPY ACCESS instruction moves the contents of one access register to another.

The LOAD ACCESS MULTIPLE instruction loads a specified set of consecutively numbered access registers from a specified storage location whose length in words equals the number of access registers loaded. Conversely, the STORE ACCESS MULTIPLE instruction function stores the contents of a set of access registers at a storage location.

The LOAD ADDRESS EXTENDED instruction is similar to the LOAD ADDRESS instruction in that it loads a specified general register with an effective address specified by means of the B, X, and D fields of the instruction. In addition, LOAD ADDRESS EXTENDED operates on the access register having the same number as the general register loaded. When the address-space control, PSW bits 16 and 17, is 00, 10, or 11 binary, LOAD ADDRESS EXTENDED loads the access register with 00000000, 00000001, or 00000002 hex, respectively. When the address space control is 01 binary, LOAD ADDRESS EXTENDED loads the target access register with a value that depends on the B field of the instruction. If the B field is zero, LOAD ADDRESS EXTENDED loads the target access register with 00000000 hex. If the B field is nonzero, LOAD ADDRESS EXTENDED loads the target access register with the contents of the access register designated by the B field. However, in the last case when bits 0-6 of the access register designated by the B field are not all zeros, the results in the target general register and access register are unpredictable.

The address-space-control values 00, 01, 10, and 11 binary specify primary-space, access-register, secondary-space, and home-space mode, respectively, when DAT is on. LOAD ADDRESS EXTENDED functions the same regardless of whether DAT is on or off.

When used in access-register translation, the access-register values 00000000 and 00000001 hex specify the primary and secondary address spaces, respectively, and the value 00000002 hex designates entry 2 in the dispatchable-unit access list. Loading the target access register with 00000002 hex when the address-space control is 11 binary is intended to support assignment, by the control program, of entry 2 in the dispatchable-unit access list as specifying the home address space.

Access-Register Translation

Access-register translation is introduced in “Access-Register-Specified Address Spaces” on page 5-43.

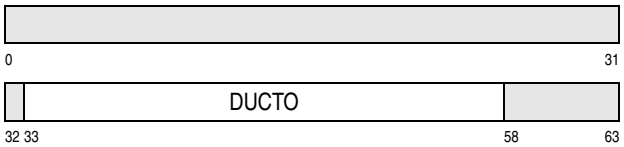
Access-Register-Translation Control

Access-register translation is controlled by an address-space control and by controls in control registers 2, 5, and 8. The address-space control, PSW bits 16 and 17, is described in “Translation Modes” on page 3-36. The other controls are described below.

Additional controls are located in the access-register-translation tables.

Control Register 2

The location of the dispatchable-unit control table is specified in control register 2. The register has the following format:



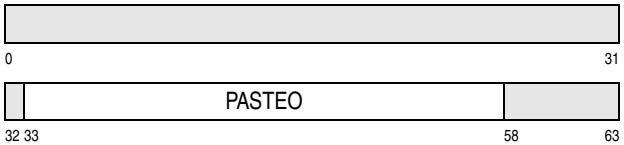
Dispatchable-Unit-Control-Table

Origin

(DUCTO): Bits 33-57 of control register 2, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the dispatchable-unit control table. Access-register translation may obtain the dispatchable-unit access-list designation from the dispatchable-unit control table.

Control Register 5

The location of the primary ASN-second-table entry is specified in control register 5. The register has the following format:

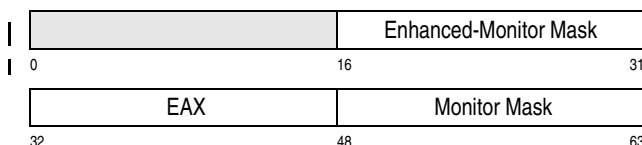


Primary-ASTE Origin (PASTEO): Bits 33-57 of control register 5, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the primary ASN-second-table entry. Access-register translation may obtain the primary-space access-list designation from the primary ASTE. The primary-ASTE origin is set by LOAD ADDRESS SPACE PARAMETERS when it performs PASN translation and by the space-switching forms of PROGRAM CALL, PROGRAM RETURN, and PROGRAM TRANSFER. When any of these instruc-

tions places the primary-ASTE origin in control register 5, it also places zeros in bit positions 32 and 58-63 of the register and leaves bits 0-31 of the register unchanged. Bits 0-32 and 58-63 of control register 5 are subject to possible future assignment, and they should not be depended upon to be zeros.

Control Register 8

The extended authorization index is in control register 8. The register has the following format:



Extended Authorization Index (EAX): Bits 32-47 of control register 8 are the extended authorization index. During access-register translation, the EAX may be compared against the access-list-entry authorization index (ALEAX) in an access-list entry, and it may be used as an index to locate a secondary bit in an authority table. The EAX may be set by a stacking PROGRAM CALL operation, and it is restored by PROGRAM RETURN. The EAX can also be saved and set by the privileged instruction EXTRACT AND SET EXTENDED AUTHORITY.

Access Registers

There are sixteen 32-bit access registers numbered 0-15. The contents of an access register are called an access-list-entry token (ALET). An ALET has the following format:



The fields in the ALET are allocated as follows:

Primary-List Bit (P): When the ALET is not 00000000 or 00000001 hex, bit 7 specifies the access list to be used by access-register translation. When bit 7 is zero, the dispatchable-unit access list is used; this is specified by the dispatchable-unit access-list designation in the dispatchable-unit control table designated by the contents of control register 2. When bit 7 is one, the primary-space access list is used; this is specified by the primary-space access-list designation in the primary ASTE designated by the contents of control register 5.

Access-List-Entry Sequence Number (ALESN):

Bits 8-15 may be used as a check on whether the access-list entry designated by the ALET has been invalidated and reallocated since the ALET was obtained. During access-register translation when the ALET is not 00000000 or 00000001 hex, bits 8-15 of the ALET are compared against the access-list-entry sequence number (ALESN) in the designated access-list entry.

Access-List-Entry Number (ALEN):

When the ALET is not 00000000 or 00000001 hex, bits 16-31 of the ALET designate an entry in either the dispatchable-unit access list or the primary-space access list, as determined by bit 7. The access-list designation that is used is called the effective access-list designation; it consists of the effective access-list origin and the effective access-list length.

During access-register translation, the ALEN, with four zeros appended on the right, is added to the 31-bit real or absolute address specified by the effective access-list origin, and the result is the real or absolute address of the designated access-list entry. The ALEN is compared against the effective access-list length to determine whether the designated access-list entry is within the list, and an ALEN-translation exception is recognized if the entry is outside the list. Although the largest possible value of the ALEN is 65,535, an access list can contain at most 1,024 entries.

Bits 0-6 must be zeros during access-register translation; otherwise, an ALET-specification exception is recognized.

When the ALET is 00000000 or 00000001 hex, it specifies the primary or secondary address space, respectively, and the above format does not apply.

Access register 0 usually is treated in access-register translation as containing 00000000 hex, and its actual contents are not examined; the access-register translation done as part of TEST ACCESS is the only exception. Access register 0 is also treated as containing 00000000 hex when it is designated by the B field of LOAD ADDRESS EXTENDED when PSW bits 16 and 17 are 01 binary. When access register 0 is specified for TEST ACCESS or as a source for COPY ACCESS, EXTRACT ACCESS, or STORE ACCESS MULTIPLE, the actual contents of the access register are used. Access register 0, like any other access register, can be loaded by COPY

ACCESS, LOAD ACCESS MULTIPLE, LOAD ADDRESS EXTENDED, and SET ACCESS.

Another definition of ALETs 00000000 and 00000001 hex is given in “BRANCH IN SUBSPACE GROUP” on page 10-13.

Access-Register-Translation Tables

When the ALET being translated is not 00000000 or 00000001 hex, access-register translation performs a two-level lookup to locate first the effective access-list designation and then an entry in the effective access list. The effective access-list designation resides in real storage. The effective access list resides in real or absolute storage.

Access-register translation uses an origin in the access-list entry to locate an ASN-second-table entry, and it may perform a one-level lookup to locate an entry in an authority table. The ASN-second-table entry resides in real storage. The authority table resides in real or absolute storage.

Authority-table entries are described in “Authority-Table Entries” on page 3-32. Access-list designations, access-list entries, and ASN-second-table entries are described in the following sections.

Dispatchable-Unit Control Table and Access-List Designations

When the ALET being translated is not 00000000 or 00000001 hex, access-register translation obtains the dispatchable-unit access-list designation if bit 7 of the ALET is zero, or it obtains the primary-space access-list designation if bit 7 is one. The obtained access-list designation is called the effective access-list designation.

The dispatchable-unit access-list designation (DUALD) is located in bytes 16-19 of a 64-byte area called the dispatchable-unit control table (DUCT). The DUCT resides in real storage, and its location is specified by the DUCT origin in control register 2.

The dispatchable-unit control table has the following format:

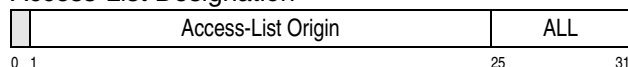
Hex	Dec	
0	0	BASTEO
4	4	SA SSASTEO
8	8	
C	12	SSASTESN
10	16	DUALD (see below)
14	20	PSW-Key Mask PSW Key RA P
18	24	
1C	28	////////////////////////////////////
In the 24-Bit or 31-Bit Addressing Mode		
20	32	
24	36	BA Bits 33-63 of Return Address
In the 64-Bit Addressing Mode		
20	32	Bits 0-31 of Return Address
24	36	Bits 32-63 of Return Address
28	40	
2C	44	Trap-Control-Block Address E
30	48	
34	52	
38	56	
3C	60	

Bytes 0-7 (BASTEO, SA, and SSASTEO) and 12-15 (SSASTESN) of the DUCT are described in “Subspace-Group Dispatchable-Unit Control Table” on page 5-62. Bytes 20-23 (PSW key mask, PSW key, RA, and P) and 32-39 (BA and return address) are described in “BRANCH AND SET AUTHORITY” on page 10-6. Bytes 44-47 (trap-control-block address and E) are described in “TRAP” on page 10-159. Bytes 8-11, 24-27, 40-43, and 48-63 are reserved for possible future extensions and should contain all zeros. Bytes 28-31 are available for use by programming.

The primary-space access-list designation (PSALD) is located in bytes 16-19 of a 64-byte area called the primary ASN-second-table entry. The primary ASTE resides in real storage, and its location is specified by the primary-ASTE origin in control register 5. The format of the primary ASTE is described in “ASN-Second-Table Entries” on page 5-54.

The dispatchable-unit and primary-space access-list designations both have the same format, which is as follows:

Access-List Designation



The fields in the access-list designation are allocated as follows:

Access-List Origin: Bits 1-24 of the access-list designation, with seven zeros appended on the right, form a 31-bit address that designates the beginning of the access list. This address is treated unpredictably as either a real address or an absolute address.

Access-List Length (ALL): Bits 25-31 of the access-list designation specify the length of the access list in units of 128 bytes, thus making the length of the access list variable in multiples of eight 16-byte entries. The length of the access list, in units of 128 bytes, is one more than the value in bit positions 25-31. The access-list length, with six zeros appended on the left, is compared against bits 0-12 of an access-list-entry number (bits 16-28 of an access-list-entry token) to determine whether the access-list-entry number designates an entry in the access list.

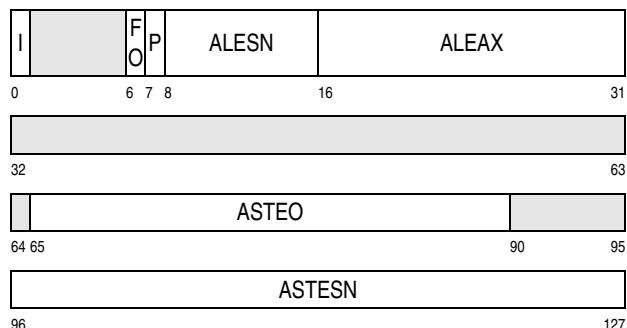
Bit 0 is reserved for a possible future extension and should be zero.

Programming Note: The maximum number of access-list entries allowed by an access-list designation is 1,024. There are two access lists available for use at any time. Therefore, a maximum of 2,048 16E-byte address spaces can be addressable without control-program intervention, which is a total of 2^{75} bytes.

Access-List Entries

The effective access list is the dispatchable-unit access list if bit 7 of the ALET being translated is zero, or it is the primary-space access list if bit 7 is

one. The entry fetched from the effective access list is 16 bytes in length and has the following format:



The fields in the access-list entry are allocated as follows:

ALEN-Invalid Bit (I): Bit 0, when zero, indicates that the access-list entry specifies an address space. When bit 0 is one during access-register translation, an ALEN-translation exception is recognized.

Fetch-Only Bit (FO): Bit 6 controls which types of operand references are permitted to the address space specified by the access-list entry. When bit 6 is zero, both fetch-type and store-type references are permitted. When bit 6 is one, only fetch-type references are permitted, and an attempt to store causes a protection exception for access-list-controlled protection to be recognized and the operation to be suppressed.

Private Bit (P): Bit 7, when zero, specifies that any program is authorized to use the access-list entry in access-register translation. When bit 7 is one, authorization is determined as described for bits 16-31.

Access-List-Entry Sequence Number (ALESN): Bits 8-15 are compared against the ALESN in the ALET during access-register translation. Inequality causes an ALE-sequence exception to be recognized. It is intended that the control program change bits 8-15 each time it reallocates the access-list entry.

Access-List-Entry Authorization Index (ALEAX): Bits 16-31 may be used to determine whether the program for which access-register translation is being performed is authorized to use the access-list entry. The program is authorized if any of the following conditions is met:

1. Bit 7 is zero.

- Bits 16-31 are equal to the extended authorization index (EAX) in control register 8.
- The EAX selects a secondary bit that is one in the authority table for the specified address space.

ASN-Second-Table-Entry Origin (ASTE_O): Bits 65-89, with six zeros appended on the right, form the 31-bit real address of the ASTE for the specified address space. Access-register translation obtains the address-space-control element for the address space from the ASTE.

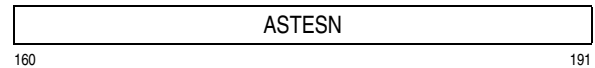
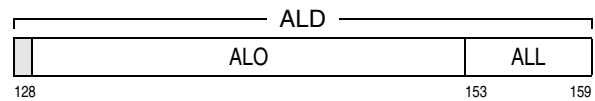
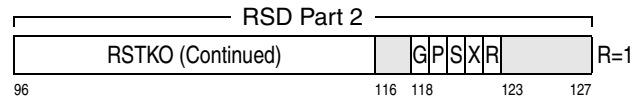
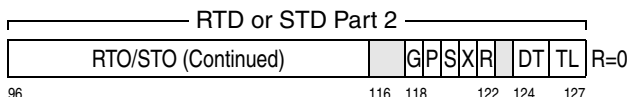
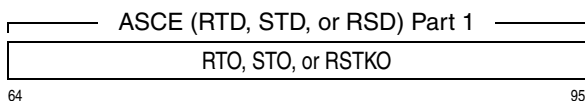
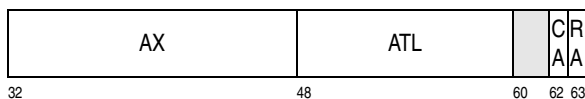
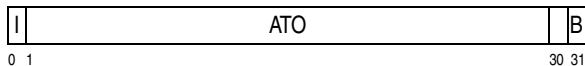
ASTE Sequence Number (ASTESN): Bits 96-127 may be used to revoke the addressing capability represented by the access-list entry. Bits 96-127 are compared against an ASTE sequence number (ASTESN) in the designated ASTE during access-register translation.

Bits 1-5, 32-64, and 90-95 are reserved for possible future extensions and should be zeros.

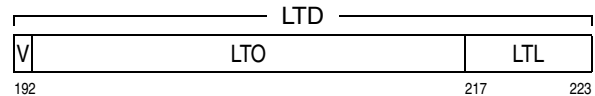
In both the dispatchable-unit access list and the primary-space access list, access-list entries 0 and 1 are intended not to be used in access-register translation. Bits 1-127 of access-list entry 0 and bits 1-63 of access-list entry 1 are reserved for possible future extensions and should be zeros. Bit 0 of access-list entries 0 and 1, and bits 64-127 of access-list entry 1, are available for use by programming. The control program should set bit 0 of access-list entries 0 and 1 to one in order to prevent the use of these entries by means of ALETs in which the ALEN is 0 or 1.

ASN-Second-Table Entries

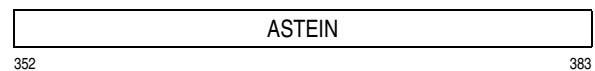
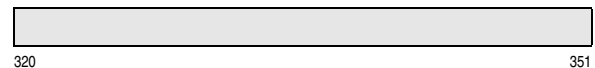
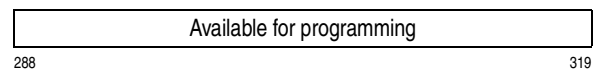
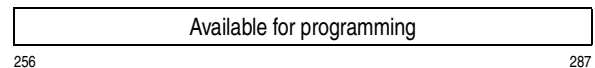
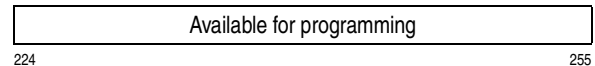
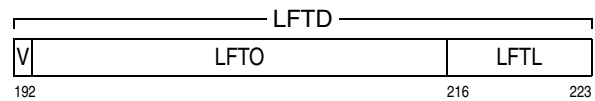
The first 48 bytes of the 64-byte ASN-second-table entry have the following format:



If ASN-and-LX Reuse Is Not Enabled



If ASN-and-LX Reuse Is Enabled



The fields in bytes 0-47 of the ASN-second-table entry (ASTE) are defined with respect to certain mechanisms and instructions in “ASN-Second-Table Entries” on page 3-27. The fields in the ASTE are defined with respect to the BRANCH IN SUBSPACE GROUP instruction in “Subspace-Group ASN-Second-Table Entries” on page 5-64. With respect to access-register translation only, and only for an instruction other than BRANCH IN SUBSPACE GROUP, the fields in the ASTE are allocated as follows:

ASX-Invalid Bit (I): Bit 0 controls whether the address space associated with the ASTE is available. When bit 0 is zero, access-register translation proceeds. When the bit is one, an ASTE-validity exception is recognized.

Authority-Table Origin (ATO): Bits 1-29, with two zeros appended on the right, form a 31-bit address

that designates the beginning of the authority table. This address is treated unpredictably as either a real address or an absolute address, although it is treated as a real address for ASN authorization. The authority table is accessed in access-register translation only if the private bit in the access-list entry is one and the access-list-entry authorization index (ALEAX) in the access-list entry is not equal to the extended authorization index (EAX) in control register 8.

Base-Space Bit (B): Bit 31 is ignored during access-register translation. Bit 31 is further described in “Subspace-Group ASN-Second-Table Entries” on page 5-64.

Authorization Index (AX): Bits 32-47 are not used in access-register translation.

Authority-Table Length (ATL): Bits 48-59 specify the length of the authority table in units of four bytes, thus making the authority table variable in multiples of 16 entries. The length of the authority table, in units of four bytes, is one more than the ATL value. The contents of the ATL field are used to establish whether the entry designated by a particular EAX is within the authority table. An extended-authority exception is recognized if the entry is not within the table.

Controlled-ASN Bit (CA): Bit 62 is not used in access-register translation.

Reusable-ASN Bit (RA): Bit 63 is not used in access-register translation.

Address-Space-Control Element (ASCE): Bits 64-127 are an eight-byte address-space-control element (ASCE) that may be a segment-table designation (STD), a region-table designation (RTD), or a real-space designation (RSD). (The term region-table designation is used to mean a region-first-table designation, region-second-table designation, or region-third-table designation.) The ASCE field is obtained as the result of access-register translation and is used by DAT to translate the logical address for the storage-operand reference being made. Bit 121, the space-switch-event control, is not used in or as a result of access-register translation. The other fields in the ASCE (RTO, STO, RSTKO, G, P, S, R, DT, and TL) are described in “Control Register 1” on page 3-38.

Access-List Designation (ALD): When this ASTE is designated by the primary-ASTE origin in control register 5, bits 128-159 are the primary-space access-list designation (PSALD). See the description of the access-list designation in “Dispatchable-Unit Control Table and Access-List Designations” on page 5-52. During access-register translation when the primary-list bit, bit 7, in the ALET being translated is one, the PSALD is the effective access-list designation.

ASN-Second-Table-Entry Sequence Number (ASTESN): Bits 160-191 are used to control revocation of the accessing capability represented by access-list entries that designate the ASTE. During access-register translation, bits 160-191 are compared against the ASTESN in the access-list entry, and inequality causes an ASTE-sequence exception to be recognized. It is intended that the control program change the value of bits 160-191 when the authorization policies for the address space specified by the ASTE change or when the ASTE is reassigned to specify another address space.

Linkage-Table Designation (LTD) or Linkage-First-Table Designation (LFTD): Bits 192-223 are not used in access-register translation.

ASN-Second-Table-Entry Instance Number (ASTEIN): Bits 352-383 are not used in access-register translation.

Bits 224-319 in the ASTE are available for use by programming.

Programming Note: All unused fields in the ASTE, including the unused fields in bytes 0-31, bytes 40-43, and bytes 48-63, should be set to zeros. These fields are reserved for future extensions, and programs which place nonzero values in these fields may not operate compatibly on future machines.

Access-Register-Translation Process

This section describes the access-register-translation process as it is performed during a storage-operand reference in the access-register mode. The following instructions perform access-register translation as described in this section, but with the exceptions noted below.

- LOAD PAGE-TABLE-ENTRY ADDRESS when the M_4 field is 0001 binary, or when the M_4 field is 0100 binary and bits 16-17 of the PSW are 01 binary
- LOAD REAL ADDRESS and STORE REAL ADDRESS when PSW bits 16 and 17 are 01 binary
- TEST ACCESS in any translation mode
- TEST PROTECTION in the access-register mode

For LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, TEST ACCESS, and TEST PROTECTION, the following exceptions cause a setting of the condition code instead of being treated as program-interruption conditions:

- ALET specification
- ALEN translation
- ALE sequence
- ASTE validity
- ASTE sequence
- Extended authority

BRANCH IN SUBSPACE GROUP performs access-register translation as described in “BRANCH IN SUBSPACE GROUP” on page 10-13.

Access-register translation operates on the access register designated in a storage-operand reference in order to obtain an address-space-control element for use by DAT. When one of access-registers 1-15 is designated, the access-list-entry token (ALET) that is in the access register is used to obtain the address-space-control element. When access register 0 is designated, an ALET having the value 00000000 hex is used, except that TEST ACCESS uses the actual contents of access register 0.

When the ALET is 00000000 or 00000001 hex, the primary or secondary address-space-control element, respectively, is obtained.

When the ALET is other than 00000000 or 00000001 hex, the leftmost seven bits of the ALET are checked for zeros, the primary-list bit in the ALET and the contents of control register 2 or 5 are used to obtain the

effective access-list designation, and the access-list entry number (ALEN) in the ALET is used to select an entry in the effective access list.

The access-list entry is checked for validity and for containing the correct access-list-entry sequence number (ALESN).

The ASN-second-table entry (ASTE) addressed by the access-list entry is checked for validity and for containing the correct ASN-second-table-entry sequence number (ASTESN).

Whether the program is authorized to use the access-list entry is determined through the use of one or more of: (1) the private bit and access-list-entry authorization index (ALEAX) in the access-list entry, (2) the extended authorization index (EAX) in control register 8, and (3) an entry in the authority table addressed by the ASN-second-table entry.

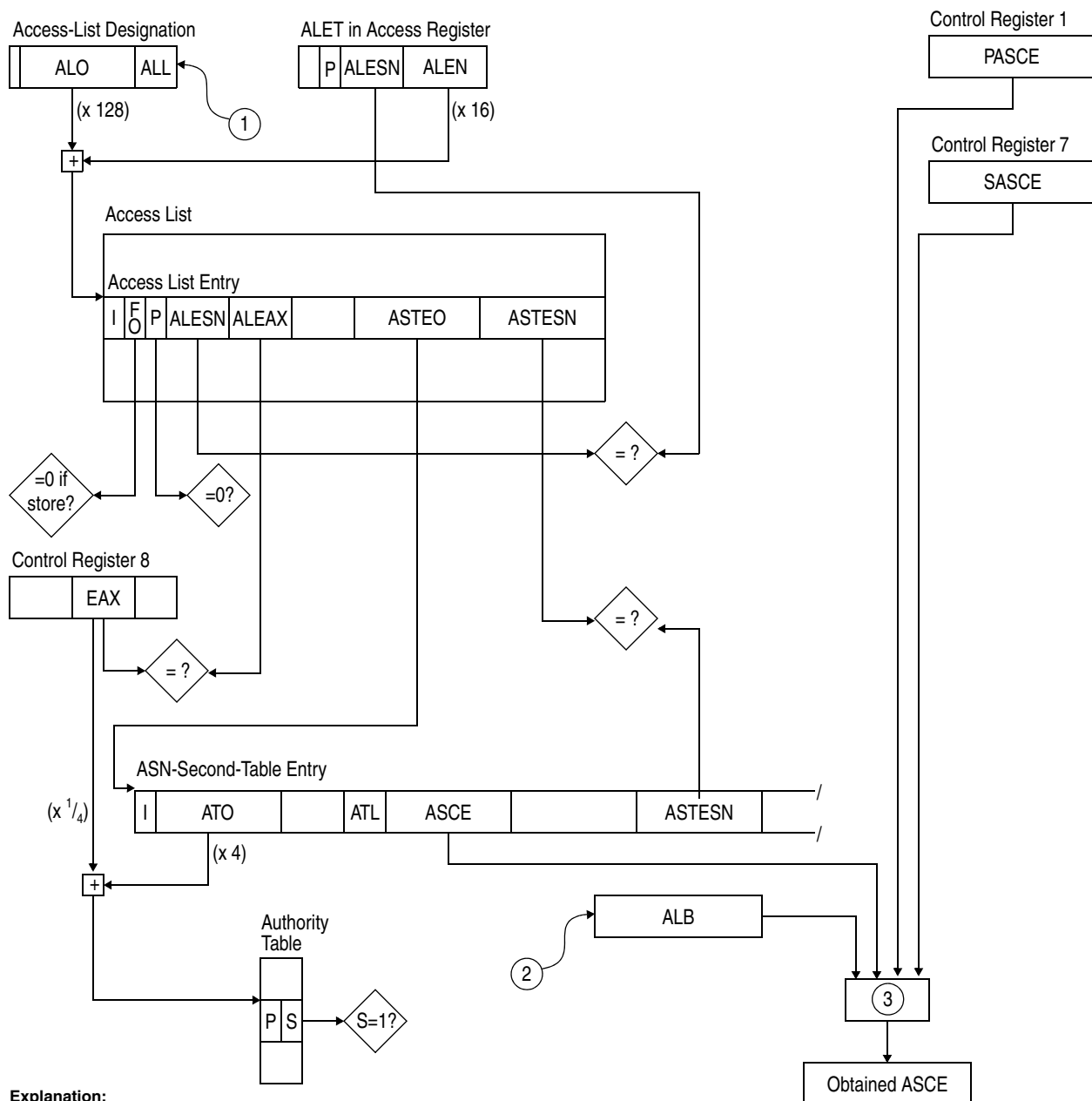
If a store-type reference is to be performed, the fetch-only bit in the access-list entry is checked for being zero.

When no exceptions are recognized, the address-space-control element in the ASN-second-table entry is obtained.

In order to avoid the delay associated with references to real or absolute storage, the information fetched from real or absolute storage normally is also placed in a special buffer, the ART-lookaside buffer (ALB), and subsequent translations involving the same information may be performed by using the contents of the ALB. The operation of the ALB is described in “ART-Lookaside Buffer” on page 5-60.

Whenever access to real or absolute storage is made during access-register translation for the purpose of fetching an entry from an access-list-designation source, access list, ASN second table, or authority table, key-controlled protection does not apply.

The principal features of access-register translation, including the effect of the ALB, are shown in Figure 5-12 on page 5-57.



Explanation:

- 1 The appropriate ALD is obtained:
When P in the ALET is zero (and the ALET is not zero or one), the DUALD in the DUCT is obtained.
When P in the ALET is one, the PSALD in the primary ASTE is obtained.
- 2 Information, which may include the ALD-source origin, ALET, ALO, and EAX, is used to search the ALB. This information, along with information from the ALE, ASTE, and ATE, may be placed in the ALB.
- 3 The appropriate ASCE is obtained:
When the ALET is zero, the PASCE in CR 1 is obtained.
When the ALET is one, the SASCE in CR 7 is obtained.
When the ALET is larger than one:
- If a match exists, the ASCE from the ALB is used.
- If no match exists, tables from real or absolute storage are fetched. The resulting ASCE from the ASTE is obtained, and entries may be formed in the ALB.

Figure 5-12. Access-Register Translation

Selecting the Access-List-Entry Token

When one of access registers 1-15 is designated, or for the access register designated by the R_1 field of TEST ACCESS, access-register translation uses the access-list-entry token (ALET) that is in the access register. When access register 0 is designated, except for TEST ACCESS, an ALET having the value 00000000 hex is used, and the contents of access register 0 are not examined.

Obtaining the Primary or Secondary Address-Space-Control Element

When the ALET being translated is 00000000 hex, the primary address-space-control element in control register 1 is obtained. When the ALET is 00000001 hex, the secondary address-space-control element in control register 7 is obtained. In each of these two cases, access-register translation is completed.

Checking the First Byte of the ALET

When the ALET being translated is other than 00000000 or 00000001 hex, bits 0-6 of the ALET are checked for being all zeros. If bits 0-6 are not all zeros, an ALET-specification exception is recognized, and the operation is suppressed.

Obtaining the Effective Access-List Designation

The primary-list bit, bit 7, in the ALET is used to perform a lookup to obtain the effective access-list designation. When bit 7 is zero, the effective ALD is the dispatchable-unit ALD located in bytes 16-19 of the dispatchable-unit control table (DUCT). When bit 7 is one, the effective ALD is the primary-space ALD located in bytes 16-19 of the primary ASN-second-table entry (primary ASTE).

When bit 7 is zero, the 31-bit real address of the dispatchable-unit ALD is obtained by appending six zeros on the right to the DUCT origin, bits 33-57 of control register 2, and adding 16. The addition cannot cause a carry into bit position 0.

When bit 7 is one, the 31-bit real address of the primary-space ALD is obtained by appending six zeros on the right to the primary-ASTE origin, bits 33-57 of control register 5, and adding 16. The addition cannot cause a carry into bit position 0.

The obtained 31-bit real address is used to fetch the effective ALD — either the dispatchable-unit ALD or the primary-space ALD, depending on bit 7 of the

ALET. The fetch of the effective ALD appears to be word concurrent, as observed by other CPUs, and is not subject to protection. When the storage address that is generated for fetching the effective ALD refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed. When the primary-space ALD is fetched, bit 0, the ASX-invalid bit, in the primary ASTE is ignored.

Access-List Lookup

A lookup in the effective access list is performed. The effective access list is the dispatchable-unit access list if bit 7 of the ALET is zero, or it is the primary-space access list if bit 7 is one. The effective access list is treated unpredictably as being in either real or absolute storage.

The access-list-entry-number (ALEN) portion of the ALET is used to select an entry in the effective access list. The 31-bit real or absolute address of the access-list entry is obtained by appending seven zeros on the right to bits 1-24 of the effective ALD and adding the ALEN, with four rightmost and 11 leftmost zeros appended. When a carry into bit position 0 occurs during the addition, an addressing exception may be recognized, or the carry may be ignored, causing the access list to wrap from $2^{31} - 1$ to 0. The 31-bit address is formed and used regardless of whether the current PSW specifies the 24-bit, 31-bit, or 64-bit addressing mode.

As part of the access-list-lookup process, the leftmost 13 bits of the ALEN are compared against the effective access-list length, bits 25-31 of the effective ALD, to establish whether the addressed entry is within the access list. For this comparison, the access-list length is extended with six leftmost zeros. If the value formed from the access-list length is less than the value in the 13 leftmost bits of the ALEN, an ALEN-translation exception is recognized, and the operation is nullified.

The 16-byte access-list entry is fetched by using the real or absolute address. The fetch of the entry appears to be word concurrent as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining three words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address that is generated for fetching the access-list entry refers to a location which is not available in the configuration, an

addressing exception is recognized, and the operation is suppressed.

Bit 0 of the access-list entry indicates whether the access-list entry specifies an address space by designating an ASN-second-table entry. This bit is inspected, and, if it is one, an ALEN-translation exception is recognized, and the operation is nullified.

When bit 0 is zero, the access-list-entry sequence number (ALESN) in bit positions 8-15 of the access-list entry is compared against the ALESN in the ALET to determine whether the ALET designates the conceptually correct access-list entry. Inequality causes an ALE-sequence exception to be recognized and the operation to be nullified.

Locating the ASN-Second-Table Entry

The ASN-second-table-entry (ASTE) origin in the access-list entry is used to locate the ASTE. Bits 65-89 of the access-list entry, with six zeros appended on the right, form the 31-bit real address of the ASTE.

The 64-byte ASTE is fetched by using the real address. The fetch of the entry appears to be word concurrent as observed by other CPUs, with the left-most word fetched first, except that the fetch of the address-space-control element in the entry appears to be doubleword concurrent as observed by other CPUs. The order in which the remaining words, after the first word, and the address-space-control element are fetched is unpredictable. The fetch access is not subject to protection. When the storage address that is generated for fetching the ASTE refers to a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the ASTE indicates whether the ASTE specifies an address space. This bit is inspected, and, if it is one, an ASTE-validity exception is recognized, and the operation is nullified.

When bit 0 is zero, the ASTE sequence number (ASTESN) in bit positions 160-191 of the ASTE is compared against the ASTESN in bit positions 96-127 of the access-list entry to determine whether the addressing capability represented by the access-list entry has been revoked. Inequality causes an ASTE-sequence exception to be recognized and the operation to be nullified.

Authorizing the Use of the Access-List Entry

The private bit, bit 7, in the access-list entry is used to determine whether the program is authorized to use the access-list entry. The access-list-entry authorization index (ALEAX) in bit positions 16-31 of the access-list entry, the extended authorization index (EAX) in bit positions 32-47 of control register 8, and the authority table designated by the ASTE may also be used.

When the private bit is zero, the program is authorized, and the authorization step of access-register translation is completed.

When the private bit is one but the ALEAX is equal to the EAX, the program is authorized, and the authorization step of access-register translation is completed.

When the private bit is one and the ALEAX is not equal to the EAX, a process called the extended-authorization process is performed. Extended authorization uses the EAX to select an entry in the authority table designated by the ASTE, and it tests the secondary-authority bit in the selected entry for being one. The program is authorized if the tested bit is one.

Extended authorization is the same as the secondary-ASN-authorization process described in "ASN Authorization" on page 3-31, except as follows:

- The authority-table origin is treated as a real or absolute address instead of as a real address.
- The EAX in control register 8 is used instead of the authorization index (AX) in control register 4.
- When the value in bit positions 0-11 of the EAX is greater than the authority-table length (ATL) in the ASTE, an extended-authority exception is recognized instead of a secondary-authority exception. The operation is nullified if the extended-authority exception is recognized.

When the private bit is one, the ALEAX is not equal to the EAX, and the secondary bit in the authority-table entry selected by the EAX is not one, an extended-authority exception is recognized, and the operation is nullified.

Checking for Access-List-Controlled Protection

If a store-type reference is to be performed and the fetch-only bit, bit 6, in the access-list entry is one, a protection exception is recognized, and the operation is suppressed.

Obtaining the Address-Space-Control Element from the ASN-Second-Table Entry

When the ALET being translated is other than 00000000 or 00000001 hex and no exception is recognized in the steps described above, access-register translation obtains the address-space-control element from bit positions 64-127 of the ASTE.

Recognition of Exceptions during Access-Register Translation

The exceptions which can be encountered during the access-register-translation process and their priority are shown in the section “Access Exceptions” in Chapter 6, “Interruptions.”

Programming Note: When updating an access-list entry or ASN-second-table entry, the program should change the entry from invalid to valid (set bit 0 of the entry to zero) as the last step of the updating. This ensures, because the leftmost word is fetched first, that words of a partially updated entry will not be fetched.

ART-Lookaside Buffer

To enhance performance, the access-register-translation (ART) mechanism normally is implemented such that access-list designations and information specified in access lists, ASN second tables, and authority tables are maintained in a special buffer, referred to as the ART-lookaside buffer (ALB). Access-list designations, access-list entries, ASN-second-table entries, and authority-table entries are collectively referred to as ART-table entries. The CPU necessarily refers to an ART-table entry in real or absolute storage only for the initial access to that entry. The information in the entry may be placed in the ALB, and subsequent ART operations may be performed using the information in the ALB. The presence of the ALB affects the ART process to the extent that (1) a modification of an ART-table entry in real or absolute storage does not necessarily have

an immediate effect, if any, on the translation, (2) the comparison against the access-list length in an access-list designation that is in storage and used in a translation may be omitted if an ALB access-list entry is used, and (3) the comparison against the authority-table length in an ASN-second-table entry that is in storage and used in a translation may be omitted if an ALB authority-table entry is used. In a multiple-CPU configuration, each CPU has its own ALB.

Entries within the ALB are not explicitly addressable by the program.

Information is not necessarily retained in the ALB under all conditions for which such retention is possible. Furthermore, information in the ALB may be cleared under conditions additional to those for which clearing is mandatory.

ALB Structure

The description of the logical structure of the ALB covers the implementation by all systems operating as defined by z/Architecture. The ALB entries are considered as being of four types: ALB access-list designations (ALB ALDs), ALB access-list entries (ALB ALEs), ALB ASN-second-table entries (ALB ASTEs), and ALB authority-table entries (ALB ATEs). An ALB entry is considered as containing within it both the information obtained from the ART-table entry in real or absolute storage and the attributes used to fetch the ART-table entry from real or absolute storage.

There is not an indication in an ALB ALD of whether the ALD-source origin used to select the ALD in real storage was the dispatchable-unit-control-table origin or the primary-ASTE origin.

Note: The following sections describe the conditions under which information may be placed in the ALB, the conditions under which information from the ALB may be used for access-register translation, and how changes to the tables affect the ART process.

Formation of ALB Entries

The formation of ALB entries and the effect of any manipulation of the contents of an ART-table entry in real or absolute storage by the program depend on whether the entry is attached to a particular CPU and on whether the entry is valid.

The *attached* state of an ART-table entry denotes that the CPU to which the entry is attached can attempt to use the entry for access-register translation. The ART-table entry may be attached to more than one CPU at a time.

An access-list entry or ASN-second-table entry is *valid* when the invalid bit associated with the entry is zero. Access-list designations and authority-table entries have no invalid bit and are always valid. The primary-space access-list designation is valid regardless of the value of the invalid bit in the primary ASTE.

An ART-table entry may be placed in the ALB whenever the entry is attached and valid.

An access-list designation is attached to a CPU when the designation is within the dispatchable-unit control table designated by the dispatchable-unit-control-table origin in control register 2 or is within the primary ASTE designated by the primary-ASTE origin in control register 5.

An access-list entry is attached to a CPU when the entry is within the access list specified by either an attached access-list designation (ALD) or a usable ALB ALD. A usable ALB ALD is explained in the next section.

An ASN-second-table entry is attached to a CPU when it is designated by the ASTE origin in either an attached and valid access-list entry (ALE) or a usable ALB ALE. A usable ALB ALE is explained in the next section.

An authority-table entry is attached to a CPU when it is within the authority table designated by either an attached and valid ASN-second-table entry (ASTE) or a usable ALB ASTE. A usable ALB ASTE is explained in the next section.

Use of ALB Entries

The *usable* state of an ALB entry denotes that the CPU can attempt to use the ALB entry for access-register translation. A usable ALB entry attaches the next-lower-level table, if any, and may be usable for a particular instance of access-register translation.

An ALB ALD is in the usable state when the ALDSO field in the ALB ALD matches the current dispatchable-unit-control-table origin or the current primary-ASTE origin.

An ALB ALD may be used for a particular instance of access-register translation when either of the following conditions is met:

1. The primary-list bit in the ALET to be translated is zero, and the ALDSO field in the ALB ALD matches the current dispatchable-unit-control-table origin.
2. The primary-list bit in the ALET to be translated is one, and the ALDSO field in the ALB ALD matches the current primary-ASTE origin.

An ALB ALE is in the usable state when the ALO field in the ALB ALE matches the ALO field in an attached ALD or a usable ALB ALD.

An ALB ALE may be used for a particular instance of access-register translation when all of the following conditions are met:

1. The ALET to be translated has a value larger than 1. (If the ALET is 0 or 1, the contents of CR 1 or CR 7 are used.)
2. The ALO field in the ALB ALE matches the ALO field in the ALD or ALB ALD being used in the translation.
3. The ALEN field in the ALB ALE matches the ALEN field in the ALET to be translated.

An ALB ASTE is in the usable state when the ASTEO field in the ALB ASTE matches the ASTEO field in an attached and valid ALE or a usable ALB ALE.

An ALB ASTE may be used for a particular instance of access-register translation when the ASTEO field in the ALB ASTE matches the ASTEO field in the ALE or ALB ALE being used in the translation.

An ALB ATE may be used for a particular instance of access-register translation when both of the following conditions are met:

1. The ATO field in the ALB ATE matches the ATO field in the ASTE or ALB ASTE being used in the translation.
2. The EAX field in the ALB ATE matches the current EAX.

Modification of ART Tables

When an attached but invalid ART-table entry is made valid, or when an unattached but valid ART-

table entry is made attached, and no entry formed from the ART-table entry is already in the ALB, the change takes effect no later than the end of the current instruction.

When an attached and valid ART-table entry is changed, and when, before the ALB is cleared of copies of that entry, an attempt is made to perform ART requiring that entry, unpredictable results may occur, to the following extent. The use of the new value may begin between instructions or during the execution of an instruction, including the instruction that caused the change. Moreover, until the ALB is cleared of copies of the entry, the ALB may contain both the old and the new values, and it is unpredictable whether the old or new value is selected for a particular ART operation. If the old and new values are used as representations of effective space designations, failure to recognize that the effective space designations are the same may occur, with the result that operand overlap may not be recognized. Effective space designations and operand overlap are discussed in “Interlocks within a Single Instruction” on page 5-87.

When LOAD ACCESS MULTIPLE or LOAD CONTROL changes the parameters associated with ART, the values of these parameters at the start of the operation are in effect for the duration of the operation.

All entries are cleared from the ALB by the execution of PURGE ALB, a COMPARE AND SWAP AND PURGE instruction that purges the ALB, and SET PREFIX, and by CPU reset.

Subspace Groups

The subspace-group facility includes the BRANCH IN SUBSPACE GROUP instruction, allocations of fields in the address-space-control element, dispatchable-unit control table, and ASN-second-table entry, and subspace-replacement operations of the PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, SET SECONDARY ASN WITH INSTANCE, and LOAD ADDRESS SPACE PARAMETERS instructions. BRANCH IN SUBSPACE GROUP is introduced in “Subroutine Linkage without the Linkage Stack” on page 5-11 and described in detail in “BRANCH IN SUBSPACE GROUP” on page 10-13.

Subspace-Group Tables

This section describes the use of the dispatchable-unit control table and ASN-second-table entry by the subspace-group facility.

Subspace-Group Dispatchable-Unit Control Table

The dispatchable-unit control table has the following format:

Hex	Dec	
0	0	BASTEO
4	4	SSASTEO
8	8	
C	12	SSASTESN
10	16	DUALD
14	20	PSW-Key Mask PSW Key R A P
18	24	
1C	28	////////////////////////////////////
In the 24-Bit or 31-Bit Addressing Mode		
20	32	
24	36	Bits 33-63 of Return Address
In the 64-Bit Addressing Mode		
20	32	Bits 0-31 of Return Address
24	36	Bits 32-63 of Return Address
28	40	
2C	44	Trap-Control-Block Address E
30	48	
34	52	
38	56	
3C	60	

The fields in the dispatchable-unit control table that are used by the subspace-group facility are allocated as follows:

Base-ASTE Origin (BASTEO): Bits 1-25 of bytes 0-3, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the ASN-second-table entry that specifies the base space of a subspace group associated with the dispatchable unit. A comparison of bits 1-25 of bytes 0-3 to the primary-ASTE origin (PASTEO) in bit positions

33-57 of control register 5 is made by BRANCH IN SUBSPACE GROUP to determine whether the current primary address space is in the subspace group for the current dispatchable unit. For this comparison, either bits 1-25 may be compared to the PASTE0 or the entire contents of bytes 0-3 may be compared to the contents of bit positions 33-63 of control register 5. A comparison of bits 1-25 of bytes 0-3 to the destination-ASTE origin (DASTE0) obtained from an access-list entry by access-register translation of an ALET other than ALETs 0 and 1 is made by BRANCH IN SUBSPACE GROUP to determine if the destination ASTE is the base-space ASTE. For this comparison, either bits 1-25 may be compared to the DASTE0 or the entire contents of bytes 0-3 may be compared to the DASTE0 with one leftmost and six rightmost zeros appended. A comparison of bits 1-25 of bytes 0-3 to an ASTE origin (ASTE0) obtained by ASN translation may be made by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, SET SECONDARY ASN WITH INSTANCE, and LOAD ADDRESS SPACE PARAMETERS. For this comparison, either bits 1-25 may be compared to the ASTE0 or the entire contents of bytes 0-3 may be compared to the ASTE0 with one leftmost and six rightmost zeros appended. When BRANCH IN SUBSPACE GROUP uses ALET 0, bits 1-25 of bytes 0-3, with six zeros appended on the right, designate the destination ASTE.

Subspace-Active Bit (SA): Bit 0 of bytes 4-7 indicates, when one, that the last BRANCH IN SUBSPACE GROUP instruction executed for the dispatchable unit transferred control to a subspace of the subspace group associated with the dispatchable unit. Bit 0 being zero indicates any one of the following: the last BRANCH IN SUBSPACE GROUP instruction executed for the dispatchable unit transferred control to the base space of the subspace group, BRANCH IN SUBSPACE GROUP has not yet been executed for the dispatchable unit, or the dispatchable unit is not associated with a subspace group. BRANCH IN SUBSPACE GROUP sets bit 0 of bytes 4-7 to one when it transfers control to a subspace of the subspace group associated with the dispatchable unit, and it sets bit 0 to zero when it transfers control to the base space of the subspace group.

Subspace-ASTE Origin (SSASTE0): Bits 1-25 of bytes 4-7, with six zeros appended on the right, form a 31-bit real address that designates the beginning of the ASN-second-table entry that specifies the sub-

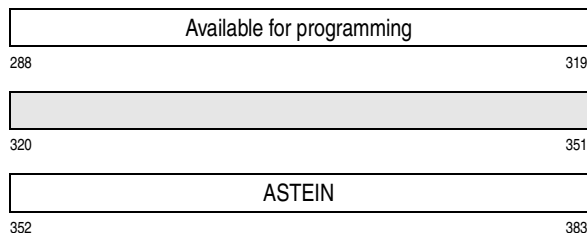
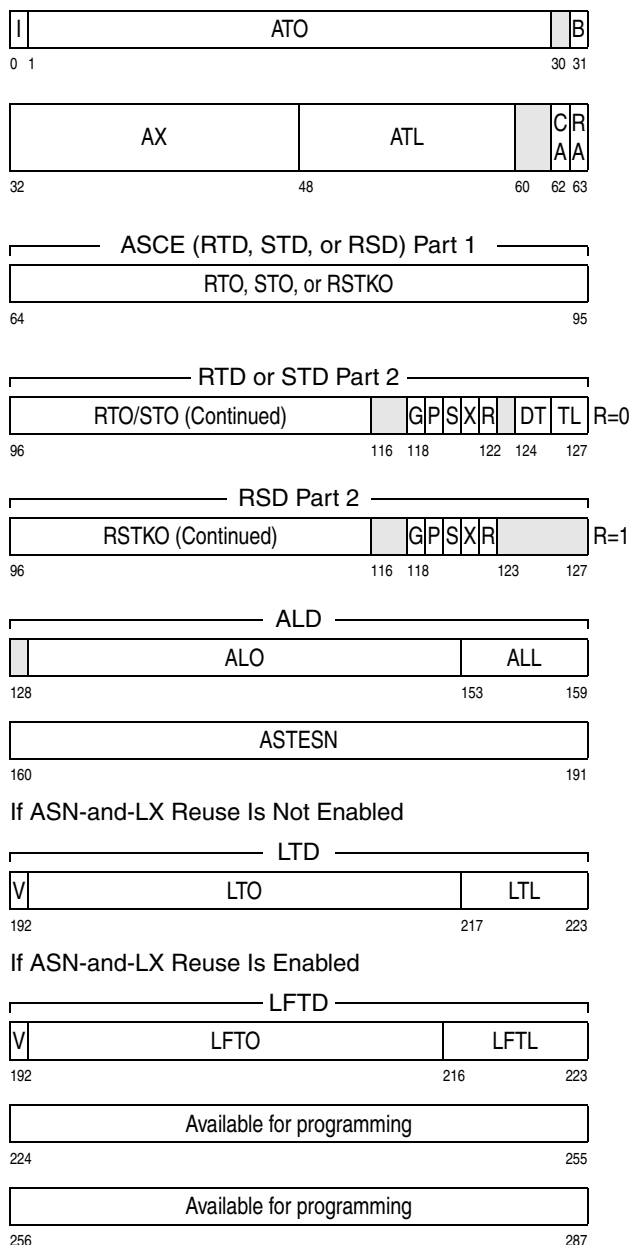
space last given control by a BRANCH IN SUBSPACE GROUP instruction executed for the dispatchable unit. When BRANCH IN SUBSPACE GROUP transfers control to a subspace by means of an ALET other than ALET 1, it places the ASTE0 for the subspace (the destination ASTE0) in bit positions 1-25 of bytes 4-7, places zeros in bit positions 26-31 of bytes 4-7, and sets the subspace-active bit, bit 0 of bytes 4-7, to one. When BRANCH IN SUBSPACE GROUP uses ALET 1 to transfer control to a subspace, bits 1-25 of bytes 4-7, with six zeros appended on the right, designate the destination ASTE, and BRANCH IN SUBSPACE GROUP sets the subspace-active bit to one and either sets bits 26-31 of bytes 4-7 to zeros or leaves those bits unchanged. However, if bits 1-25 are all zeros, a special-operation exception is recognized. When BRANCH IN SUBSPACE GROUP transfers control to the base space of the subspace group, it sets the subspace-active bit to zero, and bits 1-31 of bytes 4-7 remain unchanged. Bits 1-25 of bytes 4-7 may be used by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, SET SECONDARY ASN WITH INSTANCE, and LOAD ADDRESS SPACE PARAMETERS to set bits 0-55 and 57-63 of the primary ASCE in control register 1 or the secondary ASCE in control register 7 from the same bits of the ASCE in the subspace ASTE.

Subspace-ASTE Sequence Number (SS-ASTESN): Bytes 12-15 may be used to revoke the linkage capability represented by the SSASTE0, bits 1-25 of bytes 4-7, in the DUCT. When BRANCH IN SUBSPACE GROUP transfers control to a subspace by means of an ALET other than ALET 1, it obtains the ASTESN in the subspace ASTE and places it in bytes 12-15. When BRANCH IN SUBSPACE GROUP uses ALET 1 to transfer control to a subspace, it compares bytes 12-15 to the ASTESN in the subspace ASTE, and it recognizes an ASTE-sequence exception if they are unequal. When the SSASTE0 is used by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, SET SECONDARY ASN WITH INSTANCE, and LOAD ADDRESS SPACE PARAMETERS to set bits 0-55 and 57-63 of the primary ASCE in control register 1 or the secondary ASCE in control register 7 from the same bits of the ASCE in the subspace ASTE, those instructions first compare bytes 12-15 to the ASTESN in the subspace ASTE, and they recognize an ASTE-sequence exception if the two fields are unequal.

Bytes 16-19 are described in “Dispatchable-Unit Control Table and Access-List Designations” on page 5-52. Bytes 20-23 are described in “BRANCH AND SET AUTHORITY” on page 10-6. Bytes 32-39 and 44-47 are described in “TRAP” on page 10-159. Bytes 24-27, 40-43, and 48-63 are reserved for possible future extensions and should contain all zeros. Bytes 28-31 are available for use by programming.

Subspace-Group ASN-Second-Table Entries

The first 48 bytes of the 64-byte ASN-second-table entry have the following format:



For BRANCH IN SUBSPACE GROUP, the fields in bytes 0-47 of the ASTE are allocated as follows:

ASX-Invalid Bit (I): Bit 0 controls whether the address space associated with the ASTE is available. When bit 0 is zero during access-register translation of ALET 1 or an ALET other than 0 and 1 for BRANCH IN SUBSPACE GROUP, the translation proceeds. When the bit is one, an ASTE-validity exception is recognized. The bit is ignored during access-register translation of ALET 0. When the ASTE is designated by a subspace-ASTE origin (SSASTE0) in a dispatchable-unit control table, bit 0 is also used as described in the definition of bits 160-191 (ASTESN).

Authority-Table Origin (ATO): Bits 1-29 are not used by BRANCH IN SUBSPACE GROUP.

Base-Space Bit (B): Bit 31 specifies, when one, that the address space associated with the ASTE is the base space of a subspace group. When BRANCH IN SUBSPACE GROUP uses an ALET other than ALETs 0 and 1 to locate a destination ASTE, it recognizes a special-operation exception if the destination-ASTE origin does not equal the base-ASTE origin in the dispatchable-unit control table and bit 31 is one in the destination ASTE.

Authorization Index (AX): Bits 32-47 are not used by BRANCH IN SUBSPACE GROUP.

Authority-Table Length (ATL): Bits 48-59 are not used by BRANCH IN SUBSPACE GROUP.

Controlled-ASN Bit (CA): Bit 62 is not used by BRANCH IN SUBSPACE GROUP.

Reusable-ASN Bit (RA): Bit 63 is not used by BRANCH IN SUBSPACE GROUP.

Address-Space-Control Element (ASCE): Bits 64-127 are an eight-byte address-space-control element (ASCE) that may be a segment-table designa-

tion (STD), a region-table designation (RTD), or a real-space designation (RSD). (The term “region-table designation” is used to mean a region-first-table designation, region-second-table designation, or region-third-table designation.) The ASCE field is obtained as the result of access-register translation done for BRANCH IN SUBSPACE GROUP. When BRANCH IN SUBSPACE GROUP uses an ALET other than ALETs 0 and 1 to locate a destination ASTE, it recognizes a special-operation exception if the destination-ASTE origin does not equal the base-ASTE origin in the dispatchable-unit control table and the subspace-group-control bit, bit 118 (G), in the destination ASTE is zero. When BRANCH IN SUBSPACE GROUP transfers control to the base space of a subspace group associated with the current dispatchable unit, it places bits 64-127 in control register 1; otherwise, when BRANCH IN SUBSPACE GROUP transfers control to a subspace of the subspace group, it places bits 64-119 and 121-127 in bit positions 0-55 and 57-63, respectively, of control register 1. Bits 64-127 are used after ASN translation by PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS as described in “ASN-Second-Table Entries” on page 3-27.

Access-List Designation (ALD): When this ASTE is designated by the primary-ASTE origin in control register 5, bits 128-159 are the primary-space access-list designation (PSALD). During access-register translation when the primary-list bit, bit 7, in the ALET being translated is one, the PSALD is the effective access-list designation.

ASN-Second-Table-Entry Sequence Number (ASTESN): Bits 160-191 are used to control revocation of the accessing capability represented by access-list entries that designate the ASTE. During access-register translation, bits 160-191 are compared against the ASTESN in the access-list entry, and inequality causes an ASTE-sequence exception to be recognized.

Linkage-Table Designation (LTD) or Linkage-First-Table Designation (LFTD): Bits 192-223 are not used by BRANCH IN SUBSPACE GROUP.

ASN-Second-Table-Entry Instance Number (ASTEIN): Bits 352-383 are not used by BRANCH IN SUBSPACE GROUP.

Bits 224-319 in the ASTE are available for use by programming.

When the ASTE is designated by a subspace-ASTE origin (SSASTE0) in a dispatchable-unit control table, bits 160-191 are also used to control revocation of the linkage capability represented by that SSASTE0. When BRANCH IN SUBSPACE GROUP uses ALET 1 to transfer control to the subspace specified by the SSASTE0, or when PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, SET SECONDARY ASN, or LOAD ADDRESS SPACE PARAMETERS uses the SSASTE0 to set bits 0-55 and 57-63 of the primary ASCE in control register 1 or the secondary ASCE in control register 7 from the same bits of the ASCE in the subspace ASTE, those instructions first test bit 0 of the subspace ASTE for being zero and recognize an ASTE-validity exception if it is not, and they then compare bits 160-191 to the subspace-ASTE sequence number (SSASTESN) in the dispatchable-unit control table and recognize an ASTE-sequence exception if there is an inequality. However, when either of the two named exception conditions exists for LOAD ADDRESS SPACE PARAMETERS, the instruction sets condition code 1 or 2 instead of recognizing the exception.

Programming Note: All unused fields in the ASTE, including the unused fields in bytes 0-31 and all of bytes 32-63, should be set to zeros. These fields are reserved for future extensions, and programs which place nonzero values in these fields may not operate compatibly on future machines.

Subspace-Replacement Operations

The subspace-group facility includes subspace-replacement operations of PROGRAM CALL, PROGRAM TRANSFER, PROGRAM RETURN, SET SECONDARY ASN, and LOAD ADDRESS SPACE PARAMETERS. The operations apply when the dispatchable unit for which any of the five named instructions is executed is in a state called subspace active. A dispatchable unit is subspace active if it has used BRANCH IN SUBSPACE GROUP to transfer control to a subspace of its subspace group and has not subsequently used BRANCH IN SUBSPACE GROUP to return control to the base space of the group.

The definitions of the subspace-replacement operations are included in the definitions of the five named instructions in Chapter 10, "Control Instructions." The operations are described in a general way as follows. Whenever (1) an address space is established as the primary or secondary address space as a result of ASN translation or (2) PROGRAM CALL obtains the origin of the ASN-second-table entry specifying a new primary address space from the entry-table entry used, then, if that address space is in a subspace group, as indicated by the subspace-group-control bit, bit 54 (G), being one in the address-space-control element (ASCE) for the address space (the new PASCE in control register 1 or SASCE in control register 7), and if the dispatchable unit is subspace-active, as indicated by the subspace-active bit, bit 0 (SA) of word 1, in the dispatchable-unit control table (DUCT) being one, the ASN-second-table-entry (ASTE) origin (ASTE0) for the address space, which was obtained by ASN translation or from the entry-table entry, is compared to the base-ASTE origin (BASTE0), bits 1-25 of word 0, in the DUCT. If that ASTE0 and the BASTE0 are equal, the following occurs. An ASTE-validity exception is recognized if bit 0 in the ASTE for the last subspace entered by the dispatchable unit, which ASTE is designated by the subspace-ASTE origin (SSASTE0) in the DUCT, is one. An ASTE-sequence exception is recognized if the ASTE-sequence number (ASTESN) in word 5 of the subspace ASTE does not equal the subspace ASTESN (SSASTESN) in word 3 of the DUCT. However, LOAD ADDRESS SPACE PARAMETERS sets a nonzero condition code instead of recognizing the ASTE-validity or ASTE-sequence exception. If no exception exists, bits 0-55 and 57-63 of the ASCE for the address space (the PASCE in control register 1 or SASCE in control register 7) are replaced by the same bits of the ASCE in word 2 of the subspace ASTE.

If an addressing exception is recognized when attempting to access the DUCT or subspace ASTE, the instruction execution is suppressed. If an ASTE-validity or ASTE-sequence exception is recognized, the instruction execution is nullified. Such nullification or suppression causes all control register contents to remain unchanged from what they were at the beginning of the instruction execution.

Key-controlled protection does not apply to any accesses to the DUCT or subspace ASTE.

For comparing the ASTE0 obtained by ASN translation to the BASTE0, either the ASTE0 may be com-

pared to the BASTE0 or the ASTE0, with one leftmost and six rightmost zeros appended, may be compared to the entire contents of word 0 of the DUCT.

When the SSASTE0 in the DUCT is used to access the subspace ASTE, no check is made for whether the SSASTE0 is all zeros.

The references to the DUCT and subspace ASTE are single-access references and appear to be word concurrent as observed by other CPUs. The words of the DUCT are accessed in no particular order. The words of the subspace ASTE are accessed in no particular order except that word 0 is accessed first.

The exceptions that can be recognized during a subspace-replacement operation are referred to collectively as the subspace-replacement exceptions and are listed in priority order in "Subspace-Replacement Exceptions" on page 6-49.

Linkage-Stack Introduction

Many of the functions related to the linkage stack are described in this section and in "Linkage-Stack Operations" on page 5-72. Additionally, tracing of the stacking PROGRAM CALL instruction and of the PROGRAM RETURN instruction is described in Chapter 5, "Program Execution"; interruptions in Chapter 6, "Interruptions"; and the instructions are described in Chapter 10, "Control Instructions."

Summary

These major functions are provided:

1. A table-based subroutine-linkage mechanism that provides PSW and control-register status changing and which saves and restores this status and the contents of general registers and access registers through the use of an entry in a linkage stack.
2. A branch-type linkage mechanism that uses the linkage stack.
3. Instructions for placing an additional two words of status in the current linkage-stack entry and

for retrieving all of the status and the general-register and access-register contents that are in the entry.

4. An instruction for determining whether a program is authorized to use a particular access-list-entry token.
5. Aids for program-problem analysis.

In addition, control and authority mechanisms are incorporated to control these functions.

It is intended that a separate linkage stack be associated with and used by each dispatchable unit. The linkage stack for a dispatchable unit resides in the home address space of the dispatchable unit.

It is intended that a dispatchable unit's linkage stack be protected from the dispatchable unit by means of key-controlled protection. Key-controlled protection does not apply to the linkage-stack instructions that place information in or retrieve information from the linkage stack.

The linkage-stack functions are for use by programs considered to be semi privileged, that is, programs which are executed in the problem state but which are authorized to use additional functions. With these authorization controls, a nonhierarchical organization of programs may be established, with each program in a sequence of calling and called programs having a degree of authority that is arbitrarily different from those of programs before or after it in the sequence. The range of functions available to each program, and the ability to transfer control from one program to another, are prescribed in tables that are managed by the control program.

The linkage-stack instructions, which are semiprivileged, are described in Chapter 10, "Control Instructions." They are:

- BRANCH AND STACK
- EXTRACT STACKED REGISTERS
- EXTRACT STACKED STATE
- MODIFY STACKED STATE
- PROGRAM RETURN
- TEST ACCESS

In addition, the PROGRAM CALL instruction optionally forms an entry in the linkage stack. A PROGRAM CALL instruction that operates on the linkage stack is called stacking PROGRAM CALL. Recognition of

PROGRAM CALL as stacking PROGRAM CALL is under the control of a bit in the entry-table entry.

Linkage-Stack Functions

Transferring Program Control

The use of the linkage stack permits programs operating at arbitrarily different levels of authority to be linked directly without the intervention of the control program. The degree of authority of each program in a sequence of calling and called programs may be arbitrarily different, thus allowing a nonhierarchical organization of programs to be established. Modular authorization control can be obtained principally by associating an extended authorization index with each program module. This allows program modules with different authorities to coexist in the same address space. On the other hand, the extended authorization index in effect during the execution of a called program module can be the one that is associated with the calling program module, thus allowing the called module to be executed with different authorities on behalf of different dispatchable units. Options concerning the PSW-key mask and the secondary ASN are other means of associating different authorities with different programs or with the same called program. The authority of each program is prescribed in tables that are managed by the control program. By setting up the tables so that the same program can be called by means of different PC numbers, the program can be assigned different authorities depending on which PC number is used to call it. The tables also allow control over which PC numbers can be used by a program to call other programs.

The stacking PROGRAM CALL and PROGRAM RETURN linkage operations can link programs residing in different address spaces and having different levels of authority. The execution state and the contents of the general registers and access registers are saved during the execution of stacking PROGRAM CALL and are partially restored during the execution of PROGRAM RETURN. A linkage stack provides an efficient means of saving and restoring both the execution state and the contents of registers during linkage operations.

During the execution of a PROGRAM CALL instruction, the PC-number-translation process is performed to locate a 32-byte entry-table entry. When the PC-type bit in the entry-table entry is one, the stacking PROGRAM CALL operation is specified; otherwise, the basic PROGRAM CALL operation is specified.

In addition to the information applying to both basic PROGRAM CALL and stacking PROGRAM CALL (described in “PC-Number Translation” on page 5-30 and consisting of an authorization key mask and specifications of the new ASN, addressing mode, instruction address, problem state, PSW-key mask, primary-ASTE address, and entry parameter), the entry-table entry contains information that specifies options concerning the address-space control and PSW key in the PSW, and the PSW-key mask, extended authorization index, and secondary ASN in the control registers.

During the stacking PROGRAM CALL operation and by means of the additional information in the entry-table entry, the address-space control in the PSW can be set to specify either the primary-space mode or the access-register mode. The PSW key can be either left unchanged or replaced from the entry-table entry. The PSW-key mask in control register 3 can be either ORed to or replaced from the entry-table entry. The extended authorization index in control register 8 can be either left unchanged or replaced from the entry-table entry. The secondary ASN in control register 3 can be set equal to the primary ASN of either the calling program or the called program; thus, the ability of the called program to have access to the primary address space of the calling program can be controlled.

The stacking PROGRAM CALL operation always forms an entry, called a state entry, in the linkage stack to save the execution state and the contents of general registers 0-15 and access registers 0-15. The saved execution state includes a called-space identification, the numeric part of the PC number used, the updated PSW before any changes are made due to the entry-table entry, the extended authorization index, PSW-key mask, primary ASN, and secondary ASN existing before the operation, and the extended-addressing-mode bit existing after the operation. However, the value of the PER mask in the saved updated PSW is unpredictable. The linkage-stack state entry also contains an entry-type code that identifies the entry as one that was formed by PROGRAM CALL. If the ASN-and-LX-reuse facility is installed and enabled, the state entry also contains the primary ASN-second-table-entry instance number (primary ASTEIN) and secondary ASTEIN existing before the operation.

A space-switching operation occurs when the address-space number (ASN) specified in the entry-table entry is nonzero. When space switching occurs,

the operation is called PROGRAM CALL with space switching (PC-ss), and the ASN in the entry-table entry is placed in control register 4 as a new primary ASN. When no space switching occurs, the operation is called PROGRAM CALL to current primary (PC-cp), and there is no change to the primary ASN in control register 4.

PROGRAM CALL with space switching obtains the new ASN from the entry-table entry and places it in control register 4 as the primary ASN. It obtains a new primary-ASTE origin from the entry-table entry and new primary address-space-control element from the new primary ASTE, and it places them in control registers 5 and 1, respectively. It sets the secondary address-space-control element in control register 7 equal to either the old primary address-space-control element, or the new one, depending on whether it sets the secondary ASN in control register 3 equal to the old primary ASN or the new one, respectively. PROGRAM CALL to current primary sets the secondary ASN equal to the primary ASN and the secondary address-space-control element equal to the primary address-space-control element.

If the ASN-and-LX-reuse facility is installed and enabled, PROGRAM CALL with space switching obtains the ASN-second-table-entry instance number (ASTEIN) from the new primary ASTE and places it in control register 4 as the new primary ASTEIN. It sets the secondary ASTEIN in control register 3 equal to either the old primary ASTEIN or the new one, depending on whether it set the secondary ASN equal to the old primary ASN or the new one, respectively. PROGRAM CALL to current primary sets the secondary ASTEIN equal to the primary ASTEIN.

The instruction PROGRAM RETURN restores most of the information saved in the linkage stack by the stacking PROGRAM CALL operation. It restores the PSW, extended authorization index, PSW-key mask, primary ASN, secondary ASN, and the contents of general registers 2-14 and access-registers 2-14. However, the PER mask in the current PSW remains unchanged. If the ASN-and-LX-reuse facility is installed and enabled, PROGRAM RETURN restores the primary ASTEIN and secondary ASTEIN. The operation of PROGRAM RETURN is referred to by saying that PROGRAM RETURN unstacks a state entry.

For PROGRAM RETURN, a space-switching operation occurs when the restored primary ASN is not equal to the primary ASN existing before the opera-

tion. When space switching occurs, the operation is called PROGRAM RETURN with space switching (PR-ss). When no space switching occurs, the operation is called PROGRAM RETURN to current primary (PR-cp).

PROGRAM RETURN with space switching performs ASN translation of the restored primary ASN to obtain a new primary-ASTE origin and a new primary address-space-control element, which it places in control registers 5 and 1, respectively. For PROGRAM RETURN with space switching or to current primary, (1) if the restored secondary ASN is the same as the restored primary ASN, the secondary address-space-control element in control register 7 is set equal to the new primary address-space-control element in control register 1, or (2) if the restored secondary ASN is not the same as the restored primary ASN, ASN translation and ASN authorization of the restored secondary ASN are performed to obtain a new secondary address-space-control element, which is placed in control register 7.

If the ASN-and-LX-reuse facility is installed and enabled, PROGRAM RETURN with space switching requires that the ASN-second-table-entry instance number (ASTEIN) in the new primary ASTE be equal to the primary ASTEIN saved in the state entry being unstacked. PROGRAM RETURN with space switching or to current primary, if the restored secondary ASN is not the same as the restored primary ASN, requires that the ASTEIN in the new secondary ASTE be equal to the saved ASTEIN. An exception is recognized if either of these requirements is not met.

The stacking PROGRAM CALL operation and the PROGRAM RETURN operation each can be performed successfully only in the primary-space mode or access-register mode. An exception is recognized when the CPU is in the real mode, secondary-space mode, or home-space mode.

A bit, named the unstack-suppression bit, can be set to one in a linkage-stack state entry to cause an exception if an attempt is made by PROGRAM RETURN to unstack the entry. When the bit is one, the entry still can be operated on by the instructions that add information to or retrieve information from the entry. The unstack-suppression bit is intended to allow the control program to gain control when an attempt is made to unstack a state entry in which the bit is one.

Branching Using the Linkage Stack

The execution state and the contents of the general registers and access registers can also be saved in the linkage stack by means of the instruction BRANCH AND STACK. BRANCH AND STACK uses a branch address as do the other branching instructions, instead of using a PC number. BRANCH AND STACK, along with PROGRAM RETURN, can link programs residing in the same address space and having the same level of authority; that is, BRANCH AND STACK does not change the execution state except for the instruction address.

BRANCH AND STACK forms a linkage-stack state entry that is almost the same as one formed by PROGRAM CALL. When it is necessary to distinguish between these two types of state entry, an entry formed by PROGRAM CALL is called a program-call state entry, and one formed by BRANCH AND STACK is called a branch state entry. A branch state entry differs from a program-call state entry in two ways: (1) it contains a different entry-type code, which identifies it as a branch state entry, and (2) it contains the basic-addressing-mode bit and instruction address existing after the operation instead of a called-space identification and the numeric part of the PC number used. These new values of PSW bits 32 and 64-127 are in addition to the complete PSW that is saved in the state entry.

For BRANCH AND STACK, the basic- and extended addressing mode bits and the instruction address that are part of the complete PSW saved in the state entry can be the current (at the beginning of the operation) addressing-mode bits and the updated instruction address (the address of the next sequential instruction), or they can be specified in a register. This register can be one that had link information placed in it by a BRANCH AND LINK (BALR only), BRANCH AND SAVE, BRANCH AND SAVE AND SET MODE, or BRANCH AND SET MODE instruction. Thus, BRANCH AND STACK can be used either in a calling program or at (or near) the entry point of a called program, and, in either case, a PROGRAM RETURN instruction located at the end of the called program will return correctly to the calling program. The ability to use BRANCH AND STACK at an entry point allows the linkage stack to be used without changing old calling programs.

When the R_2 field of BRANCH AND STACK is zero, the instruction is executed without causing branching.

When PROGRAM RETURN unstacks a branch state entry, it ignores the extended authorization index, PSW-key mask, primary ASN, secondary ASN, primary ASTEIN, and secondary ASTEIN in the entry. The PROGRAM RETURN instruction restores the PSW and the contents of general registers 2-14 and access registers 2-14 that were saved in the entry. However, the PER mask in the current PSW remains unchanged.

BRANCH AND STACK can be executed successfully only in the primary-space mode or access-register mode. An exception is recognized when the CPU is in the real mode, secondary-space mode, or home-space mode. The unstack-suppression bit has the same effect in a branch state entry as it does in a program-call state entry.

Adding and Retrieving Information

The instruction MODIFY STACKED STATE can be used by a program to place two words of information, contained in a designated general-register pair, in an area, called the modifiable area, of the current linkage-stack state entry (a branch state entry or a program-call state entry). This is intended to allow a called program to establish a recovery routine that will be given control by the control program, if necessary.

The instructions EXTRACT STACKED REGISTERS and EXTRACT STACKED STATE can be used by a program to obtain any of the information saved in the current state entry by BRANCH AND STACK or PROGRAM CALL or placed there by MODIFY STACKED STATE. EXTRACT STACKED REGISTERS (EREGG) places the contents of a specified range of general registers and access registers back in the registers from which the contents were saved. EXTRACT STACKED REGISTERS (EREG) does the same except that it restores only bits 32-63 of the general registers and leaves bits 0-31 unchanged. EXTRACT STACKED STATE obtains pairs of words of the nonregister information saved or placed in a state entry and places them in bit positions 32-63 of a designated general-register pair. Alternatively, EXTRACT STACKED STATE obtains two double-words containing a PSW saved in the state entry and places them in bit positions 0-63 of a designated general-register pair. If the ASN-and-LX-reuse facility is installed, EXTRACT STACKED STATE can place the contents of two words of the state entry in bit positions 0-31 of a designated general-register pair. These contents are the saved secondary ASTEIN

and primary ASTEIN if the ASN-and-LX-reuse facility is enabled. EXTRACT STACKED STATE sets the condition code to indicate whether the current state entry is a branch state entry or a program-call state entry.

Testing Authorization

The instruction TEST ACCESS has as operands an access-list-entry token (ALET) in a designated access register and an extended authorization index (EAX) in a designated general register. TEST ACCESS applies the access-register-translation process, which uses the specified EAX instead of the current EAX in control register 8, to the ALET, and it sets the condition code to indicate the result. The condition code may indicate: (1) the ALET is 00000000 hex, (2) the ALET designates an entry in the dispatchable-unit access list and can be translated without exceptions in access-register translation, (3) the ALET designates an entry in the primary-space access list and can be translated without exceptions in access-register translation, or (4) the ALET is 00000001 hex or causes exceptions in access-register translation.

The principal purpose of TEST ACCESS is to allow a called program to determine whether an ALET passed to it by the calling program is authorized for use by the calling program by means of the calling program's EAX. This is in support of a possible programming convention in which a called program will not operate on an AR-specified address space by means of its own EAX unless the calling program is authorized to operate on that space by means of the calling program's EAX. The called program can obtain the calling program's EAX, for use by TEST ACCESS, from the current linkage-stack state entry by means of the EXTRACT STACKED STATE instruction.

Another purpose of TEST ACCESS is to indicate the special cases in which the ALET is 00000000 hex, designating the primary address space, or 00000001 hex, designating the secondary address space. Because PROGRAM CALL may change the primary and secondary address spaces, ALETs 00000000 hex and 00000001 hex may designate different address spaces when used by the called program than when used by the calling program.

Still another purpose of TEST ACCESS is to indicate whether the ALET designates an entry in the primary-space access list since such a designation

after the primary address space was changed by a space-switching program-linkage operation may be an error.

Program-Problem Analysis

To aid program-problem analysis, the option is provided of having a trace entry made implicitly for three additional linkage operations when the linkage stack is used. When branch tracing is on, a trace entry is made each time a BRANCH AND STACK instruction is executed and causes branching. When ASN tracing is on, a trace entry is made each time the stacking PROGRAM CALL operation is performed and each time PROGRAM RETURN unstacks a linkage-stack state entry formed by PROGRAM CALL. When mode tracing is on, a trace entry is made each time the stacking PROGRAM CALL operation or PROGRAM RETURN operation is performed and changes PSW bit 31, except that, for PROGRAM RETURN, a trace entry for mode tracing is not made if one due to ASN tracing is made. A detailed definition of tracing is contained in “Tracing” on page 4-10.

As a further analysis aid, BRANCH AND STACK when it causes branching, stacking PROGRAM CALL, and PROGRAM RETURN are also recognized as PER successful-branching events. For PROGRAM RETURN, the unstacked state entry may have been formed by BRANCH AND STACK or PROGRAM CALL.

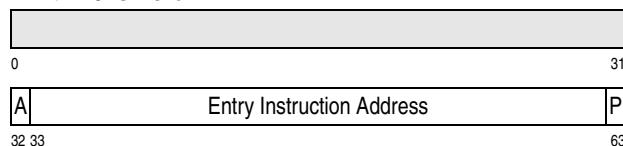
The execution of a space-switching stacking PROGRAM CALL or PROGRAM RETURN instruction causes a space-switch event if the primary space-switch-event control is one before or after the operation or if a PER event is to be indicated.

Linkage-Stack Entry-Table Entries

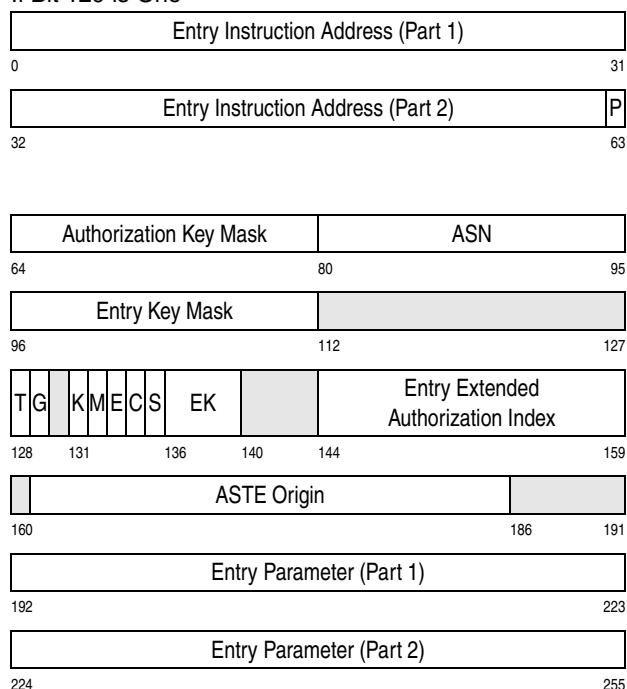
All of the fields in the entry-table entry except bits 130-159 are described in “Entry-Table Entries” on page 5-34. This section describes only bits 130-159.

The entry-table entry has the following format:

If Bit 129 is Zero



If Bit 129 is One



The fields in bit positions 130-159 are allocated as follows:

PSW-Key Control (K): Bit 131, when one, specifies that bits 136-139 are to replace the PSW key in the PSW as part of the stacking PROGRAM CALL operation. When this bit is zero, the PSW key remains unchanged. Bit 131 is ignored during the basic PROGRAM CALL operation.

PSW-Key-Mask Control (M): Bit 132, when one, specifies that bits 96-111 are to replace the PSW-key mask in control register 3 as part of the stacking PROGRAM CALL operation. When this bit is zero, bits 96-111 are ORed into the PSW-key mask in control register 3 as part of the stacking PROGRAM CALL operation. Bit 132 is ignored during the basic PROGRAM CALL operation.

Extended-Authorization-Index Control (E): Bit 133, when one, specifies that bits 144-159 are to replace the current extended authorization index in control register 8 as part of the stacking PROGRAM CALL operation. When this bit is zero, the current extended authorization index remains unchanged. Bit 133 is ignored during the basic PROGRAM CALL operation.

Address-Space-Control Control (C): Bit 134, when one, specifies that bit 17 of the current PSW is

to be set to one as part of the stacking PROGRAM CALL operation. When this bit is zero, bit 17 is set to zero. Because the CPU must be in either the primary-space mode or the access-register mode when a stacking PROGRAM CALL instruction is issued, the result is that the CPU is placed in the access-register mode if bit 134 is one or the primary-space mode if bit 134 is zero. Bit 134 is ignored during the basic PROGRAM CALL operation.

Secondary-ASN Control (S): Bit 135, when one, specifies that bits 80-95 are to become the new secondary ASN, and the new SASCE is to be set equal to the new PASCE, as part of the stacking PROGRAM CALL with-space-switching (PC-ss) operation. When this bit is zero, the new SASN and SASCE are set equal to the PASN and PASCE, respectively, of the calling program. When the ASN-and-LX-reuse facility is installed and enabled, bit 135 similarly specifies, when one, that the new SASTEIN is to be set equal to the new PASTEIN or, when zero, that the new SASTEIN is to be set equal to the PASTEIN of the calling program. Bit 135 is ignored during the basic PROGRAM CALL operation and the stacking PROGRAM CALL to-current-primary (PC-cp) operation.

Entry Key (EK): Bits 136-139 replace the PSW key in the PSW as part of the stacking PROGRAM CALL operation if the PSW-key control, bit 131, is one. Bits 136-139 are ignored, and the current PSW key remains unchanged, if bit 131 is zero. Bits 136-139 are ignored during the basic PROGRAM CALL operation.

Entry Extended Authorization Index: Bits 144-159 replace the current extended authorization index, bits 32-47 of control register 8, as part of the stacking PROGRAM CALL operation if the extended-authorization-index control, bit 133, is one. Bits 144-159 are ignored, and the current extended authorization index remains unchanged, if bit 133 is zero. Bits 144-159 are ignored during the basic PROGRAM CALL operation.

Bits 130 and 140-143 are reserved for possible future extensions and should be zeros.

Linkage-Stack Operations

A linkage stack may be formed by the control program for each dispatchable unit. The linkage stack is

used to save the execution state and the contents of the general registers and access registers during the BRANCH AND STACK and stacking PROGRAM CALL operations. The linkage stack is also used to restore a portion of the execution state and general-register and access-register contents during the PROGRAM RETURN operation.

A linkage stack resides in virtual storage. The linkage stack for a dispatchable unit is in the home address space for that dispatchable unit. The home address space is designated by the home address-space-control element in control register 13.

The linkage stack is intended to be protected from problem-state programs so that these programs cannot examine or modify the information saved in the linkage stack, except by means of the EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE instructions. This protection can be obtained by means of key-controlled protection.

A linkage stack may consist of a number of linkage-stack sections chained together. A linkage-stack section is variable in length. The maximum length of each linkage-stack section is 65,560 bytes.

There are three types of entry in the linkage stack: header entry, trailer entry, and state entry. A header entry and a trailer entry are at the beginning and end, respectively, of a linkage-stack section, and they are used to chain linkage-stack sections together. Header entries and trailer entries are formed by the control program. A state entry is used to contain the execution state and register contents that are saved during the BRANCH AND STACK or stacking PROGRAM CALL operation, and it is formed during the operation. A state entry is further distinguished as being a branch state entry if it was formed by BRANCH AND STACK or as being a program-call state entry if it was formed by PROGRAM CALL.

The actions of forming a state entry and saving information in it during the BRANCH AND STACK and stacking PROGRAM CALL operations are called the stacking process. The actions of restoring information from a state entry and logically deleting the entry during the PROGRAM RETURN operation are called the unstacking process. The part of the unstacking process that locates a state entry is also performed during the EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE operations.

Each type of linkage-stack entry has a length that is a multiple of eight bytes. A header entry and trailer entry each has a length of 16 bytes. A state entry has a length of 296 bytes.

Each of the header entry, trailer entry, and state entry has a common eight-byte area at its end, called the entry descriptor. The linkage-stack-entry address in control register 15 designates the leftmost byte of the entry descriptor of the last linkage-stack entry, other than the trailer entry, in a linkage-stack section. This entry is called the current linkage-stack entry, and the section is called the current linkage-stack section.

Each entry descriptor in a linkage-stack section, except the one in the trailer entry of the section, includes a field that specifies the amount of space existing between the end of the entry descriptor and the beginning of the trailer entry. This field is named the remaining-free-space field. The remaining-free-space field in a trailer entry is unused.

When a new state entry is to be formed in the linkage stack during the stacking process, the new entry is placed immediately after the entry descriptor of the current linkage-stack entry, provided that there is enough remaining free space in the current linkage-stack section to contain the new entry. If there is not enough remaining free space in the current section, and if the trailer entry in the current section indicates that another section follows the current section, the new entry is placed immediately after the entry descriptor of the header entry of that following section, provided that there is enough remaining free space in that section. If the trailer entry indicates that there is not a following section, an exception is recognized, and a program interruption occurs. It is then the responsibility of the control program to allocate another section, chain it to the current section, and cause the **BRANCH AND STACK** or stacking **PROGRAM CALL** instruction to be reexecuted. If there is a following section but there is not enough remaining free space in it, an exception is recognized.

If the remaining-free-space value that is used to locate a trailer entry is not a multiple of 8, an exception is recognized. The remaining-free-space value in the header entry of a linkage-stack section must be set to a multiple of 8 to ensure that the remaining-free-space value that may be used to locate the trailer entry of the section will be a multiple of 8.

When the stacking process is successful in forming a new state entry, it updates the linkage-stack-entry

address in control register 15 so that the address designates the leftmost byte of the entry descriptor of the new entry, which thus becomes the new current linkage-stack entry.

When, during the unstacking process in **PROGRAM RETURN**, the current linkage-stack entry is a state entry, the process operates on that entry and then updates the linkage-stack-entry address so that it designates the entry descriptor of the preceding entry in the same linkage-stack section. The preceding entry thus becomes the current entry. The new current entry may be another state entry, or it may be a header entry.

The header entry of a linkage-stack section indicates whether there is a preceding section. If there is a preceding section, the header entry contains the address of the last linkage-stack entry, other than the trailer entry, in the preceding section. That last entry should be a state entry (not another header entry), unless there is an error in the linkage stack.

If the unstacking process is performed when the current linkage-stack entry is a header entry, and if the header entry indicates that a preceding linkage-stack section exists, the unstacking process proceeds by treating the entry designated in the preceding section as if it were the current entry, provided that this entry is a state entry. If the header entry does not indicate a preceding section, or if the entry designated in the preceding section is not a state entry, an exception is recognized.

When the unstacking process is performed in **EXTRACT STACKED REGISTERS**, **EXTRACT STACKED STATE**, or **MODIFY STACKED STATE**, the process locates a state entry but does not change the linkage-stack-entry address in control register 15.

Each entry descriptor in a linkage-stack section includes a field that specifies the length of the next linkage-stack entry, other than the trailer entry, in the section. When a state entry is created during the stacking process, zeros are placed in this field in the created entry, and the length of the state entry is placed in this field in the preceding entry. When a state entry is logically deleted during the unstacking process in **PROGRAM RETURN**, zeros are placed in this field in the preceding entry. This field is named the next-entry-size field.

When the stacking or unstacking process operates on the linkage stack, key-controlled protection does

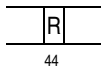
not apply, but low-address and DAT protection do apply.

Linkage-Stack-Operations Control

The use of the linkage stack is controlled by the ASN-and-LX-reuse control in control register 0, the home address-space-control element in control register 13 and the linkage-stack-entry address in control register 15. The home address-space-control element is described in “Dynamic Address Translation” on page 3-34.

The ASN-and-LX-reuse control and the linkage-stack-entry address are

Control Register 0



ASN-and-LX-Reuse Control (R): Bit 44 of control register 0 is the ASN-and-LX-reuse-control bit (R) and is assigned if the ASN-and-LX-reuse facility is installed. The effects of a one value of the bit specifically on the linkage stack are as follows:

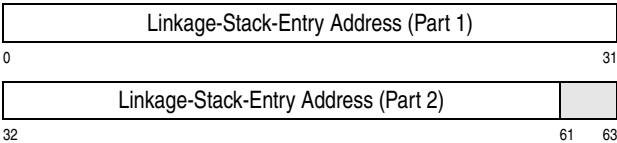
- BRANCH AND STACK and stacking PROGRAM CALL save the secondary ASN-second-table-entry instance number (ASTEIN), bits 0-31 of control register 3, and the primary ASTEIN, bits 0-31 of control register 4, in bytes 176-179 and 180-183, respectively, of the linkage-stack state entry that either of the instructions forms.
- PROGRAM RETURN with space switching, after it has used the PASN in bytes 134 and 135 of the state entry to locate an ASN-second-table entry, compares the primary ASTEIN in bytes 180-183 of the state entry to the ASTEIN in the ASN-second-table entry. PROGRAM RETURN to current primary or with space switching, if it uses the SASN in bytes 130 and 131 of the state entry to locate an ASN-second-table entry (because the new SASN is not equal to the new PASN), compares the secondary ASTEIN in bytes 176-179 of the state entry to the ASTEIN in the ASN-second-table entry. The one or two comparisons of ASTEINs must each give equal results; oth-

erwise, an ASTE-instance exception is recognized. The comparisons occur regardless of the reusable-ASN bits in the ASN-second-table entries. PROGRAM RETURN to current primary or with space switching restores the secondary and primary ASTEINs saved in the state entry to bit positions 0-31 of control registers 3 and 4, respectively (if no exception is recognized). When unstacking a state entry formed by BRANCH AND STACK, PROGRAM RETURN does not compare or restore ASTEINs.

The effect of the ASN-and-LX-reuse control on PC-number translation is described in “ASN-and-LX-Reuse Control (R):” on page 5-31. Other effects of the control are described in “ASN-Second-Table-Entry Instance Number and ASN Reuse” on page 3-21.

Control Register 15

The location of the entry descriptor of the current linkage-stack entry is specified in control register 15. The register has the following format:



Linkage-Stack-Entry Address: Bits 0-60 of control register 15, with three zeros appended on the right, form the 64-bit home virtual address of the entry descriptor of the current linkage-stack entry in the current linkage-stack section. Bits 0-60 are changed during the stacking process in BRANCH AND STACK and stacking PROGRAM CALL and during the unstacking process in PROGRAM RETURN. Bits 61-63 of control register 15 are set to zeros when bits 0-60 are changed.

Linkage Stack

The linkage stack consists of one or more linkage-stack sections containing linkage-stack entries. There are three principal types of linkage-stack entry: header entry, trailer entry, and state entry. A state entry is further distinguished as being either a branch state entry or a program-call state entry.

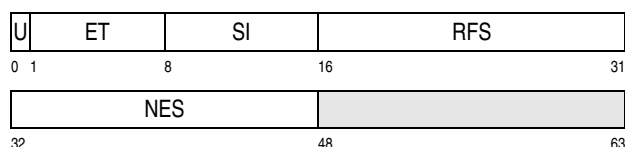
Each type of linkage-stack entry has an entry descriptor at its end. The leftmost byte of the entry

descriptor of the current linkage-stack entry in the current linkage-stack section is designated by the linkage-stack-entry address in control register 15.

The linkage stack resides in the home address space, designated by the home address-space-control element in control register 13.

Entry Descriptors

An entry descriptor is at the end of each linkage-stack entry. The entry descriptor is eight bytes in length and has the following format:



The fields in the entry descriptor are allocated as follows:

Unstack-Suppression Bit (U): When bit 0 is one in the entry descriptor of a header entry or state entry encountered during the unstacking process in PROGRAM RETURN, a stack-operation exception is recognized. Bit 0 is ignored in a trailer entry and during the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE. The control program can temporarily set bit 0 to one in the current linkage-stack entry (a header entry or state entry) to prevent PROGRAM RETURN from being executed successfully while still allowing EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE to be executed successfully. Bit 0 is set to zero in the entry descriptor of a state entry when the entry is formed during the stacking process.

Entry Type (ET): Bits 1-7 are a code that specifies the type of the linkage-stack entry containing the entry descriptor. The assigned codes are:

Code (in Binary)	Entry Type
0001001	Header entry
0001010	Trailer entry
0001100	Branch State entry
0001101	Program-call state entry

Codes 0000000-0001000, 0001011, and 0001110 through 0111111 binary are reserved for possible future assignments. Codes 1000000 through

1111111 binary are available for use by programming.

Bits 1-7 are set to 0001100 or 0001101 binary in the entry descriptor of a state entry when the entry is formed during the stacking process.

A stack-type exception is recognized during the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN if bits 1-7 in the current linkage-stack entry do not indicate that the entry is a state entry or a header entry; or, when the current entry is a header entry, if bits 1-7 in the entry designated by the backward stack-entry address in the header entry do not indicate that the designated entry is a state entry. However, a stack-specification exception is recognized, instead of a stack-type exception, if both the current entry and the designated entry are header entries.

Section Identification (SI): Bits 8-15 are an identification, provided by the control program, of the linkage-stack section containing the entry descriptor. In the state entry formed by a stacking process, the process sets bits 8-15 equal to the contents of the section-identification field in the preceding linkage-stack entry.

Remaining Free Space (RFS): Bits 16-31 specify the number of bytes between the end of this entry descriptor and the beginning of the trailer entry in the same linkage-stack section, except that this field in a trailer entry has no meaning. Thus, in the last state entry in a section, or in the header entry if there is no state entry, bits 16-31 specify the number of bytes available in the section for performance of the stacking process. In the state entry formed by a stacking process, the process sets bits 16-31 equal to the contents of the remaining-free-space field in the preceding linkage-stack entry minus the size, in bytes, of the new entry. Bits 16-31 must be a multiple of 8 (bits 29-31 must be zeros) in the entry descriptor of the header entry in a linkage-stack section; otherwise, a value that is not a multiple of 8 will be propagated to bits 16-31 in the entry descriptor of each state entry in the section, and a stack-specification exception will be recognized if the stacking process attempts to locate the trailer entry in the section in order to proceed to the next section.

Next-Entry Size (NES): Bits 32-47 specify the size in bytes of the next linkage-stack entry, other than a trailer entry, in the same linkage-stack section. This

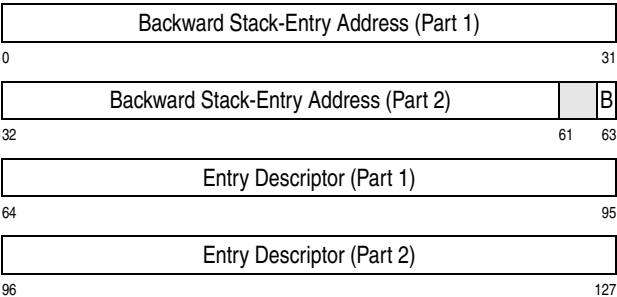
field in the current linkage-stack entry contains all zeros. This field in a trailer entry has no meaning. When the stacking process forms a state entry, it places zeros in the next-entry-size field of the new entry, and it places the size of the new entry in the next-entry-size field of the preceding entry. When the unstacking process logically deletes a state entry, it places zeros in the next-entry-size field of the preceding entry, which entry becomes the current entry.

Bits 48-63 are set to zeros in a state entry when the entry is formed during the stacking process. In a header entry, trailer entry, or state entry, bits 48-63 are reserved for possible future extensions and should always be zeros.

Programming Note: No entry-type code will be assigned in which the leftmost bit of the code is one. The control program can temporarily set the leftmost bit to one in the entry-type code of the current linkage-stack entry (a header entry or a state entry) to prevent the successful execution of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN.

Header Entries

A header entry is at the beginning of each linkage-stack section. The header entry is 16 bytes in length and has the following format:



The fields in the first eight bytes of the header entry are allocated as follows:

Backward Stack-Entry Validity Bit (B): Bit 63 when one, specifies that the preceding linkage-stack section is available and that the backward stack-entry address, bits 0-60 is valid. Bit 63 is set to one during the stacking process when the process proceeds to this section from the preceding one because there is not enough space available in the preceding section to perform the process. During the unstacking process when this header entry is the current linkage-

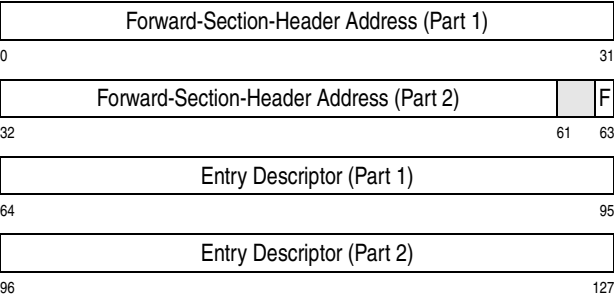
stack entry, a stack-empty exception is recognized if bit 63 is zero.

Backward Stack-Entry Address (BSEA): When bit 63 is one, bits 0-60 with three zeros appended on the right, form the 64-bit home virtual address of the entry descriptor of the last linkage-stack entry, other than the trailer entry, in the preceding linkage-stack section. However, if the current linkage-stack entry is in the preceding or an earlier linkage-stack section, bits 0-60 may have no meaning because the entry they designate, and earlier entries, may have been logically deleted. Bits 0-60 are set during the stacking process when the process proceeds to this section from the preceding one because there is not enough space available in the preceding section to perform the process. During the unstacking process when this header entry is the current linkage-stack entry and bit 63 is one, the entry is treated as the current entry. 0-60 is treated as the current entry.

Bits 61 and 62 are set to zeros when bits 0-60 are set during the stacking process. Bits 61 and 62 are reserved for possible future extensions.

Trailer Entries

A trailer entry is at the end of each linkage-stack section. The trailer entry begins immediately after the area specified by the remaining-free-space field in the entry descriptors of the header entry and each state entry in the same linkage-stack section. The trailer entry is 16 bytes in length and has the following format:



The fields in the first eight bytes of the trailer entry are allocated as follows:

Forward-Section Validity Bit (F): Bit 63 when one, specifies that the next linkage-stack section is available and that the forward-section-header address, bits is valid. During the stacking process when there is not enough space available in the current linkage-stack section to perform the process, a

stack-full exception is recognized if bit 63 in the trailer entry of the current section is zero.

Forward-Section-Header Address (FSHA):

When bit 63 is one, bits 0-60, with three zeros appended on the right, form the 64-bit home virtual address of the entry descriptor of the header entry in the next linkage-stack section. During the stacking process when there is not enough space available in the current section to perform the process and bit 63 is one, the header entry designated by bits 0-60 becomes the current linkage-stack entry.

Bits 61 and 62 are reserved for possible future extensions.

Programming Note: All of the fields in the trailer entry are set only by the control program.

State Entries

Zero, one, or more state entries may follow the header entry in each linkage-stack section. A state entry may be a branch state entry, formed by a BRANCH AND STACK instruction, or a program-call state entry, formed by a stacking PROGRAM CALL instruction. The state entry is 296 bytes in length and has the following format:

Hex	Dec	Contents	Bytes
0 8 : 70 78	0 8 : 112 120	Contents of General Registers 0 - 15	128
80 88 : D0 D8	128 136 : 208 216	Other Status Information	96
E0 E8 : 110 118	224 232 : 272 280	Contents of Access Registers 0 - 15	64
120	288	Entry Descriptor	8

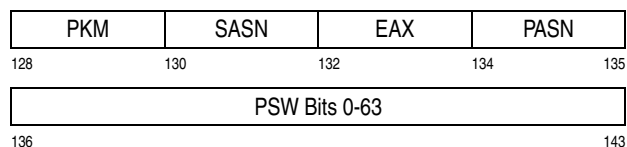
Bytes 0-127 of the state entry contain the contents of general registers 0-15 in the ascending order of the register numbers. Bytes 224-287 contain the con-

tents of access registers 0-15 in the ascending order of the register numbers. The contents of these fields are moved from the registers to the state entry during the BRANCH AND STACK and stacking PROGRAM CALL operations. The contents of general registers 2-14 and access registers 2-14 are restored from the state entry to the registers during the PROGRAM RETURN operation. The contents of a specified range of general registers and access registers can be restored from the state entry to the registers by EXTRACT STACKED REGISTERS.

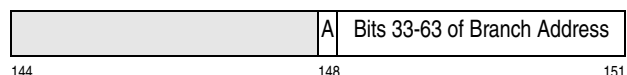
Bytes 128-223 of the state entry contain the other status information that is placed in the entry by BRANCH AND STACK, stacking PROGRAM CALL,

and MODIFY STACKED STATE. A portion of this status information is restored to the PSW and control registers by PROGRAM RETURN, and all of the information can be examined by means of EXTRACT STACKED STATE. Bytes 288-295 contain the entry descriptor. EXTRACT STACKED STATE sets the condition code to indicate whether the entry-type code in the entry descriptor specifies a branch state entry or a program-call state entry.

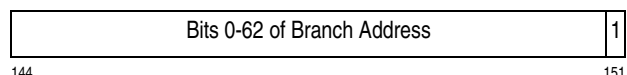
Bytes 128-223 of the state entry have the following detailed format:



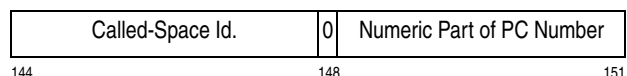
In a Branch State Entry Made in 24-Bit or 31-Bit Mode



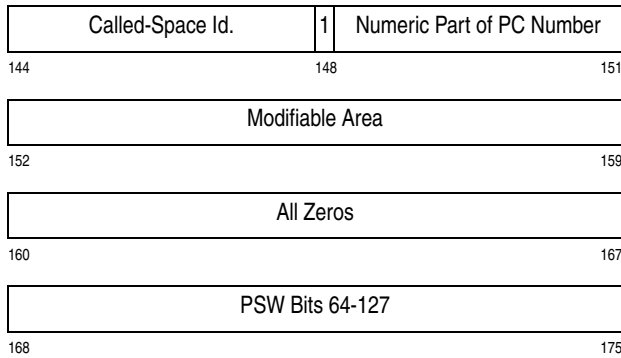
In a Branch State Entry Made in 64-Bit Mode



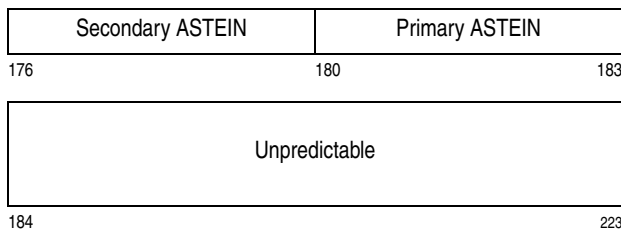
In a Program-Call State Entry Made When Resulting Mode Is 24 Bit or 31 Bit



In a Program-Call State Entry Made When Resulting Mode Is 64 Bit



If ASN-and-LX Reuse Is Enabled; otherwise Unpredictable



The fields in bytes 128-183 are allocated as follows. In the following, "of the calling program" means the value existing at the beginning of the execution of the BRANCH AND STACK or stacking PROGRAM CALL instruction that formed the state entry.

PSW-Key Mask (PKM): Bytes 128-129 contain the PSW-key mask, bits 32-47 of control register 3, of the calling program. The PSW-key mask is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL, and it is restored to the control register by a PROGRAM RETURN instruction that unstacks an entry formed by stacking PROGRAM CALL.

Secondary ASN (SASN): Bytes 130-131 contain the secondary ASN, bits 48-63 of control register 3, of the calling program. The SASN is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL, and it is restored to the control register by a PROGRAM RETURN instruction that unstacks an entry formed by stacking PROGRAM CALL.

Extended Authorization Index (EAX): Bytes 132-133 contain the extended authorization index, bits 32-47 of control register 8, of the calling program. The EAX is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL, and it is

restored to the control register by a PROGRAM RETURN instruction that unstacks an entry formed by stacking PROGRAM CALL.

Primary ASN (PASN): Bytes 134-135 contain the primary ASN, bits 48-63 of control register 4, of the calling program. The PASN is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL, and it is restored to the control register by a PROGRAM RETURN instruction that unstacks an entry formed by stacking PROGRAM CALL.

Program-Status Word (PSW): In a branch state entry formed by a BRANCH AND STACK instruction in which the R_1 field is zero, and in a program-call state entry, bytes 136-143 and 168-175 contain the updated PSW of the calling program. Bytes 136-143 contain bits 0-63 of the PSW, and bytes 168-175 contain bits 64-127 of the PSW. Thus, the basic and extended addressing-mode bits in this PSW specify the addressing mode of the calling program, and the instruction address designates the next sequential instruction following the BRANCH AND STACK or stacking PROGRAM CALL instruction that formed the state entry, or following an execute-type instruction that had the BRANCH AND STACK or stacking PROGRAM CALL instruction as its target instruction. In a branch state entry formed by a BRANCH AND STACK instruction in which the R_1 field is nonzero, bytes 136-143 and 168-175 contain the PSW of the calling program, except that the extended-addressing-mode bit in bit position 31 of bytes 136-139, the basic-addressing-mode bit in bit position 0 of byte 140, and the instruction address in bytes 168-175 are as specified by the contents of the general register designated by the R_1 field. See the definition of BRANCH AND STACK in Chapter 10, "Control Instructions" for how the basic- and extended-addressing-mode bits and instruction address are specified. The value of the PER mask in bytes 136-143 is always unpredictable. The PSW is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL and is restored as the current PSW by PROGRAM RETURN, except that the PER mask is not restored. PROGRAM RETURN does not change the PER mask in the current PSW.

Basic Addressing Mode (A): In a branch state entry made in the 24-bit or 31-bit addressing mode, bit position 0 of bytes 148-151 contains the basic-addressing-mode bit, bit 32 of the PSW, at the end of the execution of the BRANCH AND STACK instruction that formed the state entry. The basic-addressing-mode bit is saved in bit position 0 of bytes

148-151 by BRANCH AND STACK. BRANCH AND STACK does not change the basic-addressing-mode bit in the PSW.

Branch Address: In a branch state entry made in the 24-bit or 31-bit addressing mode, bit positions 1-31 of bytes 148-151 contain bits 33-63 of the instruction address in the PSW at the end of the execution of the BRANCH AND STACK instruction that formed the state entry, and the contents of bytes 144-147 are unpredictable. In a branch state entry made in the 64-bit addressing mode, bytes 144-151 contain bits 0-62 of that instruction address with a one appended on the right. The instruction address is saved in bytes 148-151 or 144-151 (depending on the addressing mode) by BRANCH AND STACK. When the R_2 field of BRANCH AND STACK is non-zero, the instruction causes branching, and the instruction address in bytes 148-151 or 144-151 is the branch address. When the R_2 field of BRANCH AND STACK is zero, the instruction is executed without branching, and the instruction address in bytes 148-151 or 144-151 is the address of the next sequential instruction following the BRANCH AND STACK instruction, or following an execute-type instruction that had the BRANCH AND STACK instruction as its target instruction.

Called-Space Identification: In a program-call state entry, bytes 144-147 contain the called-space identification (CSI). The CSI is saved in the state entry by stacking PROGRAM CALL.

If the PROGRAM CALL operation was space switching, bytes 0 and 1 of the CSI (bytes 144 and 145 of the state entry) contain the new primary ASN that was placed in control register 4 by the PROGRAM CALL instruction. When the ASN-and-LX-reuse facility is not enabled, bytes 2 and 3 of the CSI (bytes 146 and 147 of the state entry) contain the rightmost two bytes of the ASTE sequence number (ASTESN) in the new primary ASTE whose address was placed in control register 5 by the PROGRAM CALL instruction. When the ASN-and-LX-reuse facility is installed and enabled, bytes 2 and 3 of the CSI (bytes 146 and 147 of the state entry) contain the rightmost two bytes of the ASTE instance number (ASTEIN) in the new primary ASTE whose address was placed in control register 5 by the PROGRAM CALL instruction.

If the PROGRAM CALL operation was the to-current-primary operation, the CSI is all zeros.

Numeric Part of PC Number: In a program-call state entry, bit positions 1-31 of bytes 148-151 contain the numeric part of the PC number used by the stacking PROGRAM CALL instruction that formed the entry. When ASN-and-LX reuse is not enabled, or when it is and bit 44 of the effective address used by stacking PROGRAM CALL is zero, stacking PROGRAM CALL places bits 44-63 of the effective address, with 11 zeros appended on the left, in bit positions 1-31 of bytes 148-151. When ASN-and-LX reuse is enabled and bit 44 of the effective address is one, stacking PROGRAM CALL places bits 45-63 of the effective address, with bits 32-43 of the effective address appended on the left, in bit positions 1-31 of bytes 148-151. In any case, stacking PROGRAM CALL places a zero in bit position 0 of the bytes if the resulting addressing mode is the 24-bit or 31-bit mode or a one in bit position 0 if the resulting addressing mode is the 64-bit mode.

Modifiable Area: Bytes 152-159 are the field that is set by MODIFY STACKED STATE. BRANCH AND STACK and stacking PROGRAM CALL place all zeros in bytes 152-159.

Secondary ASTEIN (SASTEIN): If the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control in control register 0, bytes 176-179 contain the secondary ASTEIN, bits 0-31 of control register 3, of the calling program. The SASTEIN is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL. When PROGRAM RETURN unstacks a program-call state entry, it compares the SASTEIN in the state entry to the ASTEIN in the ASN-second-table entry if SASN translation occurs, and it always restores the SASTEIN to control register 3.

Primary ASTEIN (PASTEIN): If the ASN-and-LX-reuse facility is installed and is enabled by the ASN-and-LX-reuse control in control register 0, bytes 180-183 contain the primary ASTEIN, bits 0-31 of control register 4, of the calling program. The PASTEIN is saved in the state entry by BRANCH AND STACK or stacking PROGRAM CALL. When PROGRAM RETURN unstacks a program-call state entry, it compares the PASTEIN in the state entry to the ASTEIN in the ASN-second-table entry if PASN translation occurs, and it always restores the PASTEIN to control register 4.

All zeros are placed in bytes 160-167 by BRANCH AND STACK and stacking PROGRAM CALL.

The contents of bytes 184-223 are unpredictable.

Stacking Process

The stacking process is performed as part of a BRANCH AND STACK or stacking PROGRAM CALL operation. The process locates space for a new linkage-stack state entry, forms the entry, updates the next-entry-size field in the preceding entry, and updates the linkage-stack-entry address in control register 15 so that the new entry becomes the current linkage-stack entry.

For the stacking process to be performed successfully, DAT must be on and the CPU must be in the primary-space mode or access-register mode; otherwise, a special-operation exception is recognized, and the operation is suppressed.

Except as just mentioned, the stacking process is performed independent of the current addressing mode and translation mode, as specified by bits 31, 32, 16, and 17 of the current PSW. All addresses used during the stacking process are always 64-bit home virtual addresses.

During the stacking process when any address is formed through the addition or subtraction of a value to or from another address, a carry out of, or a borrow into, bit position 0 of the address, if any, is ignored.

When the stacking process fetches or stores by using an address that designates, after translation, a location that is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Key-controlled protection does not apply to the accesses made during the stacking process, but DAT protection and low-address protection do apply. A protection exception causes the operation to be suppressed.

Locating Space for a New Entry

The linkage-stack-entry address in control register 15 is used to locate the current linkage-stack entry. Bits 0-60 of control register 15, with three zeros appended on the right, form the 64-bit home virtual address of the leftmost byte of the entry descriptor of the current linkage-stack entry.

The first word of the entry descriptor of the current linkage-stack entry is fetched by using the 64-bit home virtual address. This fetch is for the purpose of obtaining the section-identification and remaining-free-space fields in the word; the unstack-suppression bit and entry-type field in the word are not examined.

The 16-bit unsigned binary value in the remaining-free-space field, bits 16-31 of the entry descriptor, is compared against the size in bytes of the linkage-stack entry to be formed. The size of a state entry is 296 bytes. If the value in the field is equal to or greater than the size of the entry to be formed, processing continues as described in "Forming the New Entry" on page 5-81; otherwise, processing continues as described below.

When the remaining-free-space field in the current linkage-stack entry indicates that there is not enough space available in the current linkage-stack section to form the new entry, the first doubleword of the trailer entry of the current section is fetched. The address for fetching this doubleword is determined as follows: to the address formed from the contents of control register 15, add 8 to address the first byte after the entry descriptor of the current entry, and then add the contents of the remaining-free-space field of the current entry to address the first byte of the trailer entry. The remaining-free-space value used in the addition must be a multiple of 8; otherwise, a stack-specification exception is recognized, and the operation is nullified.

If the forward-section-validity bit, bit 63 of the trailer entry is zero, a stack-full exception is recognized, and the operation is nullified; otherwise, the forward-section-header address in the trailer entry is used to locate the header entry in the next linkage-stack section. Bits 0-60 of the trailer entry, with three zeros appended on the right, form the 64-bit home virtual address of the leftmost byte of the entry descriptor of the header entry in the next section.

The first word of the entry descriptor of the header entry in the next linkage-stack section is fetched. This fetch is for the purpose of obtaining the section-identification and remaining-free-space fields in the word; the unstack-suppression bit and entry-type field in the word are not examined.

The value in the remaining-free-space field of the header entry in the next linkage-stack section is compared against the size in bytes of the entry to be

formed. If the value in the field is equal to or greater than the size of the entry to be formed, the following occurs:

- The linkage-stack-entry address, bits 0-60 of control register 15, is placed, as the backward stack-entry address, in bit positions 0-60 of the header entry in the next linkage-stack section, and zeros are placed in bit positions 61 and 62.
- The backward stack-entry validity bit, bit 63, in the header entry in the next section is set to one.
- Bits 0-60 of the 64-bit home virtual address of the entry descriptor of the header entry in the next section are placed in bit positions 0-60 of control register 15, and zeros are placed in bit positions 61-63 of control register 15. Thus, the header entry in the next section becomes the current linkage-stack entry, and the next section becomes the current linkage-stack section.
- Processing continues as described in “Forming the New Entry”.

If the value in the remaining-free-space field of the header entry in the next section (before the next section becomes the current section) is less than the size of the linkage-stack entry to be formed, a stack-specification exception is recognized, and the operation is nullified.

Forming the New Entry

When the remaining-free-space field in the current linkage-stack entry indicates that there is enough space available in the current linkage-stack section to form the new entry, the new entry is formed beginning immediately after the entry descriptor of the current entry.

The new entry is a state entry. The contents of general registers 0-15 are stored in bytes 0-127 of the new entry, in the ascending order of the register numbers. The contents of access registers 0-15 are stored in bytes 224-287 of the new entry, in the ascending order of the register numbers. The PSW-key mask, bits 32-47 of control register 3; secondary ASN, bits 48-63 of control register 3; extended authorization index, bits 32-47 of control register 8; and primary ASN, bits 48-63 of control register 4, are stored in bytes 128-129, 130-131, 132-133, and 134-135,

respectively, of the new entry. The current PSW, in which the instruction address has been updated, is stored in bytes 136-143 and 168-175 of the new entry. Bytes 136-143 contain bits 0-63 of the PSW, and bytes 168-175 contain bits 64-127 of the PSW. However, the value of the PER mask, bit 1 in the PSW stored, is unpredictable. Also, if the instruction being executed is a BRANCH AND STACK instruction in which the R_1 field is nonzero, the extended- and basic-addressing-mode bits stored in bytes 139 and 140, respectively, of the new entry, and the instruction address stored in bytes 168-175 of the new entry, are as specified by the contents of the general register designated by the R_1 field.

If the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, the secondary ASTEIN, bits 0-31 of control register 3, and the primary ASTEIN, bits 0-31 of control register 4, are stored in bytes 176-179 and 180-183, respectively, of the new entry.

When the instruction is PROGRAM CALL, the called-space identification is stored in bytes 144-147 of the new entry. When the instruction is performing the space-switching PROGRAM CALL operation and the ASN-and-LX-reuse facility is not enabled, the called-space identification is the two-byte ASN, bytes 10 and 11 in the entry-table entry used by the instruction, followed by bytes 2 and 3 of the ASTE sequence number, bytes 2 and 3 being bits 176-191, in the ASN-second-table entry specified by the ASN. When the instruction is performing the space-switching PROGRAM CALL operation and the ASN-and-LX-reuse facility is installed and enabled, the called-space identification is the two-byte ASN, bytes 10 and 11 in the entry-table entry used by the instruction, followed by bytes 2 and 3 of the ASTE instance number, bytes 2 and 3 being bits 368-383, in the ASN-second-table entry specified by the ASN. When the instruction is performing the to-current-primary PROGRAM CALL operation, the called-space identification is all zeros.

When the instruction is BRANCH AND STACK in the 24-bit or 31-bit addressing mode, the basic-addressing-mode bit from the current PSW is stored in bit position 0 of byte 148 in the state entry, bits 33-63 of the branch address, or of the updated instruction address if the operation is performed without branching, are stored in bit positions 1-31 of bytes 148-151, and the contents of bytes 144-147 are unpredictable. In the 64-bit addressing mode, bits 0-62 of the

branch address or updated instruction address, with a one appended on the right, are stored in bytes 144-151 of the state entry.

When the instruction is PROGRAM CALL, the numeric part of the PC number used, with 11 zeros appended on the left if the number is 20 bits, is stored in bit positions 1-31 of bytes 148-151. If the resulting addressing mode after the execution of PROGRAM CALL is the 24-bit or 31-bit addressing mode, a zero is stored in bit position 0 of byte 148. If the resulting addressing mode is the 64-bit addressing mode, a one instead of a zero is stored in bit position 0 of byte 148.

Zeros are stored in bytes 152-167 of the new entry. The contents of bytes 176-223 are unpredictable, or the contents of bytes 184-223 are unpredictable if the secondary ASTEIN and primary ASTEIN were stored.

Bytes 288-295 of the new entry are its entry descriptor. The unstack-suppression bit, bit 0, of this entry descriptor is set to zero. The code 0001100 binary is stored in the entry-type field, bits 1-7, of this entry descriptor if the instruction being executed is BRANCH AND STACK. The code 0001101 binary is stored if the instruction is PROGRAM CALL. The value in the section-identification field of the current linkage-stack entry is stored in the section-identification field, bits 8-15, of this entry descriptor. The value in the remaining-free-space field of the current entry, minus the size in bytes of the new entry, is stored in the remaining-free-space field of this entry descriptor. Zeros are stored in the next-entry-size field, bits 32-47, and in bit positions 48-63 of this entry descriptor.

The stores into the new entry appear to be word concurrent as observed by other CPUs. The order in which the stores occur is unpredictable.

Updating the Current Entry

The size in bytes of the new linkage-stack entry is stored in the next-entry-size field of the current entry. The remainder of the current entry remains unchanged.

The order of the stores into the current entry and the new entry is unpredictable.

Updating Control Register 15

Bits 0-60 of the 64-bit home virtual address of the entry descriptor of the new linkage-stack entry are placed in bit positions 0-60 of control register 15, the linkage-stack-entry address. Zeros are placed in bit positions 61-63 of control register 15. Thus, the new entry becomes the current linkage-stack-entry.

Recognition of Exceptions during the Stacking Process

The exceptions which can be encountered during the stacking process and their priority are described in the definitions of the BRANCH AND STACK and PROGRAM CALL instructions.

Programming Note: Any exception recognized during the execution of BRANCH AND STACK and PROGRAM CALL causes either nullification or suppression. Therefore, if an exception is recognized, the stacking process does not store into any linkage-stack entry or change the contents of control register 15.

Unstacking Process

The unstacking process is performed as part of the PROGRAM RETURN operation. The process locates the last state entry in the linkage stack, restores a portion of the information in the entry to the CPU registers, updates the next-entry-size field in the preceding entry, and updates the linkage-stack-entry address in control register 15 so that the preceding entry becomes the current linkage-stack entry. The part of the unstacking process that locates the last state entry is also performed as part of the EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE operations.

For the unstacking process to be performed successfully, DAT must be on and the CPU must be in the primary-space mode or access-register mode; otherwise, a special-operation exception is recognized, and the operation is suppressed. However, when the unstacking process is performed as part of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY STACKED STATE, the CPU may be in the primary-space, access-register, or home-space mode.

Except as just mentioned, the unstacking process is performed independent of the current addressing

mode and translation mode, as specified by bits 31, 32, 16, and 17 of the current PSW. All addresses used during the unstacking process are always 64-bit home virtual addresses.

During the unstacking process when any address is formed through the addition or subtraction of a value to or from another address, a carry out of, or a borrow into, bit position 0 of the address, if any, is ignored.

When the unstacking process fetches or stores by using an address that designates, after translation, a location that is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Key-controlled protection does not apply to the accesses made during the unstacking process, but DAT protection and low-address protection do apply. A protection exception causes the operation to be suppressed.

Locating the Current Entry and Processing a Header Entry

The linkage-stack-entry address in control register 15 is used to locate the current linkage-stack entry. Bits 0-60 of control register 15, with three zeros appended on the right, form the 64-bit home virtual address of the leftmost byte of the entry descriptor of the current linkage-stack entry.

The first word of the entry descriptor of the current linkage-stack entry is fetched by using the 64-bit home virtual address. If the entry-type code in bits 1-7 of the entry descriptor is not 0001001 binary, indicating that the entry is not a header entry, processing continues as described in “Checking for a State Entry”; otherwise, processing continues as described below.

When the entry-type code in the current linkage-stack entry is 0001001 binary, indicating a header entry, the next processing depends on which instruction is being executed. When the unstacking process is performed as part of the PROGRAM RETURN operation and the unstack-suppression bit, bit 0, in the entry descriptor of the current entry is one, a stack-operation exception is recognized, and the operation is nullified. When the unstacking process is performed as part of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY

STACKED STATE, the unstack-suppression bit is ignored.

When there is not an exception due to the unstack-suppression bit, the first doubleword of the current linkage-stack entry (a header entry) is fetched. The address of this doubleword is determined by subtracting 8 from the address of the entry descriptor of the current entry.

If the backward stack-entry validity bit, bit 63, of the current entry is zero, a stack-empty exception is recognized, and the operation is nullified; otherwise, the backward stack-entry address in the current entry is used to locate a linkage-stack entry referred to here as the designated entry. Bits 0-60 of the current entry, with three zeros appended on the right, form the 64-bit home virtual address of the leftmost byte of the entry descriptor of the designated entry.

It is assumed in this definition of the unstacking process that the designated linkage-stack entry is the last entry, other than the trailer entry, in the preceding linkage-stack section. This assumption does not imply any processing that is not explicitly described.

The first word of the entry descriptor of the designated entry is fetched. If the entry-type code in this entry descriptor is not 0001001 binary, indicating that the entry is not a header entry, the following occurs:

- When the unstacking process is performed as part of the PROGRAM RETURN operation, bits 0-60 of the 64-bit home virtual address of the entry descriptor of the designated entry are placed in bit positions 0-60 of control register 15, and zeros are placed in bit positions 61-63 of control register 15. Thus, the designated entry becomes the current linkage-stack entry, and the preceding section (based on the assumption) becomes the current linkage-stack section. When the unstacking process is performed as part of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY STACKED STATE, the contents of control register 15 remain unchanged, but the designated entry is temporarily, during the remainder of the definition of the instruction, referred to as the current linkage-stack entry.
- Processing continues as described in “Checking for a State Entry”.

If the entry-type code in the designated entry is 0001001 binary, indicating a header entry, a stack-specification exception is recognized, and the operation is nullified.

Checking for a State Entry

When the entry-type code in the current linkage-stack entry indicates that the entry is not a header entry, the code is checked for being 0001100 or 0001101 binary, which are the codes assigned to a state entry.

If the current linkage-stack entry is a state entry, the next processing depends on which instruction is being executed. When the unstacking process is performed as part of the PROGRAM RETURN operation, processing continues as described in “Restoring Information” on page 5-84. When the process is performed as part of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, or MODIFY STACKED STATE, the process is completed; that is, no additional processing occurs as a part of the unstacking process.

If the current linkage-stack entry is not a state entry (and necessarily not a header entry either), a stack-type exception is recognized, and the operation is nullified.

Restoring Information

The remaining parts of the unstacking process occur only in the PROGRAM RETURN operation.

The current linkage-stack entry is a state entry. If the unstack-suppression bit in the entry is one, a stack-operation exception is recognized, and the operation is nullified.

When there is not an exception due to the unstack-suppression bit, a portion of the contents of the current linkage-stack entry are restored to the CPU registers. The contents of general registers 2-14 and access registers 2-14 are restored to those registers from where they were saved in the current entry by the stacking process. When the entry-type code in the current entry is 0001101 binary, indicating a program-call state entry, the PSW-key mask and secondary ASN in control register 3, extended authorization index in control register 8, and primary ASN in control register 4 are similarly restored. During this restoration, the authorization index in control register 4 and the monitor masks and enhanced-monitor masks in control register 8 remain

unchanged. (The authorization index may be changed by the part of the PROGRAM RETURN execution that occurs after the unstacking process.)

When the state entry is a program-call state entry, and if the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control in control register 0, the secondary ASTEIN in control register 3 and the primary ASTEIN in control register 4 are restored from where they were saved in the state entry. However, the primary ASTEIN and secondary ASTEIN may be subject to certain tests, as described in the PROGRAM RETURN definition.

When the entry-type code is 0001100 binary, indicating a branch state entry, the PSW-key mask, secondary ASN, extended authorization index, primary ASN, secondary ASTEIN, and primary ASTEIN in the current entry are ignored, and all contents of the control registers remain unchanged.

When the current entry is either a branch state entry or a program-call state entry, bits 0-63 and 64-127 of the current PSW are restored from bytes 136-143 and bytes 168-175, respectively, of the entry, except that the PER mask is not restored. The PER mask in the current PSW remains unchanged. Bytes 144-159 and bytes 160-167 of the current entry are ignored.

The fetches from the current entry appear to be word concurrent as observed by other CPUs. The order in which the fetches occur is unpredictable.

Updating the Preceding Entry

Zeros are stored in the next-entry-size field, bits 32-47, of the entry descriptor of the preceding linkage-stack entry. The remainder of the preceding entry remains unchanged. The address of the entry descriptor of the preceding entry is determined by subtracting the size in bytes of the current entry from the address of the entry descriptor of the current entry.

The order of the store into the preceding entry and the fetches from the current entry is unpredictable.

Updating Control Register 15

Bits 0-60 of the 64-bit home virtual address of the entry descriptor of the preceding linkage-stack entry are placed in bit positions 0-60 of control register 15, the linkage-stack-entry address. Zeros are placed in bit positions 61-63 of control register 15. Thus, the

preceding entry becomes the current linkage-stack entry.

Recognition of Exceptions during the Unstacking Process

The exceptions which can be encountered during the unstacking process and their priority are described in the definition of the PROGRAM RETURN instruction. The exceptions which apply to EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, and MODIFY STACKED STATE are described in the definitions of those instructions.

Programming Notes:

1. Any exceptions recognized during the execution of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN cause either nullification or suppression. Therefore, if an exception is recognized, the unstacking process does not change the contents of any CPU register (except for updating the instruction address in the PSW in the case of suppression) or store into any linkage-stack entry.
2. The unstacking process in PROGRAM RETURN does not restore the PER mask in the PSW so that an act of turning PER on or off after the execution of the related BRANCH AND STACK or PROGRAM CALL instruction but before the execution of the PROGRAM RETURN instruction will not be counteracted. When PROGRAM CALL or PROGRAM RETURN is space switching, the space-switch event can be used as a signal to turn PER on or off, if desired.

Sequence of Storage References

The following sections describe the effects which can be observed in storage due to overlapped operations and piecemeal execution of a CPU program. Most of the effects described in these sections are observable only when two or more CPUs or channel programs are in simultaneous execution and access common storage locations. Thus, most of the effects need be taken into account by a program only if the program interacts with another CPU or a channel program.

Some of the effects described in the following sections are independent of interaction with another CPU or a channel program. These effects, which are therefore more readily observable, relate to prefetched instructions and overlapping operands of a single instruction. These effects are described in "Conceptual Sequence" and in "Interlocks for Virtual-Storage References" on page 5-86.

Conceptual Sequence

The CPU conceptually processes instructions one at a time, with the execution of one instruction preceding the execution of the following instruction. The execution of the instruction designated by a successful branch follows the execution of the branch. Similarly, an interruption takes place between instructions or, for interruptible instructions, between units of operation of such instructions.

The sequence of events implied by the processing just described is sometimes called the conceptual sequence.

Each operation of instruction execution appears to the program itself to be performed sequentially, with the current instruction being fetched after the preceding operation is completed and before the execution of the current operation is begun. This appearance is maintained even though the storage-implementation characteristics and overlap of instruction execution with storage accessing may cause actual processing to be different. The results generated are those that would have been obtained had the operations been performed in the conceptual sequence. Thus, it is possible for an instruction to modify the next succeeding instruction in storage.

A case in which the copies of prefetched instructions are not necessarily discarded occurs when the fetch and the store are done by means of different effective addresses that map to the same real address. This case is described in more detail in "Interlocks for Virtual-Storage References" on page 5-86.

Overlapped Operation of Instruction Execution

In simple models in which operations are not overlapped, the conceptual and actual sequences are essentially the same. However, in more complex machines, overlapped operation, buffering of operands and results, and execution times which are comparable to the propagation delays between units can cause the actual sequence to differ considerably

from the conceptual sequence. In these machines, special circuitry is employed to detect dependencies between operations and ensure that the results obtained, as observed by the CPU which generates them, are those that would have been obtained if the operations had been performed in the conceptual sequence. However, other CPUs and channel programs may, unless otherwise constrained, observe a sequence that differs from the conceptual sequence.

Divisible Instruction Execution

It can normally be assumed that the execution of each instruction occurs as an indivisible event. However, in actual operation, the execution of an instruction consists in a series of discrete steps. Depending on the instruction, operands may be fetched and stored in a piecemeal fashion, and some delay may occur between fetching operands and storing results. As a consequence, intermediate or partially completed results may be observable by other CPUs and by channel programs.

When a program interacts with the operation on another CPU, or with a channel program, the program may have to take into consideration that a single operation may consist in a series of storage references, that a storage reference may in turn consist in a series of accesses, and that the conceptual and observed sequences of these accesses may differ.

Storage references associated with instruction execution are of the following types: instruction fetches, ART-table and DAT-table fetches, and storage-operand references. For the purpose of describing the sequence of storage references, accesses to storage in order to perform ASN translation, PC-number translation, tracing, and the linkage-stack stacking and unstacking processes are considered to be storage-operand references.

Programming Note: The sequence of execution of a CPU may differ from the simple conceptual definition in the following ways:

- As observed by the CPU itself, instructions may appear to be prefetched when different effective addresses are used. (See “Interlocks for Virtual-Storage References”.)
- As observed by other CPUs and by channel programs, the execution of an instruction may appear to be performed as a sequence of piece-

meal steps. This is described for each type of storage reference in the following sections.

- As observed by other CPUs and by channel programs, the storage-operand accesses associated with one instruction are not necessarily performed in the conceptual sequence. (See “Relation between Operand Accesses” on page 5-98.)
- As observed by channel programs, in certain unusual situations, the contents of storage may appear to change and then be restored to the original value. (See “Storage Change and Restoration for DAT-Associated Access Exceptions” on page 5-23.)

Interlocks for Virtual-Storage References

As described in the immediately preceding sections, CPU operation appears, with certain exceptions, to be performed sequentially as observed by the CPU itself; the stores performed by one instruction generally appear to be completed before the next instruction and its operands are fetched. This appearance is maintained in overlapped machines by means of interlock circuitry that detects accesses to a common storage location.

For those instructions which alter the contents of storage and have more than one operand, the instruction definition normally describes the results that are obtained when the operands overlap in storage, this definition being in terms of a sequence of stores and fetches. The interlock circuitry is used in determining whether operand overlap exists.

The purpose of this section is to define those cases in which the machine must appear to operate sequentially, and in which operands of a single instruction must or must not be treated as overlapping.

Proper operation is provided in part by comparing effective addresses. For the purpose of this definition, the term “effective address” means an address before translation, if any, regardless of whether the address is virtual, real, or absolute. If two effective addresses have the same value, the effective addresses are said to be the same even though one may be real or in a different address space.

The values of two virtual effective addresses do not necessarily indicate whether or not the addresses designate the same storage location. The address-translation tables may be set up so that different effective addresses map to the same real address, or so that the same effective address in different address spaces maps to different real addresses.

The interlocks for virtual-storage references are considered in two situations: storage references of one instruction as they affect storage references of another instruction, and multiple storage references of a single instruction.

Interlocks between Instructions

As observed by the CPU itself, the storage accesses for operands for each instruction appear to occur in the conceptual sequence independent of the effective address used. That is, the operand stores for one instruction appear to be completed before the operand fetches for the next instruction occur. For instruction fetches, the operand stores for one instruction necessarily appear to be completed before the next instruction is fetched only when the same effective address is used for the operand store and the instruction fetch.

When an instruction changes the contents of a main-storage location in which a conceptually subsequent instruction is to be executed, either directly or by means of an execute-type instruction, and when different effective addresses are used to designate that location for storing the result and fetching the instruction, the instruction may appear to be fetched before the store occurs. If an intervening operation causes the prefetched instructions to be discarded, then the updated value is recognized. A definition of when prefetched instructions must be discarded is included in “Instruction Fetching” on page 5-89.

Any change to the storage key appears to be completed before the conceptually following reference to the associated storage block is made, regardless of

whether the reference to the storage location is made by means of a virtual, real, or absolute address. Analogously, any conceptually prior references to the storage block appear to be completed when the key for that block is changed or inspected.

Interlocks within a Single Instruction

For those instructions which alter the contents of storage and have more than one operand, the instruction definition normally describes the results which are obtained when the operands overlap in storage. This result is normally defined in terms of the sequence of the storage accesses; that is, a portion of the results of a store-type operand must appear to be placed in storage before some portion of the other operand is fetched. This definition applies provided that the store and fetch accesses are specified by means of the same effective addresses and the same effective space designations.

When multiple address spaces are involved in the access-register mode, the term “effective space designation” is used to denote the value used by the machine to determine whether two spaces are the same. In the access-register mode, the 32-bit access-list-entry-token (ALET) value associated with each storage-operand address is called the effective space designation. When a B field of zero is specified, a value of all zeros is used for the effective space designation. If the effective space designations are different, the spaces are considered to be different even if both ALETs map to the same address-space-control-element value.

When the store and the fetch accesses are specified by means of different effective space designations or by means of different effective addresses, the operand and fetch may appear to precede the operand store.

Figure 5-13 on page 5-88 summarizes the cases of overlap and the specified results, including when

MOVE LONG (MVCL) sets condition code 3, for each case.

Effective Space Designations Equal?	Effective Addresses Overlap Destructively?	Operand Overlap Destructively in Real Storage?	Is Overlap Recognized ?	
			MVCL Sets CC 3	Operand Results
Yes	No	No	No	No
		Yes	No	Unpredictable
	Yes	No	—	—
		Yes	Yes	Yes
No	No	No	No	No
		Yes	No	Unpredictable
	Yes	No	No	No
		Yes	No	Unpredictable
Explanation:				
— This case cannot occur.				

Figure 5-13. Virtual-Storage Interlocks within a Single Instruction

Effective space designations may be represented by ALB entries, and the test for whether two effective space designations are the same may be performed by comparing ALB entries. If the program changes an attached and valid ART-table entry without subsequently causing the execution of PURGE ALB or a COMPARE AND SWAP AND PURGE instruction that purges the ALB, two effective space designations that are the same may have different representations in the ALB, and failure to recognize operand overlap may result. The use of the ALB never causes overlap to be recognized when the effective space designations are different.

Programming Note:

A single main-storage location can be accessed by means of more than one address in several ways:

1. The DAT tables may be set up such that multiple addresses in a single address space, or addresses in different address spaces, including the real address specified by a real-space designation, map to a single real address.
2. The translation of logical, instruction, and virtual addresses may be changed by loading the DAT parameters in the control registers, by changing the address-space-control bits in the PSW, or, for logical and instruction addresses, by turning DAT on or off.
3. In the access-register mode, different address spaces may be selected by means of each access register. In addition, the primary address space is selected for instruction fetching and the target of an execute-type instruction.
4. STORE USING REAL ADDRESS performs a store by means of a real address.
5. Certain other instructions also use real addresses (even when a logical address is not translated by means of a real-space designation, which is a situation covered in case 1), and the instructions MOVE TO PRIMARY and MOVE TO SECONDARY access two address spaces.
6. Accesses to storage for the purpose of storing and fetching information for interruptions is performed by means of real addresses, and, for the store-status function, by means of absolute addresses, whereas accesses by the program may be by means of virtual addresses.
7. The real-to-absolute mapping may be changed by means of the SET PREFIX instruction or a reset.
8. A main-storage location may be accessed by channel programs by means of an absolute address and by the CPU by means of an absolute, a real, or a virtual address.
9. A main-storage location may be accessed by another CPU by means of one type of address

and by this CPU by means of a different type of address.

The primary purpose of this section on interlocks is to describe the effects caused in cases 1, 3, and 4, above.

For case 2, no effect is observable because prefetched instructions are discarded when the translation parameters are changed, and the delay of stores by a CPU is not observable by the CPU itself.

For case 5, for those instructions which fetch by using real addresses (for example, LOAD REAL ADDRESS, which fetches a segment-table entry and a page-table entry, and may fetch a region-table entry), no effect is observable because only operand accesses between instructions are involved. All instructions that store by using a real address, except STORE USING REAL ADDRESS, or that store across address spaces, except in the access-register mode, cause prefetched instructions to be discarded, and no effect is observable.

Cases 6 and 7 are situations which are defined to cause serialization, with the result that prefetched instructions are discarded. In these cases, no effect is observable.

The handling of cases 8 and 9 involves accesses as observed by other CPUs and by channel programs and is covered in the following sections in this chapter.

Instruction Fetching

Instruction fetching consists in fetching the one, two, or three halfwords designated by the instruction address in the current PSW. The immediate field of an instruction is accessed as part of an instruction fetch. If, however, an instruction designates a storage operand at the location occupied by the instruction itself, the location is accessed both as an instruction and as a storage operand. The fetch of the target instruction of an execute-type instruction is considered to be an instruction fetch.

The bytes of an instruction may be fetched piecemeal and are not necessarily accessed in a left-to-right direction. The instruction may be fetched multiple times for a single execution; for example, it may be fetched for testing the addressability of operands or

for inspection of PER events, and it may be refetched for actual execution.

Instructions are not necessarily fetched in the sequence in which they are conceptually executed and are not necessarily fetched each time they are executed. In particular, the fetching of an instruction may precede the storage-operand references for an instruction that is conceptually earlier. The instruction fetch occurs prior to all storage-operand references for all instructions that are conceptually later.

An instruction may be prefetched by using a virtual address only when the associated DAT table entries are attached and valid or when entries which qualify for substitution for the table entries exist in the TLB. An instruction that has been prefetched may be interpreted for execution only for the same virtual address for which the instruction was prefetched.

No limit is established on the number of instructions which may be prefetched, and multiple copies of the contents of a single storage location may be fetched. As a result, the instruction executed is not necessarily the most recently fetched copy. Storing caused by other CPUs and by channel programs does not necessarily change the copy of prefetched instructions. However, if a store that is conceptually earlier is made by the same CPU using the same effective address as that by which the instruction is subsequently fetched, the updated information is obtained, except as noted below. If the effective addresses are different, the updated information is not necessarily obtained. However, the updated information is obtained if either execution is in the real mode since prefetched instructions are discarded if DAT is turned on or off.

All copies of prefetched instructions are discarded when:

- A serializing function is performed. However, for PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, SET SECONDARY ASN WITH INSTANCE, and TRACE, it is unpredictable whether or not a store into a trace-table entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store; for PROGRAM CALL and PROGRAM RETURN it is unpredictable whether or not a store into a trace-table entry or linkage-stack entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store. Additionally,

when the store-clock-fast facility is installed and the TRACE TOD-clock control in bit 32 of control register 0 is one, it is unpredictable whether explicit tracing causes serialization to be performed.

- The CPU enters the operating state.
- DAT is turned on or off.
- A change is made to a translation parameter in control register 1 when in the primary-space, secondary-space, or access-register mode, or in control register 13 when in the home-space mode.

The SET ADDRESS SPACE CONTROL instruction can change the translation mode between any of the primary-space, secondary-space, access-register, and home-space modes, and it performs serialization. The SET ADDRESS SPACE CONTROL FAST instruction can perform the same mode changes, but it does not serialize.

Programming Notes:

1. As observed by a CPU itself, its own instruction prefetching may be apparent when different effective addresses map to a single real address. This is described in “Conceptual Sequence” on page 5-85 and “Interlocks for Virtual-Storage References” on page 5-86.
2. Any means of changing PSW bits 16 and 17, except the SET ADDRESS SPACE CONTROL FAST instruction, causes serialization to be performed and prefetched instructions to be discarded. Turning DAT on or off causes prefetched instructions to be discarded. Therefore, any change of the translation mode, except a change made by SET ADDRESS SPACE CONTROL FAST, always causes prefetched instructions to be discarded.
3. The following are some effects of instruction prefetching on one CPU as observed by other CPUs and by channel programs.

It is possible for one CPU to prefetch the contents of a storage location, after which another CPU or a channel program can change the contents of that storage location and then set a flag to indicate that the change has been made. Subsequently, the first CPU can test and find the flag set, branch to the modified location, and execute the original prefetched contents.

It is possible, if another CPU or a channel program concurrently modifies the instruction, for one CPU to recognize the changes to some but not all bit positions of an instruction.

It is possible for one CPU to prefetch an instruction and subsequently, before the instruction is executed, for another CPU to change the storage key. As a result, the first CPU may appear to execute instructions from a protected storage location. However, the copy of the instructions executed is the copy prefetched before the location was protected.

4. Previous versions of both the *ESA/390 Principles of Operation* and the *z/Architecture Principles of Operation* have allowed a CPU to not recognize a store into the instruction stream when the CPU was operating in the home-space or access-register modes. To ensure that stores into the instruction stream were recognized in these modes, the program was required to insert a serializing operation following the store.

No IBM processor has ever implemented this characteristic; stores into the instruction stream using the same effective address as that by which a subsequent instruction is fetched have always been recognized, regardless of the address-space-control mode. This characteristic is now removed from the architecture. Any serializing operations that were inserted to ensure recognition of stores into the instruction stream while in the AR or home-space modes may be safely removed from programs that execute on IBM processors.

ART-Table and DAT-Table Fetches

The access-register-translation (ART) table entries are access-list designations, access-list entries, ASN-second-table entries, and authority-table entries. The dynamic-address-translation (DAT) table entries are region-table entries, segment-table entries, and page-table entries. The fetching of these entries may occur as follows:

1. An ART-table entry may be prefetched into the ART-lookaside buffer (ALB) and used from the ALB without refetching from storage, until the entry is cleared by a COMPARE AND SWAP AND PURGE, PURGE ALB, or SET PREFIX instruction, by CPU reset, or by a set prefix or set-architecture SIGNAL PROCESSOR order. A

DAT-table entry may be prefetched into the translation-lookaside buffer (TLB) and used from the TLB without refetching from storage, until the entry is cleared by a COMPARE AND SWAP AND PURGE, INVALIDATE DAT TABLE ENTRY, INVALIDATE PAGE TABLE ENTRY, PURGE TLB, or SET PREFIX instruction, by CPU reset, or by a set prefix or set-architecture SIGNAL PROCESSOR order. ART-table and DAT-table entries are not necessarily fetched in the sequence conceptually called for; they may be fetched at any time they are attached and valid, including during the execution of conceptually previous instructions.

2. The fetching of access-list designations, access-list entries, ASN-second-table entries, and DAT-table entries appears to be word concurrent as observed by other CPUs, except that the fetching of an address-space-control element from an ASN-second-table entry appears to be double-word concurrent as observed by other CPUs. The reference to an entry may appear to access a single byte at a time as observed by channel programs.
3. The order in which the words of an access-list entry or ASN-second-table entry are fetched is unpredictable, except that the leftmost word of an entry is fetched first. However, the leftmost word of an ASN-second-table entry is not fetched when access-list-entry token 00000000 hex is translated for BRANCH IN SUBSPACE GROUP.
4. An ART-table or DAT-table entry may be fetched even after some operand references for the instruction have already occurred. The fetch may occur as late as just prior to the actual byte access requiring the ART-table or DAT-table entry.
5. An ART-table or DAT-table entry may be fetched for each use of the address, including any trial execution, and for each reference to each byte of each operand.
6. The DAT page-table-entry fetch precedes the reference to the page. When no copy of the page-table entry is in the TLB, the fetch of the associated segment-table entry precedes the fetch of the page-table entry. When no copy of the segment-table entry is in the TLB, the fetch of the region-third-table entry, if one is required, precedes the fetch of the segment-table entry. Similarly, the fetch of a required region-second-table

entry precedes the fetch of the region-first-table entry, and the fetch of a required region-first-table entry precedes the fetch of the region-second-table entry.

7. When no copy of a region-table entry or segment-table entry designated by means of an ART-obtained address-space-control element is in the TLB, the ART fetch of the ASN-second-table entry precedes the DAT region-table-entry or segment-table-entry fetch. When no copy of a required authority-table entry is in the ALB, the ART fetch of the associated ASN-second-table entry precedes the fetch of the authority-table entry. When no copy of a required ASN-second-table entry is in the ALB, the fetch of the associated access-list entry precedes the fetch of the ASN-second-table entry. When no copy of a required access-list entry is in the ALB, the fetch of the associated access-list designation precedes the fetch of the access-list entry.

Storage-Key Accesses

In the following sections, the term *key-setting instruction* refers either to the SET STORAGE KEY EXTENDED instruction or to the PERFORM FRAME MANAGEMENT FUNCTION instruction when the set-key control is one. A *nonquiescing key-setting instruction* refers to a key-setting instruction in which the nonquiescing control is set to one.

References to the storage key are handled as follows:

1. Whenever a reference to storage is made and key-controlled protection applies to the reference, the four access-control bits and the fetch-protection bit associated with the storage location are inspected concurrently. The inspection of the access-control and fetch-protection bits occurs concurrently with the reference to the storage location, except that when a nonquiescing key-setting operation is performed for a storage location by one CPU, the following applies:
 - The inspection of the access-control and fetch-protection bits for the storage location by any CPU may precede a store reference to the location. In the case where the inspection of the access-control and fetch-protection bits precedes the store, the inspection occurs no earlier than after the last serialization operation on the CPU, and no earlier

than the last quiescing key-setting operation for the same storage location on any CPU.

- When a unit of operation or instruction execution on any other CPU causes multiple accesses to the same 4 K-byte block as that of the key-setting operation, the other CPU necessarily inspects the access-control and fetch-protection bits only for the first reference to the block. The other CPU does not necessarily inspect the access-control and fetch-protection bits for the subsequent accesses within the block by the same unit of operation.

When (a) enhanced DAT does not apply, (b) enhanced DAT applies but the storage is accessed by means of a segment-table entry in which the STE-format control is zero, or (c) enhanced DAT applies, the storage is accessed by means of a segment-table entry in which the STE-format control is one, but the ACCF-validity control is zero, the access-control bits and the fetch-protection bit are in bits 0-4 of the storage key for the 4K-byte block.

When enhanced DAT applies and the storage is accessed by means of a segment-table entry in which both the STE-format control and ACCF-validity control are one, it is unpredictable whether bits 0-4 of the storage key or bits 48-52 of the segment-table entry provide the access-control bits and fetch-protection bit. Furthermore, when the segment-table entry provides the access-control bits and fetch-protection bit, a buffered copy from the translation-lookaside buffer may be used.

For accesses made by the channel subsystem, the access-control bits and fetch-protection bits are in bits 0-4 of the storage key for the 4K-byte block.

2. When storing is performed by a CPU, the change bit is set to one in the associated storage key concurrently with the completion of the store access, as observed by the CPU itself. When storing is performed by a CPU or a channel program, the change bit is set to one in the associated storage key either before or after the completion of the store access, as observed by other (if the store was performed by a CPU) or all (if the store was performed by a channel program) CPUs. As observed by other or all CPUs, the change bit is set no earlier than (1) after the

last serialization function performed previously by the CPU or channel program performing the store, and (2) after the execution, by any CPU in the configuration, of a quiescing key-setting instruction that last set the associated storage key before the completion of the store. As observed by other or all CPUs, a change-bit setting necessarily occurs only when any of the following occurs subsequent to the storing operation:

- The CPU or channel program that performed the store performs a serialization function.
- The store was performed by a CPU or a channel program, and any CPU in the configuration sets the subject change bit by executing a key-setting instruction after the store access is completed. The change-bit setting due to the store access occurs before the setting by the key-setting instruction, except that when the nonquiescing key-setting facility is installed, the change bit may appear to be set following a nonquiescing key-setting instruction.
- The store was performed by a CPU and is or will be completed, and any CPU in the configuration executes a COMPARE AND SWAP AND PURGE, INVALIDATE DAT TABLE ENTRY, or INVALIDATE PAGE TABLE ENTRY instruction that clears from the ALB or TLB of the storing CPU any entry used to complete the store. Completion of the clearing instruction is delayed until the subject store and change-bit setting have been completed.
- The store was performed by a CPU, and that CPU examines the subject change bit by means of an INSERT STORAGE KEY EXTENDED or RESET REFERENCE BIT EXTENDED instruction. See "Relation between Operand Accesses" on page 5-98.

When enhanced DAT applies, and either (a) the STE-format control is zero, and the change-recording override is one in the page-table entry used by DAT, or (b) the STE-format control is one, and the change-recording override is one in the segment-table entry used by DAT, it is unpredictable whether the CPU sets the change bit when performing a store operation. The change-recording override may be buffered in the translation-lookaside-buffer copy of the PTE or STE.

3. When the conditional-SSKE facility is not installed, the SET STORAGE KEY EXTENDED instruction causes all seven bits to be set concurrently in the storage key. When the conditional-SSKE facility is installed, SET STORAGE KEY EXTENDED may be used to set all or portions of the storage key based on program-specified criteria. The access to the storage key for SET STORAGE KEY EXTENDED follows the sequence rules for storage-operand store references and is a single-access reference. When the enhanced-DAT facility is installed, SET STORAGE KEY EXTENDED or PERFORM FRAME MANAGEMENT FUNCTION may be used to set all or portions of one or more storage keys based on program-specified criteria.
4. The INSERT STORAGE KEY EXTENDED instruction provides a consistent image of bits 0-6 of the storage key for a 4K-byte block. Similarly, the instructions INSERT VIRTUAL STORAGE KEY and TEST PROTECTION provide a consistent image of the access-control bits and the fetch-protection bit. The access to the storage key for all of these instructions follows the sequence rules for storage-operand fetch references and is a single-access reference.
5. The instruction RESET REFERENCE BIT EXTENDED modifies only the reference bit. All other bits of the storage key remain unchanged. The reference bit and change bit are inspected concurrently to set the condition code. The fetch and store accesses to the storage key for RESET REFERENCE BIT EXTENDED follow the sequence rules for storage-operand fetch and store references, respectively, and are single-access references.
6. For each of the 64 storage keys accessed, the instruction RESET REFERENCE BITS MULTIPLE modifies only the reference bit. All other bits of the storage key remain unchanged. The fetch and store accesses to each of the individual storage keys for RESET REFERENCE BITS MULTIPLE follow the sequence rules for storage-operand fetch and store references, respectively, and are single-access references.

The record of references provided by the reference bit is not necessarily accurate. However, in the majority of situations, reference recording approximately coincides with the related storage reference.

The change bit may be set in cases when no storing has occurred. See "Exceptions to Nullification and Suppression" on page 5-23.

Storage-Operand References

A storage-operand reference is the fetching or storing of the explicit operand or operands in the storage locations designated by the instruction.

During the execution of an instruction, all or some of the storage operands for that instruction may be fetched, intermediate results may be maintained for subsequent modification, and final results may be temporarily held prior to placing them in storage. Stores caused by other CPUs and by channel programs do not necessarily affect these intermediate results.

Storage-operand references are of three types: fetches, stores, and updates.

Storage-Operand Fetch References

When the bytes of a storage operand participate in the instruction execution only as a source, the operand is called a fetch-type operand, and the reference to the location is called a storage-operand fetch reference. A fetch-type operand is identified in individual instruction definitions by indicating that the access exception is for fetch.

All bits within a single byte of a fetch-type operand are accessed concurrently. When an operand consists of more than one byte, the bytes may be fetched from storage piecemeal, one byte at a time. Unless otherwise specified, the bytes are not necessarily fetched in any particular sequence.

The storage-operand fetch references of one instruction occur after those of all preceding instructions and before those of subsequent instructions, as observed by other CPUs and by channel programs. The operands of any one instruction are fetched in the sequence specified for that instruction. The CPU may fetch the operands of instructions before the instructions are executed. There is no defined limit on the length of time between when an operand is fetched and when it is used. Still, as observed by the CPU itself, its storage-operand references are performed in the conceptual sequence.

For certain special instructions, the fetch references for multiple operands may appear to be interlocked

against certain accesses by other CPUs and by channel programs. Such a fetch reference is called an interlocked-fetch reference. The fetch accesses associated with an interlocked-fetch reference do not necessarily occur one immediately after the other, but store accesses by other CPUs may not occur at the same locations as the interlocked-fetch reference between the fetch accesses of the interlocked-fetch reference.

The storage-operand fetch reference for the LOAD PAIR DISJOINT instruction may be an interlocked-fetch reference. Whether or not LOAD PAIR DISJOINT is able to fetch both operands by means of an interlocked fetch is indicated by the condition code.

Storage-Operand Store References

When the bytes of a storage operand participate in the instruction execution only as a destination, to the extent of being replaced by the result, the operand is called a store-type operand, and the reference to the location is called a storage-operand store reference. A store-type operand is identified in individual instruction definitions by indicating that the access exception is for store.

All bits within a single byte of a store-type operand are accessed concurrently. When an operand consists of more than one byte, the bytes may be placed in storage piecemeal, one byte at a time. Unless otherwise specified, the bytes are not necessarily stored in any particular sequence.

The CPU may delay placing results in storage. There is no defined limit on the length of time that results may remain pending before they are stored. This delay does not affect the sequence in which results are placed in storage.

The results of one instruction are placed in storage after the results of all preceding instructions have been placed in storage and before any results of the succeeding instructions are stored, as observed by other CPUs and by channel programs. The results of any one instruction are stored in the sequence specified for that instruction.

The CPU does not fetch operands, ART-table entries, or DAT-table entries from a storage location until all information destined for that location by the CPU has been stored. Prefetched instructions may appear to be updated before the information appears in storage.

The stores are necessarily completed only as a result of a serializing operation and before the CPU enters the stopped state.

Storage-Operand Update References

In some instructions, the storage-operand location participates both as a source and as a destination. In these cases, the reference to the location consists first in a fetch and subsequently in a store. The operand is called an update-type operand, and the combination of the two accesses is referred to as an update reference. Instructions such as MOVE ZONES, TRANSLATE, OR (OC, OI), and ADD DECIMAL cause an update to the first-operand location. An update-type operand is identified in the individual instruction definition by indicating that the access exception is for both fetch and store.

Noninterlocked-Update References: For most instructions which have update-type operands, the fetch and store accesses associated with an update reference do not necessarily occur one immediately after the other, and it is possible for other CPUs and channel programs to make fetch and store accesses to the same location during this time. Such an update reference is sometimes called a noninterlocked-update storage reference.

Interlocked-Update References: For certain special instructions, the update reference is interlocked against certain accesses by other CPUs and channel programs. Such an update reference is called an interlocked-update reference. The fetch and store accesses associated with an interlocked-update reference do not necessarily occur one immediately after the other, but all store accesses by other CPUs and channel programs and the fetch and store accesses associated with interlocked-update references by other CPUs are prevented from occurring at the same location between the fetch and the store accesses of an interlocked-update reference.

The storage-operand update reference for the following instructions appears to be an interlocked-update reference as observed by other CPUs and channel programs.

- ADD IMMEDIATE (ASI and AGSI), when the interlocked-update facility is installed and the first operand is aligned on an integral boundary corresponding to its size
- ADD LOGICAL WITH SIGNED IMMEDIATE, when the interlocked-update facility is installed

and the first operand is aligned on an integral boundary corresponding to its size

- COMPARE AND SWAP
- COMPARE AND SWAP AND PURGE
- COMPARE AND SWAP AND STORE
- COMPARE DOUBLE AND SWAP
- LOAD AND ADD
- LOAD AND ADD LOGICAL
- LOAD AND AND
- LOAD AND EXCLUSIVE OR
- LOAD AND OR
- STORE CHARACTERS UNDER MASK, on models in which the instruction with a mask of zero fetches and stores the byte designated by the second-operand address
- TEST AND SET

Within the limitations of the above requirements, the fetch and store accesses associated with an update reference follow the same rules as the fetches and stores described in the previous sections.

Programming Notes:

1. When two CPUs attempt to update information at a common main-storage location by means of a noninterlocked-update reference, it is possible for both CPUs to fetch the information and subsequently make the store access. The change made by the first CPU to store the result in such a case is lost. Similarly, if one CPU updates the contents of a field by means of a noninterlocked-update reference, but another CPU makes a store access to that field between the fetch and store parts of the update reference, the effect of the store is lost. If, instead of a store access, a CPU makes an interlocked-update reference to the common storage field between the fetch and store portions of a noninterlocked-update reference due to another CPU, any change in the contents produced by the interlocked-update reference is lost.
2. The instructions TEST AND SET, COMPARE AND SWAP, COMPARE AND SWAP AND STORE, and COMPARE DOUBLE AND SWAP facilitate updating of a common storage field by two or more CPUs. To ensure that no changes are lost, all CPUs must use an instruction providing an interlocked-update reference. In addition, the program must ensure that channel programs do not store into the same storage location since such stores may occur between the fetch and

store portions of an interlocked-update reference.

3. Only those bytes which are included in the result field of both operations are considered to be part of the common main-storage location. However, all bits within a common byte are considered to be common even if the bits modified by the two operations do not overlap. As an example, if (1) one CPU executes the instruction OR (OC) with a length of 1 and the value 80 hex in the second-operand location, (2) the other CPU executes AND (NC) with a length of 1 and the value FE hex in the second-operand location, and (3) the first operand of both instructions is the same byte, then the result of one of the updates can be lost.
4. When the store access is part of an update reference by the CPU, the execution of the storing is not necessarily contingent on whether the information to be stored is different from the original contents of the location. In particular, the contents of all designated byte locations are replaced, and, for each byte in the field, the entire contents of the byte are replaced.

Depending on the model, an access to store information may be performed, for example, in the following cases:

- a. Execution of the OR instruction (OI or OC) with a second operand of all zeros.
- b. Execution of OR (OC) with the first-and second-operand fields coinciding.
- c. For those locations of the first operand of TRANSLATE where the argument and function values are the same.

Storage-Operand Consistency

Single-Access References

A fetch reference is said to be a single-access reference if the value is fetched in a single access to each byte of the data field. In the case of overlapping operands, the location may be accessed once for each operand. A store-type reference is said to be a single-access reference if a single store access occurs to each byte location within the data field. An update reference is said to be single access if both the fetch and store accesses are each single access.

Except for the accesses associated with multiple-access references and the stores associated with storage change and restoration for DAT-associated access exceptions, all storage-operand references are single-access references.

Multiple-Access References

In some cases, multiple accesses may be made to all or some of the bytes of a storage operand. The following cases may involve multiple-access references:

1. The storage operands of the following instructions:

- CHECKSUM
- CIPHER MESSAGE
- CIPHER MESSAGE WITH CFB
- CIPHER MESSAGE WITH CHAINING
- CIPHER MESSAGE WITH COUNTER
- CIPHER MESSAGE WITH OFB
- COMPARE AND FORM CODEWORD
- COMPARE UNTIL SUBSTRING EQUAL
- COMPUTE INTERMEDIATE MESSAGE DIGEST
- COMPUTE LAST MESSAGE DIGEST
- COMPUTE MESSAGE AUTHENTICATION CODE
- CONVERT TO BINARY
- CONVERT TO DECIMAL
- CONVERT UTF-16 TO UTF-32
- CONVERT UTF-16 TO UTF-8
- CONVERT UTF-32 TO UTF-16
- CONVERT UTF-32 TO UTF-8
- CONVERT UTF-8 TO UTF-16
- CONVERT UTF-8 TO UTF-32
- LOAD ADDRESS SPACE PARAMETERS
- LOAD REVERSED
- MOVE INVERSE
- MOVE LONG UNICODE
- MOVE PAGE
- MOVE WITH OFFSET
- PACK
- PACK ASCII
- PACK UNICODE
- PERFORM CRYPTOGRAPHIC COMPUTATION
- PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATIONS
- RESUME PROGRAM
- STORE REVERSED
- STORE SYSTEM INFORMATION
- TEST BLOCK
- TRANSLATE

- TRANSLATE AND TEST
- TRANSLATE AND TEST EXTENDED
- TRANSLATE AND TEST REVERSE
- TRANSLATE AND TEST REVERSE EXTENDED
- TRANSLATE EXTENDED
- TRANSLATE ONE TO ONE
- TRANSLATE ONE TO TWO
- TRANSLATE TWO TO ONE
- TRANSLATE TWO TO TWO
- UNPACK
- UNPACK ASCII
- UNPACK UNICODE
- UPDATE TREE

2. The storage operands of MOVE LONG and MOVE LONG EXTENDED, when the padding character is not B1 hex.
3. The stores into that portion of the first operand of MOVE LONG, MOVE LONG EXTENDED, or MOVE LONG UNICODE which is filled with padding bytes.
4. The storage operands of the decimal instructions.
5. The main-storage operands of PAGE IN and PAGE OUT.
6. The storage operands of the I/O instructions.
7. The stores into a trace entry.
8. The stores associated with the stop-and-store-status and store-status-at-address SIGNAL PROCESSOR orders.
9. The trap control block and trap save area used by TRAP.
10. The operands, dictionaries, and symbol-translation table of COMPRESSION CALL.

When a storage-operand store reference to a location is not a single-access reference, the value placed at a byte location is not necessarily the same for each store access; thus, intermediate results in a single-byte location may be observed by other CPUs and by channel programs.

Programming Notes:

1. When multiple fetch or store accesses are made to a single byte that is being changed by another CPU or by a channel program, the result is not necessarily limited to that which could be

obtained by fetching or storing the bits individually. For example, the execution of MULTIPLY DECIMAL may consist in repetitive additions and subtractions, each of which causes the second operand to be fetched from storage and the first operand to be updated in storage.

2. When CPU instructions which make multiple-access references are used to modify storage locations being simultaneously accessed by another CPU or by a channel program, multiple store accesses to a single byte by the CPU may result in intermediate values being observed by the other CPU or by the channel program. To avoid these intermediate values (for example, when modifying a CCW chain), only instructions making single-access references should be used.
3. An instruction fetch, including the fetch of the target of an execute-type instruction, is a multiple-access reference.

Block-Concurrent References

For some references, the accesses to all bytes within a halfword, word, doubleword, or quadword are specified to appear to be block concurrent as observed by other CPUs and channel programs. The halfword, word, doubleword, or quadword is referred to in this section as a block. When a fetch-type reference is specified to appear to be concurrent within a block, no store access to the block by another CPU or channel program is permitted during the time that bytes contained in the block are being fetched. When a store-type reference is specified to appear to be concurrent within a block, no access to the block, either fetch or store, is permitted by another CPU or channel program during the time that the bytes within the block are being stored.

For all instructions in the RX, RXE, RXY, S, SIL, or SIY format, with the exception of CONVERT TO DECIMAL, CONVERT TO BINARY, LOAD PSW EXTENDED, LOAD REVERSED, RESUME PROGRAM, STORE CLOCK EXTENDED, STORE REVERSED, STORE SYSTEM INFORMATION, TRAP, and the I/O instructions, when the operand is addressed on a boundary which is integral to the size of the operand, the storage-operand references appear to be block concurrent as observed by other CPUs. For LOAD PSW EXTENDED, the accesses to each of the two doublewords of the storage operand appear to be doubleword concurrent as observed by other CPUs.

For the instructions COMPARE AND SWAP, COMPARE AND SWAP AND PURGE, COMPARE DOUBLE AND SWAP, LOAD AND ADD, LOAD AND ADD LOGICAL, LOAD AND AND, LOAD AND EXCLUSIVE OR, LOAD AND OR, and LOAD PAIR DISJOINT, all accesses to the storage operand appear to be block concurrent as observed by other CPUs. For COMPARE AND SWAP AND STORE, all accesses to the first operand and the store of the second operand appear to be block concurrent as observed by other CPUs.

For the instruction PERFORM LOCKED OPERATION, the accesses to the even-numbered storage operands appear to be word concurrent, as observed by other CPUs, for function codes that are a multiple of 4 and appear to be doubleword concurrent, as observed by other CPUs, for function codes that are one, 2, or 3 more than a multiple of 4. The accesses to the doublewords in the parameter list appear to be doubleword concurrent, as observed by other CPUs, regardless of the function code.

The instructions LOAD MULTIPLE (LM), LOAD MULTIPLE DISJOINT, LOAD MULTIPLE HIGH, STORE MULTIPLE (STM), and STORE MULTIPLE HIGH, when the operand or operands start on a word boundary; the instructions LOAD MULTIPLE (LMG) and STORE MULTIPLE (STMG), when the operand starts on a doubleword boundary; and the instructions COMPARE LOGICAL (CLC), COMPARE LOGICAL CHARACTERS UNDER MASK, INSERT CHARACTERS UNDER MASK, LOAD CONTROL (LCTLG), STORE CHARACTERS UNDER MASK, and STORE CONTROL (STCTG) access their storage operands in a left-to-right direction, and all bytes accessed within each doubleword appear to be accessed concurrently as observed by other CPUs.

When any operand of EXTRACT CPU TIME starts on a doubleword boundary, all eight bytes within the operand appear to be accessed concurrently as observed by other CPUs.

The instructions LOAD ACCESS MULTIPLE, LOAD CONTROL (LCTL), STORE ACCESS MULTIPLE, and STORE CONTROL (STCTL) access the storage operand in a left-to-right direction, and all bytes accessed within each word appear to be accessed concurrently as observed by other CPUs.

When destructive overlap does not exist, the operands of MOVE (MVC), MOVE WITH KEY, MOVE TO

PRIMARY, and MOVE TO SECONDARY are accessed as follows:

1. The first operand is accessed in a left-to-right direction, and all bytes accessed within a doubleword appear to be accessed concurrently as observed by other CPUs.
2. The second operand is accessed in a left-to-right direction, and all bytes within a doubleword in the second operand that are moved into a single doubleword in the first operand appear to be fetched concurrently as observed by other CPUs. Thus, if the first and second operands begin on the same byte offset within a doubleword, the fetch of the second operand appears to be doubleword concurrent as observed by other CPUs. If the offsets within a doubleword differ by 4, the fetch of the second operand appears to be word concurrent as observed by other CPUs.

Destructive overlap is said to exist when the result location is used as a source after the result has been stored, assuming processing to be performed one byte at a time.

The operands of MOVE WITH SOURCE KEY, MOVE WITH DESTINATION KEY, and MOVE STRING are accessed the same as those of MOVE (MVC), except that destructive overlap is assumed not to exist.

Except as noted in the individual instruction descriptions, accesses to operands of MOVE LONG, MOVE LONG EXTENDED, and MOVE LONG UNICODE do not necessarily appear to occur in a left-to-right direction as observed by other CPUs and by channel programs. The operands of these instructions do appear to be accessed doubleword concurrent, as observed by other CPUs, when all of the following are true:

- Both operands start on doubleword boundaries and are an integral number of doublewords in length.
- The operands do not overlap.
- The nonpadding part of the operation is being executed.

The operands of COMPARE LOGICAL LONG, COMPARE LOGICAL LONG EXTENDED, and COMPARE LOGICAL LONG UNICODE appear to be accessed doubleword concurrent, as observed by other CPUs,

when both operands start on doubleword boundaries and are an integral number of doublewords in length.

The operands of COMPARE LOGICAL STRING appear to be accessed doubleword concurrent, as observed by other CPUs, when both operands start on doubleword boundaries. The operands of SEARCH STRING and SEARCH STRING UNICODE appear to be accessed doubleword concurrent, as observed by other CPUs, when it starts on a doubleword boundary.

For EXCLUSIVE OR (XC), the operands are processed in a left-to-right direction, and, when the first and second operands coincide, all bytes accessed within a doubleword appear to be accessed concurrently as observed by other CPUs.

Programming Note: In the case of EXCLUSIVE OR (XC) designating operands which coincide exactly, the bytes within the field may appear to be accessed as many as three times, by two fetches and one store: once as the fetch portion of the first operand update, once as the second-operand fetch, and then once as the store portion of the first-operand update. Each of the three accesses appears to be doubleword concurrent as observed by other CPUs, but the three accesses do not necessarily appear to occur one immediately after the other. One or both fetch accesses may be omitted since the instruction can be completed without fetching the operands.

Relation between Operand Accesses

As observed by other CPUs and by channel programs, storage-operand fetches associated with one instruction execution appear to precede all storage-operand references for conceptually subsequent instructions. A storage-operand store specified by one instruction appears to precede all storage-operand stores specified by conceptually subsequent instructions, but it does not necessarily precede storage-operand fetches specified by conceptually subsequent instructions. However, a storage-operand store appears to precede a conceptually subsequent storage-operand fetch from the same main-storage location.

When an instruction has two storage operands both of which cause fetch references, it is unpredictable which operand is fetched first, or how much of one

operand is fetched before the other operand is fetched. When the two operands overlap, the common locations may be fetched independently for each operand.

When an instruction has two storage operands the first of which causes a store and the second a fetch reference, it is unpredictable how much of the second operand is fetched before the results are stored. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

When an instruction has two storage operands the first of which causes an update reference and the second a fetch reference, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. Similarly, it is unpredictable how much of the result is processed before it is returned to storage. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

The independent fetching of a single location for each of two operands may affect the program execution in the following situation. When the same storage location is designated by two operand addresses of an instruction, and another CPU or a channel program causes the contents of the location to change during execution of the instruction, the old and new values of the location may be used simultaneously. For example, comparison of a field to itself may yield a result other than equal, or EXCLUSIVE-ORing of a field with itself may yield a result other than zero.

Other Storage References

The restart, program, supervisor-call, external, input/output, and machine-check PSWs appear to be accessed doubleword concurrent as observed by other CPUs. These references appear to occur after the conceptually previous unit of operation and before the conceptually subsequent unit of operation. The relationship between the new-PSW fetch, the old-PSW store, and the interruption-code store is unpredictable.

Store accesses for interruption codes are not necessarily single-access stores. The store accesses for the external and supervisor-call-interruption codes appear to occur between the conceptually previous and conceptually subsequent operations. The store

accesses for the program-interruption codes may precede the storage-operand references associated with the instruction which results in the program interruption.

Relation between Storage-Key Accesses

As observed by other CPUs, storage-key fetches and stores due to instructions that explicitly manipulate a storage key (INSERT STORAGE KEY EXTENDED, INSERT VIRTUAL STORAGE KEY, PERFORM FRAME MANAGEMENT FUNCTION (when SK is one), RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED) are ordered among themselves and among storage-operand references as if the storage-key accesses were themselves storage-operand fetches and stores, respectively.

Accesses of the access-control and fetch-protection bits due to storage-operand references are concurrent with the references. When enhanced DAT applies and a storage-operand reference is made using a segment-table entry in which the STE-format and ACCF-validity controls are both one, the source of the access-control and fetch-protection bits may be any of the following: (a) the storage key for the corresponding 4K-byte block, (b) the access-control bits and the fetch-protection bit in the segment-table entry, or (c) a buffered copy of these bits in the TLB; it is unpredictable which source is used.

Accesses of the reference and change bits due to storage-operand references are in no particular order within the interval in which they are defined to occur. (See the description of when the change bit is set in "Storage-Key Accesses" on page 5-91.) However, whether due to an instruction that explicitly manipulates a storage key or due to a storage-operand reference, a storage-key store appears to precede a conceptually subsequent storage-key fetch from the same storage key.

Serialization

The sequence of functions performed by a CPU is normally independent of the functions performed by other CPUs and by channel programs. Similarly, the sequence of functions performed by a channel pro-

gram is normally independent of the functions performed by other channel programs and by CPUs. However, at certain points in its execution, serialization of the CPU occurs. Serialization also occurs at certain points for channel programs.

CPU Serialization

All interruptions and the execution of certain instructions cause a serialization of CPU operations. A serialization function consists of completing all conceptually previous storage accesses and related reference-bit and change-bit settings by the CPU, as observed by other CPUs and by channel programs, before the conceptually subsequent storage accesses and related reference-bit and change-bit settings occur. A serialization function affects the sequence of all CPU accesses to storage and to the storage keys, except for those associated with ART-table-entry and DAT-table-entry fetching.

A serialization function is performed by CPU reset, all interruptions, and by the execution of the following instructions:

- The general instructions **BRANCH ON CONDITION (BCR)** with the M_1 and R_2 field containing 1111 binary and 0000 binary, respectively.
- When the fast-BCR-serialization facility is installed, the general instruction **BRANCH ON CONDITION (BCR)** with the M_1 and R_2 fields containing 1110 binary and 0000 binary, respectively.
- The general instructions **COMPARE AND SWAP**, **COMPARE AND SWAP AND STORE**, **COMPARE DOUBLE AND SWAP**, **STORE CLOCK**, **STORE CLOCK EXTENDED**, **SUPERVISOR CALL**, and **TEST AND SET**.
- **COMPARE AND SWAP AND PURGE**, which can also cause the ALB and the TLB to be cleared of all entries on all CPUs.
- **INVALIDATE DAT TABLE ENTRY** and **INVALIDATE PAGE TABLE ENTRY**.
- All I/O instructions.
- **LOAD PSW**, **LOAD PSW EXTENDED**, **PERFORM FRAME MANAGEMENT FUNCTION**, and **SET STORAGE KEY EXTENDED**.

- **MOVE TO PRIMARY**, **MOVE TO SECONDARY**, and **SET ADDRESS SPACE CONTROL**.
- **PAGE IN** and **PAGE OUT**.
- **PERFORM LOCKED OPERATION**. A serialization function is performed immediately after the lock is obtained and again immediately before it is released. However, values fetched from the parameter list before the lock is obtained are not necessarily refetched.
- **PROGRAM CALL**, and, when the state entry to be unstacked is a program-call state entry, **PROGRAM RETURN**. However, it is unpredictable whether or not a store into a trace-table entry or linkage-stack entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store.
- **PROGRAM TRANSFER**, **PROGRAM TRANSFER WITH INSTANCE**, **SET SECONDARY ASN**, and **SET SECONDARY ASN WITH INSTANCE**, and **TRACE**. However, it is unpredictable whether or not a store into a trace-table entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store. Additionally, when the store-clock-fast facility is installed and the **TRACE TOD-clock** control in bit 32 of control register 0 is one, it is unpredictable whether explicit tracing causes a serialization function to be performed.
- **PURGE ALB**, **PURGE TLB**, and **SET PREFIX**. **PURGE ALB** and **SET PREFIX** also cause the ALB to be cleared of all entries. **PURGE TLB** and **SET PREFIX** also cause the TLB to be cleared of all entries.
- **SIGNAL PROCESSOR**. The set-architecture **SIGNAL PROCESSOR** order causes serialization on all CPUs in the configuration.
- **TEST BLOCK**.

The four trace functions — branch tracing, ASN tracing, mode tracing, and explicit tracing — cause a serialization function to be performed before the trace action and after completion of the trace action. However, it is unpredictable whether or not a store into a trace-table entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store. Additionally, when the store-clock-fast facility is installed and the **TRACE TOD-clock** control in bit 32 of control register 0 is one, it is

unpredictable whether explicit tracing causes a serialization function to be performed.

In the following discussion, the term *serializing instruction* refers to an instruction which causes one or more serialization functions to be performed. The term *serializing operation* refers to a unit of operation within an instruction or to a machine operation such as an interruption which causes a serialization function to be performed.

The sequence of events associated with a serializing operation is as follows:

1. All conceptually previous storage accesses by the CPU are completed as observed by other CPUs and by channel programs. This includes all conceptually previous stores and changes to the storage keys.
2. The normal function associated with the serializing operation is performed. In the case of instruction execution, operands are fetched, and the storing of results is completed. The exceptions are LOAD PSW, LOAD PSW EXTENDED, and SET PREFIX, in which the operand may be fetched before previous stores have been completed, and interruptions, in which the interruption code and associated fields may be stored prior to the serialization. The fetching of the serializing instruction occurs before the execution of the instruction and may precede the execution of previous instructions, but may not precede the completion of any previous serializing operation. In the case of an interruption, the old PSW, the interruption code, and other information, if any, are stored, and the new PSW is fetched, but not necessarily in that sequence.
3. Finally, instruction fetch and operand accesses for conceptually subsequent operations may begin.

A serializing function affects the sequence of storage accesses that are under the control of the CPU in which the serializing function takes place. It does not affect the sequence of storage accesses under the control of other CPUs and of channel programs.

Programming Notes:

1. The following are some effects of a serializing operation:
 - a. When the execution of an instruction changes the contents of a storage location that is used as a source of a following instruction and when different addresses are used to designate the same absolute location for storing the result and fetching the instruction, a serializing operation following the change ensures that the modified instruction is executed.
 - b. When a serializing operation takes place, other CPUs and channel programs observe instruction and operand fetching and result storing to take place in the sequence established by the serializing operation.
2. Storing into a location from which a serializing instruction is fetched does not necessarily affect the execution of the serializing instruction unless a serializing function has been performed after the storing and before the execution of the serializing instruction.
3. Following is an example showing the effects of serialization. Location A initially contains FF hex.

CPU 1		CPU 2	
MVI	A,X'00'	G	CLI A,X'00'
BCR	15,0		BNE G

The BCR 15,0 instruction executed by CPU 1 is a serializing instruction that ensures that the store by CPU 1 at location A is completed. However, CPU 2 may loop indefinitely, or until the next interruption on CPU 2, because CPU 2 may already have fetched from location A for every execution of the CLI instruction. A serializing instruction must be in the CPU-2 loop to ensure that CPU 2 will again fetch from location A.

Specific-Operand Serialization

Certain instructions cause specific-operand serialization to be performed for an operand of the instruction. As observed by other CPUs and by the channel subsystem, a specific-operand-serialization operation consists in completing all conceptually previous storage accesses by the CPU before a conceptually subsequent accesses to the specific storage operand of the instruction may occur. At the completion of an

instruction causing specific-operand serialization, the instruction's store is completed as observed by other CPUs and channel programs.

Specific-operand serialization is performed by the execution of the following instructions:

- ADD IMMEDIATE (ASI, AGSI) and ADD LOGICAL WITH SIGNED IMMEDIATE, for the first operand, when the interlocked-access facility is installed and the first operand is aligned on a boundary which is integral to the size of the operand.
- LOAD AND ADD, LOAD AND ADD LOGICAL, LOAD AND AND, LOAD AND EXCLUSIVE OR, LOAD AND OR, for the second operand.

Channel-Program Serialization

Serialization of a channel program occurs as follows:

1. All storage accesses and storage-key accesses by the channel program follow initiation of the execution of START SUBCHANNEL, or, if suspended, RESUME SUBCHANNEL, as observed by CPUs and by other channel programs. This includes all accesses for the CCWs, IDAWs, MIDAWs, and data.
2. All storage accesses and storage-key accesses by the channel program are completed, as observed by CPUs and by other channel programs, before the subchannel status indicating status-pending with primary status is made available to any CPU.
3. If a CCW contains a PCI flag or a suspend flag which is one, all storage accesses and storage-key accesses due to CCWs preceding it in the CCW chain are completed, as observed by CPUs and by other channel programs, before the subchannel status indicating status-pending with intermediate status (PCI or suspended) is made available to any CPU.

The serialization of a channel program does not affect the sequence of storage accesses or storage-key accesses performed by other channel programs or by a CPU.

Quiescing

Quiescing is an additional means of serialization that may be performed to ensure that updates to a configuration-wide resource are immediately visible to all CPUs. A quiescing operation initiated by one CPU causes the following to occur on all other CPUs in the configuration:

- Any instruction or unit of operation executing on the CPU is completed.
- All locally-cached copies of the configuration-wide resource being updated are discarded.
- Resumption of instruction execution is delayed until the update to the configuration-wide resource is visible to all CPUs.

Quiescing may occur as a result of the execution of the key-setting instructions, PERFORM FRAME MANAGEMENT FUNCTION (when the set-key control is one) and SET STORAGE KEY EXTENDED. When the nonquiescing key-setting facility is not installed, or when the facility is installed but the NQ control of the instruction is zero, a quiescing operation is performed by the instruction. When the nonquiescing key-setting facility is installed and the NQ control of the instruction is one, the quiescing operation is not performed.

Programming Note: Although quiescing is defined as a additional serialization mechanism, the term *serialization function* (used in the description of CPU instructions and channel operations) is limited to CPU serialization and channel-program serialization.

Monitor-Event Counting

When the execution of a MONITOR CALL instruction causes a monitor event to occur, the enhanced-monitor facility is installed, and the enhanced-monitor-mask bit corresponding to the monitor class is one, a monitor-event counting operation is performed. See "MONITOR CALL" on page 7-231 for further details on the execution of the MONITOR CALL instruction.

The enhanced-monitor counter-array size contained in the word at real location 264, extended on the left with 32 binary zeros, is logically compared with the monitor code formed by the second-operand address. If the monitor code is greater than or equal

to the enhanced-monitor counter-array size, then one is logically added to the enhanced-monitor exception count contained in the word at real location 268, and the instruction completes. Any carry as a result of this addition is ignored.

If the monitor code is less than the enhanced-monitor counter-array size (extended on the left with 32 zeros), then one is added to the enhanced-monitor counter corresponding to the monitor code. An enhanced-monitor counter comprises two discontinuous storage locations: an unsigned halfword which is incremented first, and an unsigned word which is incremented whenever a carry out of bit position 0 of the halfword addition occurs.

The enhanced-monitor counter-array origin in bits 0-60 of the doubleword at real location 256, appended on the right with three binary zeros, forms the 64-bit virtual address of the enhanced-monitor counter array in the home address space.

Although the enhanced-monitor counter array is located in the home address space (as designated by control register 13), the updating of a counter occurs regardless of whether DAT is on. The access to the counters may be performed with the use of the translation-lookaside buffer (TLB), and TLB entries may be formed, regardless of whether DAT is on.

The enhanced-monitor-counter-array-size field at real location 264 specifies the number of halfword and word entries in the array; the halfword entries in the array appear first, followed immediately by the word entries.

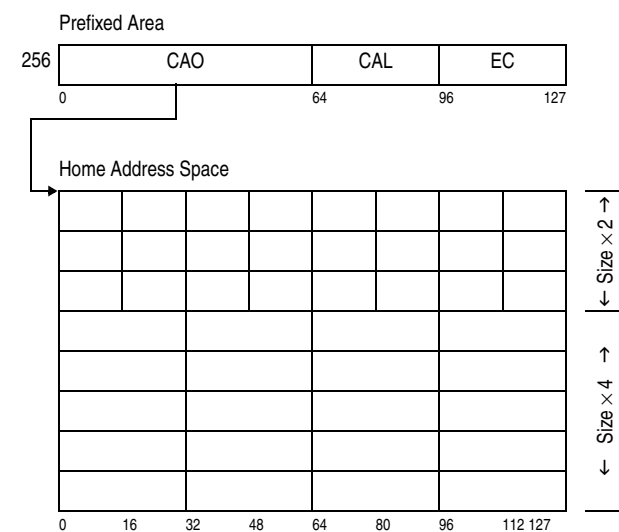
The monitor code multiplied by two is added to the enhanced-monitor counter-array origin to form the virtual address of a halfword to which one is logically added. If this addition results in a carry out of bit position 0 of the halfword, then one is logically added to the corresponding word. The enhanced-monitor-counter-array-size value multiplied by two is added to the monitor code multiplied by four; this value is added to the enhanced-monitor counter-array origin to form the address of the word. Any carry in the word addition is ignored.

Key-controlled protection does not apply to any accesses made in the process of incrementing the counters. Any other access exception recognized in the process of incrementing the counters results in (a) one being logically added to the enhanced-monitor exception count contained in the word at real

location 268, and (b) the instruction completing. Any carry as a result of this addition is ignored.

The word portion of a counter is accessed only if the addition of the corresponding halfword portion of the counter results in an overflow. If the addition of the halfword results in an overflow, the storing of the updated halfword is performed only if both the halfword and word portions of the counter are accessible.

Figure 5-14 illustrates the structures used in the monitor-event-counting operation.



Explanation:

CAO Enhanced-monitor counter-array origin
 CAL Enhanced-monitor counter-array size
 EC Enhanced-monitor exception count

Figure 5-14. Monitor-Event-Counting Structures

Programming Notes:

1. It is recommended that the enhanced-monitor counter array be allocated on a cache-line boundary of the CPU's first-level data cache. The cache line size may be determined by the EXTRACT CACHE ATTRIBUTES (ECAG) instruction.
2. The enhanced-monitor counters may be incremented using a non-interlocked update. Therefore, each enhanced-monitor counter array should be processor unique.
3. It is recommended that the enhanced-monitor-counter-array be allocated in fixed storage such

that dynamic-address-translation (DAT) exceptions do not occur when accessing the array.

Chapter 6. Interruptions

Interruption Action	6-2	Fixed-Point-Divide Exception	6-23
Interruption Code	6-4	Fixed-Point-Overflow Exception	6-24
Enabling and Disabling	6-5	HFP-Divide Exception	6-24
Handling of Floating Interruption Conditions	6-6	HFP-Exponent-Overflow Exception	6-24
Instruction-Length Code	6-6	HFP-Exponent-Underflow Exception	6-24
Zero ILC	6-6	HFP-Significance Exception	6-24
ILC on Instruction-Fetching Exceptions	6-7	HFP-Square-Root Exception	6-24
Exceptions Associated with the PSW	6-8	LFX-Translation Exception	6-25
Early Exception Recognition	6-8	LSTE-Sequence Exception	6-25
Late Exception Recognition	6-9	LSX-Translation Exception	6-25
External Interruption	6-9	LX-Translation Exception	6-25
Clock Comparator	6-10	Monitor Event	6-26
CPU Timer	6-11	Operand Exception	6-27
Emergency Signal	6-11	Operation Exception	6-27
External Call	6-11	Page-Translation Exception	6-27
Interrupt Key	6-11	PC-Translation-Specification Exception	6-28
Malfunction Alert	6-12	PER Event	6-28
Service Signal	6-12	Primary-Authority Exception	6-28
Timing Alert	6-12	Privileged-Operation Exception	6-29
ETR-Timing-Alert Condition	6-12	Protection Exception	6-30
STP-Timing-Alert Condition	6-12	Region-First-Translation Exception	6-30
I/O Interruption	6-13	Region-Second-Translation Exception	6-31
Machine-Check Interruption	6-14	Region-Third-Translation Exception	6-31
Program Interruption	6-14	Secondary-Authority Exception	6-32
Data-Exception Code (DXC)	6-14	Segment-Translation Exception	6-32
Priority of Program Interruptions for Data		Space-Switch Event	6-32
Exceptions	6-15	Special-Operation Exception	6-33
Program-Interruption Conditions	6-17	Specification Exception	6-35
Addressing Exception	6-17	Stack-Empty Exception	6-38
AFX-Translation Exception	6-18	Stack-Full Exception	6-39
ALEN-Translation Exception	6-18	Stack-Operation Exception	6-39
ALE-Sequence Exception	6-19	Stack-Specification Exception	6-39
ALET-Specification Exception	6-19	Stack-Type Exception	6-39
ASCE-Type Exception	6-19	Trace-Table Exception	6-39
ASTE-Instance Exception	6-20	Translation-Specification Exception	6-40
ASTE-Sequence Exception	6-20	Collective Program-Interruption Names	6-40
ASTE-Validity Exception	6-20	Recognition of Access Exceptions	6-41
ASX-Translation Exception	6-21	Multiple Program-Interruption Conditions	6-44
Crypto-Operation Exception	6-21	Access Exceptions	6-47
Data Exception	6-21	ASN-Translation Exceptions	6-49
Decimal-Divide Exception	6-22	Subspace-Replacement Exceptions	6-49
Decimal-Overflow Exception	6-22	Trace Exceptions	6-49
Execute Exception	6-22	Restart Interruption	6-50
EX-Translation Exception	6-23	Supervisor-Call Interruption	6-50
Extended-Authority Exception	6-23	Priority of Interruptions	6-50

The interruption mechanism permits the CPU to change its state as a result of conditions external to the configuration, within the configuration, or within

the CPU itself. To permit fast response to conditions of high priority and immediate recognition of the type of condition, interruption conditions are grouped into

six classes: external, input/output, machine check, program, restart, and supervisor call.

Interruption Action

An interruption consists in storing the current PSW as an old PSW, storing information identifying the cause of the interruption, and fetching a new PSW. Processing resumes as specified by the new PSW.

The old PSW stored on an interruption normally contains the address of the instruction that would have been executed next had the interruption not occurred, thus permitting resumption of the interrupted program. For program and supervisor-call interruptions, the information stored also contains a

code that identifies the length of the last-executed instruction, thus permitting the program to respond to the cause of the interruption. In the case of some program conditions for which the normal response is reexecution of the instruction causing the interruption, the instruction address directly identifies the instruction last executed.

Except for restart, an interruption can occur only when the CPU is in the operating state. The restart interruption can occur with the CPU in either the stopped or operating state.

The details of source identification, location determination, and instruction execution are explained in later sections and are summarized in Figure 6-1 on page 6-2.

Source Identification	Interruption Code		PSW-Mask Bits	Mask Bits in Control Registers		ILC Set	Execution of Instruction Identified by Old PSW
				Reg	Bit		
MACHINE CHECK (old PSW 352, new PSW 480) Exigent condition Repressible condition	Locations 232-239 ¹		13 13			u u	terminated or nullified ² unaffected ²
SUPERVISOR CALL (old PSW 320, new PSW 448) Instruction bits	Locations 138-139 00000000 ssssssss					1, 2, 3	completed
PROGRAM (old PSW 336, new PSW 464)	Locations 142-143						
	Binary	Hex ³					
Operation	00000000 p0000001	0001				1, 2, 3	suppressed
Privileged operation	00000000 p0000010	0002				2, 3	suppressed
Execute	00000000 p0000011	0003				2, 3	suppressed
Protection	00000000 p0000100	0004				1, 2, 3	suppressed or terminated
Addressing	00000000 p0000101	0005				1, 2, 3	suppressed or terminated
Specification	00000000 p0000110	0006				0, 1, 2, 3	suppressed or completed
Data	00000000 p0000111	0007				1, 2, 3	suppressed, terminated or completed
Fixed-point overflow	00000000 p0001000	0008	20			1, 2, 3	completed
Fixed-point divide	00000000 p0001001	0009				1, 2, 3	suppressed or completed
Decimal overflow	00000000 p0001010	000A	21			2, 3	completed
Decimal divide	00000000 p0001011	000B				2, 3	suppressed
HFP exp. overflow	00000000 p0001100	000C				1, 2, 3	completed
HFP exp. underflow	00000000 p0001101	000D	22			1, 2, 3	completed

Figure 6-1. Interruption Action (Part 1 of 3)

Source Identification	Interruption Code			PSW-Mask Bits	Mask Bits in Control Registers		ILC Set	Execution of Instruction Identified by Old PSW
					Reg	Bit		
HFP significance	00000000	p0001110	000E	23			1, 2, 3	completed
HFP divide	00000000	p0001111	000F				1, 2, 3	suppressed
Segment translation	00000000	p0010000	0010				1, 2, 3	nullified
Page translation	00000000	p0010001	0011				1, 2, 3	nullified
Translation spec	00000000	p0010010	0012				1, 2, 3	suppressed
Special operation	00000000	p0010011	0013		0	33	1, 2, 3	suppressed
Operand	00000000	p0010101	0015				2, 3	suppressed
Trace table	00000000	p0010110	0016				1, 2, 3	nullified
Space-switch event	00000000	p0011100	001C		1	57	0, 1, 2, 3	completed
HFP square root	00000000	p0011101	001D				2, 3	suppressed
PC-transl spec	00000000	p0011111	001F				2, 3	suppressed
AFX translation	00000000	p0100000	0020				1, 2, 3	nullified
ASX translation	00000000	p0100001	0021				1, 2, 3	nullified
LX translation	00000000	p0100010	0022				2, 3	nullified
EX translation	00000000	p0100011	0023				2, 3	nullified
Primary authority	00000000	p0100100	0024				2, 3	nullified
Secondary authority	00000000	p0100101	0025				1, 2, 3	nullified
LFX translation	00000000	p0100110	0026				2, 3	nullified
LSX translation	00000000	p0100111	0027				2, 3	nullified
ALET specification	00000000	p0101000	0028				1, 2, 3	suppressed
ALEN translation	00000000	p0101001	0029				1, 2, 3	nullified
ALE sequence	00000000	p0101010	002A				1, 2, 3	nullified
ASTE validity	00000000	p0101011	002B				1, 2, 3	nullified
ASTE sequence	00000000	p0101100	002C				1, 2, 3	nullified
Extended authority	00000000	p0101101	002D				1, 2, 3	nullified
LSTE sequence	00000000	p0101110	002E				2, 3	nullified
ASTE instance	00000000	p0101111	002F				1, 2, 3	nullified
Stack full	00000000	p0110000	0030				2, 3	nullified
Stack empty	00000000	p0110001	0031				1, 2, 3	nullified
Stack specification	00000000	p0110010	0032				1, 2, 3	nullified
Stack type	00000000	p0110011	0033				1, 2, 3	nullified
Stack operation	00000000	p0110100	0034				1, 2, 3	nullified
ASCE type	00000000	p0111000	0038				1, 2, 3	nullified
Region first trans	00000000	p0111001	0039				1, 2, 3	nullified
Region second trans	00000000	p0111010	003A				1, 2, 3	nullified
Region third trans	00000000	p0111011	003B				1, 2, 3	nullified
Monitor event	00000000	p0100000	0040		8	32-47	2, 3	completed
PER basic event	0000000n	1nnnnnn ⁵	0080	1	9	32-36	0, 1, 2, 3	completed ⁶
PER nullification event	00000000	10000000	0080	1	9	33,39	0	nullified ⁷
Crypto operation	00000001	p0011001	0119				2, 3	nullified

Figure 6-1. Interruption Action (Part 2 of 3)

Source Identification	Interruption Code			PSW-Mask Bits	Mask Bits in Control Registers		ILC Set	Execution of Instruction Identified by Old PSW
					Reg	Bit		
EXTERNAL (old PSW 304, new PSW 432) Interrupt key Clock comparator CPU timer Malfunction alert Emergency Signal External call Timing alert Service signal	Location 134-135			7	0	57	u	unaffected
	Binary		Hex ³					
	00000000	01000000	0040					
	00010000	00000100	1004					
	00010000	00000101	1005					
	00010010	00000000	1200					
	00010010	00000001	1201					
	00010010	00000010	1202					
	00010100	00000110	1406					
	00100100	00000001	2401					
INPUT/OUTPUT (old PSW 368, new PSW 496) I/O-interruption subclass	Locations 184-191			6	6	32-39 ⁴	u	unaffected
RESTART (old PSW 288, new PSW 416) Restart key	None						u	unaffected

Explanation:

Locations for the old PSWs, new PSWs, and interruption codes are real locations.

¹ A model-independent machine-check interruption code of 64 bits is stored at real locations 232-239.

² The effect of the machine-check condition is indicated by bits in the machine-check-interruption code. The setting of these bits indicates the extent of the damage and whether the unit of operation is nullified, terminated, or unaffected.

³ The interruption code in the column labeled “Hex” is the hex code for the basic interruption; this code does not show the effects of concurrent interruption conditions represented by n or p in the column labeled “Binary.”

⁴ Bits 32-39 of control register 6 provide detailed masking of I/O-interruption subclasses 0-7 respectively.

⁵ When the interruption code indicates a PER basic event, an ILC of 0 may be stored only when bits 8-15 of the interruption code are 10000110 (PER, specification).

⁶ The unit of operation is completed, unless a program exception concurrently indicated causes the unit of operation to be nullified, suppressed, or terminated.

⁷ The unit of operation is nullified and no other program interruption is indicated when a nullifying PER event is recognized.

n A possible nonzero code indicating another concurrent program-interruption condition

p If one, the bit indicates a concurrent PER-event interruption condition.

s Bits of the I field of SUPERVISOR CALL.

u Not stored.

Figure 6-1. Interruption Action (Part 3 of 3)

Interruption Code

The six classes of interruptions (external, I/O, machine check, program, restart, and supervisor

call) are distinguished by the storage locations at which the old PSW is stored and from which the new PSW is fetched. For most classes, the causes are further identified by an interruption code and, for some classes, by additional information placed in permanently assigned real storage locations during the interruption. (See “Assigned Storage Locations” on page 3-61.) For external, program, and supervisor-call interruptions, the interruption code consists of 16 bits.

For external interruptions, the interruption code is stored at real locations 134-135. A parameter may be stored at real locations 128-131, or a CPU address may be stored at real locations 132-133.

For I/O interruptions, the I/O-interruption code is stored at real locations 184-195. The I/O-interruption code consists of a 32-bit subsystem-identification word, a 32-bit interruption parameter, and a 32-bit I/O-interruption identification word.

For machine-check interruptions, the interruption code consists of 64 bits and is stored at real locations 232-239. Additional information for identifying the cause of the interruption and for recovering the state of the machine may be provided by the contents of the machine-check failing-storage address and the contents of the fixed-logout and machine-check-save areas. (See Chapter 11, “Machine-Check Handling.”)

For program interruptions, the interruption code is stored at real locations 142-143, and the instruction-length code is stored in bit positions 5 and 6 of real location 141. Further information may be provided in the form of the data-exception code (DXC), monitor-class number, PER code, addressing-and-translation-mode identification, PER address, exception access identification, PER access identification, operand-access identification, translation-exception identification, and monitor code, which are stored at real locations 144-162 and 168-183.

Enabling and Disabling

By means of mask bits in the current PSW, floating-point-control (FPC) register, and control registers, the CPU may be enabled or disabled for all external, I/O, and machine-check interruptions and for some program interruptions. When a mask bit is one, the CPU is enabled for the corresponding class of interruptions, and those interruptions can occur.

When a mask bit is zero, the CPU is disabled for the corresponding interruptions. The conditions that cause I/O interruptions remain pending. External-interruption conditions either remain pending or persist until the cause is removed. Machine-check-interruption conditions, depending on the type, are ignored, remain pending, or cause the CPU to enter the check-stop state. The disallowed program-interruption conditions are ignored, except that some causes are indicated also by the setting of the condition code, and IEEE exceptions set flags in the FPC register. The setting of the HFP-significance and HFP-exponent-underflow program-mask bits affects the manner in which HFP operations are completed when the corresponding condition occurs. Similarly, the setting of the IEEE mask bits in the FPC register affects the manner in which IEEE computational operations are completed when the corresponding condition occurs.

Programming Notes:

1. Mask bits in the PSW provide a means of disallowing most maskable interruptions; thus, subsequent interruptions can be disallowed by the new PSW introduced by an interruption. Furthermore, the mask bits can be used to establish a hierarchy of interruption priorities, where a condition in one class can interrupt the program handling a condition in another class but not vice versa. To prevent an interruption-handling routine from being interrupted before the necessary house-keeping steps are performed, the new PSW must disable the CPU for further interruptions within the same class or within a class of lower priority.
2. Because the mask bits in control registers are not changed as part of the interruption procedure, these masks cannot be used to prevent an interruption immediately after a previous interruption in the same class. The mask bits in control registers provide a means for selectively enabling the CPU for some sources and disabling it for others within the same class.
3. Controlling bits exist for several program interruptions, but with no mask bit in the PSW. Such bits include the IEEE mask bits in the FPC register, the monitor masks in bit positions 48-63 of control register 8, and the primary space-switch-event-control bits, bit 57 of control registers 1 and 13, respectively. A bit of this nature is somewhat arbitrarily considered to be a “mask” bit only if the

polarity is such that interruption is enabled when the bit is one.

Thus, for example, the SSM-suppression-control bit, bit 33 of control register 0, is considered to be a mask bit, while the AFP-register-control bit, bit 45 of control register 0, is not. Regardless of the polarity of such control bits, to avoid another program interruption, an interruption-handling routine must avoid issuing instructions subject to these bits until they have been set appropriately.

The enhanced-monitor masks in bit positions 16-31 of control register 8 are an exception to the above definition of a mask in that they work in conjunction with the monitor masks in bit positions 48-63 of control register 8. When both a monitor mask bit and its corresponding enhanced-monitor mask bit are one, a monitor-event counting operation occurs rather than an interruption.

Handling of Floating Interruption Conditions

An interruption condition which can be presented to any CPU in the configuration is called a floating interruption condition. The condition is presented to the first CPU in the configuration which is enabled for the corresponding interruption and which can perform the interruption, and then the condition is cleared and not presented to any other CPU in the configuration. A CPU cannot perform the interruption when it is in the check-stop state, has an invalid prefix, is in a string of program interruptions due to a specification exception of the type which is recognized early or is in the stopped state. However, a CPU with the rate control set to instruction step can perform the interruption when the start key is activated.

Service signal, I/O, and certain machine-check conditions are floating interruption conditions.

Instruction-Length Code

The instruction-length code (ILC) occupies two bit positions and provides the length of the last instruction executed. It permits identifying the instruction causing the interruption when the instruction address in the old PSW designates the next sequential instruction. The ILC is provided also by the BRANCH

AND LINK instructions in the 24-bit addressing mode.

The ILC for program and supervisor-call interruptions is stored in bit positions 5 and 6 of the bytes at real locations 141 and 137, respectively. For external, I/O, machine-check, and restart interruptions, the ILC is not stored since it cannot be related to the length of the last-executed instruction.

For supervisor-call and program interruptions, a non-zero ILC identifies in halfwords the length of the instruction that was last executed. That instruction may be one for which a specification exception was recognized due to an odd instruction address or for which an access exception (addressing, ASCE-type, page-translation, protection, region-translation, segment-translation, or translation-specification) was recognized during the fetching of the instruction. Whenever an instruction is executed by means of EXECUTE, instruction-length code 2 is set to indicate the length of EXECUTE and not that of the target instruction. Similarly, when an instruction is executed by means of EXECUTE RELATIVE LONG, instruction length code 3 is set.

The value of a nonzero instruction-length code is related to the leftmost two bits of the instruction. The value does not depend on whether the operation code is assigned or on whether the instruction is installed. The following table summarizes the meaning of the instruction-length code:

ILC		Instruction Bits 0-1	Instruction Length
Decimal	Binary		
0	00		Not available
1	01	00	One halfword
2	10	01	Two halfwords
2	10	10	Two halfwords
3	11	11	Three halfwords

Zero ILC

Instruction-length code 0, after a program interruption (other than for an instruction-fetching nullification event), indicates that the instruction address stored in the old PSW does not identify the instruction causing the interruption. In the case of a program interruption due to an instruction-fetching nullification event, the ILC is set to zero. The remainder of this section discusses zero ILC due to cases other than for instruction-fetching nullification events.

An ILC of 0 occurs when a specification exception due to a PSW-format error is recognized as part of early exception recognition and the PSW has been introduced by LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, or an interruption. (See “Exceptions Associated with the PSW” on page 6-8.) In the case of LOAD PSW, LOAD PSW EXTENDED, or PROGRAM RETURN, the instruction address of the instruction or of an execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG) has been replaced by the instruction address in the new PSW. When the invalid PSW is introduced by an interruption, the PSW-format error cannot be attributed to an instruction.

In the case of LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, and the supervisor-call interruption, a PER event may be indicated concurrently with a specification exception for which the ILC is 0.

In the case of a PROGRAM RETURN instruction that causes both a space-switch event and a PSW-format error, the space-switch event is recognized, but it is unpredictable whether the ILC is 0 or 1, or 0 or 2 if EXECUTE was used, or 0 or 3 if EXECUTE RELATIVE LONG was used.

ILC on Instruction-Fetching Exceptions

When a program interruption occurs because of an exception that prohibits access to the instruction, the instruction is considered to have been executed without being fetched, and the instruction-length code cannot be set on the basis of the first two bits of the instruction. As far as the significance of the ILC for this case is concerned, the following two situations are distinguished:

1. When an odd instruction address causes a specification exception to be recognized or when an addressing, protection, or translation-specification exception is encountered on fetching an instruction, the ILC is set to 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the instruction address is incremented by 2, 4, or 6. By reducing the instruction address in the old PSW by the number of halfword locations indicated in the ILC, the instruction address originally appearing in the PSW may be obtained.
2. When an ASCE-type, region-translation, segment-translation, or page-translation exception is recognized while fetching an instruction, the ILC

is arbitrarily set to 1, 2, or 3. In this case, the operation is nullified, and the instruction address is not incremented.

The ILC is not necessarily related to the first two bits of the instruction when the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword. The ILC may be arbitrarily set to 1, 2, or 3 in these cases. The instruction address is or is not updated, as described in situations 1 and 2 above.

When any exceptions are encountered on fetching the target instruction of EXECUTE, the ILC is 2. When any exceptions are encountered on fetching the target instruction of EXECUTE RELATIVE LONG, the ILC is 3.

Programming Notes:

1. A nonzero instruction-length code for a program interruption indicates the number of halfword locations by which the instruction address in the program old PSW must be reduced to obtain the instruction address of the last instruction executed, unless one of the following situations exists:
 - a. The interruption is caused by an exception resulting in nullification.
 - b. An interruption for a PER event occurs before the execution of an interruptible instruction is completed, and no other program-interruption condition is indicated concurrently.
 - c. The interruption is caused by a PER event or space-switch event due to LOAD PSW, LOAD PSW EXTENDED, or a branch or linkage instruction, including SUPERVISOR CALL (but not including MONITOR CALL).
 - d. The interruption is caused by an addressing exception or protection exception for the storage operand of a LOAD CONTROL instruction that loads the control register (1 or 13) containing the address-space-control element that specifies the address space from which instructions are fetched.

For situations a and b above, the instruction address in the PSW is not incremented, and the instruction designated by the instruction address is the same as the last one executed. These situ-

ations are the only ones in which the instruction address in the old PSW identifies the instruction causing the exception. Situation b can be distinguished from a PER event indicated after completion of an interruptible or noninterruptible instruction in that, for situation b, the instruction address in the PSW is the same as the PER address in the doubleword at real location 152.

For situation c, the instruction address has been replaced as part of the operation, and the address of the last instruction executed cannot be calculated using the one appearing in the program old PSW.

For situation d, the effective address of the last instruction executed can be calculated, but, since the address-space-control element for the instruction address space is unpredictable, the corresponding real address is unknown.

2. The instruction-length code (ILC) is redundant when a PER event is indicated since the PER address in the doubleword at real location 152 identifies the instruction causing the interruption (or the execute-type instruction, as appropriate). Similarly, the ILC is redundant when the operation is nullified, since in this case the instruction address in the PSW is not incremented. If the ILC value is required in this case, it can be derived from the operation code of the instruction identified by the old PSW.
3. The address of the last instruction executed before a program interruption is insufficient to locate the program problem if one of the following situations exists:
 - a. The interruption is caused by an access exception encountered in fetching an instruction, and the instruction address was introduced into the PSW by a means other than sequential operation (by a branch or linkage instruction, LOAD PSW, LOAD PSW EXTENDED, an interruption, or conclusion of an IPL sequence).
 - b. The interruption is caused by a specification exception due to an odd instruction address, which necessarily also results from introduction of an instruction address into the PSW.
 - c. The interruption is caused by an early specification exception due to a STORE THEN OR SYSTEM MASK or SET SYSTEM MASK instruction that switches to or from the real

mode while introducing invalid values in bit positions 0-7 of the PSW.

For situations a and b, the instruction address was replaced by the operation preceding the last instruction execution, and the address of the program location related to that preceding operation is unavailable.

For situation c, the address of the last instruction executed is available, but the corresponding real address is unknown.

4. The address of the last instruction executed is not available when an interruption is caused by an early specification exception due to a LOAD PSW, LOAD PSW EXTENDED, or PROGRAM RETURN instruction or an interruption.

Exceptions Associated with the PSW

Exceptions associated with erroneous information in the current PSW may be recognized when the information is introduced into the PSW or may be recognized as part of the execution of the next instruction. Errors in the PSW which are specification-exception conditions are called PSW-format errors.

Early Exception Recognition

For the following error conditions, a program interruption for a specification exception occurs immediately after the PSW becomes active:

- Any of the unassigned bits (0, 2-4, 24-30, or 33-63) is a one.
- Bit 12 is a one.
- Bits 31 and 32 are zero and one, respectively, and bits 64-96 are not all zeros.
- Bits 31 and 32 are both zero, and bits 64-103 are not all zeros.
- Bits 31 and 32 are one and zero, respectively.

The interruption occurs regardless of whether the wait state is specified. If the invalid PSW causes the CPU to become enabled for a pending I/O, external, or machine-check interruption, the program interruption occurs instead, and the pending interruption is subject to the mask bits of the new PSW introduced by the program interruption.

When an interruption or the execution of LOAD PSW, LOAD PSW EXTENDED, or PROGRAM RETURN introduces a PSW with one of the above error conditions, the instruction-length code is set to 0, and the newly introduced PSW is stored unmodified as the old PSW. When one of the above error conditions is introduced by execution of SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the instruction-length code is set to 2, and the instruction address is incremented by 4. The PSW containing the invalid value introduced into the system-mask field is stored as the old PSW.

Late Exception Recognition

For the following conditions, the exception is recognized as part of the execution of the next instruction:

- A specification exception is recognized due to an odd instruction address in the PSW (PSW bit 127 is one).
- An access exception (addressing, ASCE-type, page-translation, protection, region-translation, segment-translation, or translation-specification) is associated with the location designated by the instruction address or with the location of the second or third halfword of the instruction starting at the designated instruction address.

The instruction-length code and instruction address stored in the program old PSW under these conditions are discussed in “ILC on Instruction-Fetching Exceptions” on page 6-7, and an example is given in Figure 4-9 on page 4-36.

If an I/O, external, or machine-check-interruption condition is pending and the PSW causes the CPU to be enabled for that condition, the corresponding interruption occurs, and the PSW is not inspected for exceptions which are recognized late. Similarly, a PSW specifying the wait state is not inspected for exceptions which are recognized late.

Programming Notes:

1. The execution of LOAD ADDRESS SPACE PARAMETERS, LOAD PSW, LOAD PSW EXTENDED, PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, RESUME PROGRAM, SET PREFIX, SET SECONDARY ASN, SET SECONDARY ASN WITH INSTANCE, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK

is suppressed on an addressing or protection exception, and hence the program old PSW provides information concerning the program causing the exception.

2. When the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword, the ILC is not necessarily related to the operation code.
3. If the new PSW introduced by an interruption contains a PSW-format error, a string of interruptions may occur. (See “Priority of Interruptions” on page 6-50.)

External Interruption

The external interruption provides a means by which the CPU responds to various signals originating from either inside or outside the configuration.

An external interruption causes the old PSW to be stored at real locations 304-319 and a new PSW to be fetched from real locations 432-447.

The source of the interruption is identified in the interruption code which is stored at real locations 134-135. The instruction-length code is not stored.

Additionally, for the malfunction-alert, emergency-signal, and external-call conditions, a 16-bit CPU address is associated with the source of the interruption and is stored at real locations 132-133. When the CPU address is stored, bit 6 of the interruption code is set to one. For all other conditions, no CPU address is stored, bit 6 of the interruption code is set to zero, and zeros are stored at real locations 132-133.

For the timing-alert and service-signal interruptions, a 32-bit parameter is associated with the interruption and is stored at real locations 128-131. Bit 5 of the external-interruption code indicates that a parameter has been stored. When bit 5 is zero, the contents of real locations 128-131 remain unchanged.

External-interruption conditions are of two types: those for which an interruption-request condition is held pending, and those for which the condition directly requests the interruption. Clock comparator and CPU timer are conditions which directly request external interruptions. If a condition which directly

requests an external interruption is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from a single occurrence of the condition.

When several interruption requests for a single source are generated before the interruption occurs, and the interruption condition is of the type which is held pending, only one request for that source is preserved and remains pending.

An external interruption for a particular source can occur only when the CPU is enabled for interruption by that source. The external interruption occurs at the completion of a unit of operation. The external mask, PSW bit 7, and external subclass-mask bits in control register 0 control whether the CPU is enabled for a particular source. Each source for an external interruption has a subclass-mask bit assigned to it, and the source can cause an interruption only when the external-mask bit is one and the corresponding subclass-mask bit is one.

When the CPU becomes enabled for a pending external-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

More than one source may present a request for an external interruption at the same time. When the CPU becomes enabled for more than one concurrently pending request, the interruption occurs for the pending condition or conditions having the highest priority.

The priorities for external-interruption requests in descending order are as follows:

- Interrupt key
- Malfunction alert
- Emergency signal
- External call
- Clock comparator
- CPU timer
- Timing alert
- Service signal

All requests are honored one at a time. When more than one emergency-signal request exists at a time or when more than one malfunction-alert request exists at a time, the request associated with the smallest CPU address is honored first.

Clock Comparator

When the TOD-clock-steering facility is not installed, an interruption request for the clock comparator exists whenever either of the following conditions is met:

1. The TOD clock is in the set or not-set state, and the value of the clock comparator is less than the value in the compared portion of the TOD clock, both compare values being considered unsigned binary integers.
2. The TOD clock is in the error or not-operational state.

When the TOD-clock-steering facility is installed, an interruption request for the clock comparator exists whenever the physical clock is in the set state and the value of the clock comparator is less than the value in the compared portion of the logical TOD clock, both compare values being considered unsigned binary integers.

When the TOD-clock-steering facility is installed and a CPU remains in the wait state for a long period of time, clock-comparator interruptions may be proportionally late by the maximum steering rate.

If the condition responsible for the request is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from a single occurrence of the condition.

When the TOD-clock-steering facility is not installed, and the TOD clock is set or changes state, interruption conditions, if any, that are due to the clock comparator may or may not be recognized for up to 1.048576 seconds after the change.

When the TOD-clock-steering facility is installed and the physical clock is set or changes state, or the logical TOD clock is changed by PTFF-ATO or PTFF-STO, the CPUs in the configuration do not necessarily recognize this change for purposes of clock comparator interruptions until one of the following instructions is issued: SET CLOCK COMPARATOR (SCKC), STORE CLOCK (STCK), or STORE CLOCK EXTENDED (STCKE). If a CPU is in the wait state when this change occurs, it may not recognize

the change until it leaves the wait state and one of the aforementioned instructions is executed.

The subclass-mask bit is in bit position 52 of control register 0. This bit is initialized to zero.

The clock-comparator condition is indicated by an external-interruption code of 1004 hex.

CPU Timer

An interruption request for the CPU timer exists whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). If the value is made positive before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may occur from a single occurrence of the condition.

When the TOD-clock-steering facility is not installed, and the TOD clock is set or changes state, interruption conditions, if any, that are due to the CPU timer may or may not be recognized for up to 1.048576 seconds after the change.

When the TOD-clock-steering facility is installed, CPU timer interruptions may or may not be recognized while the physical clock is in the stopped state. After the physical clock enters the set state, interruption conditions for the CPU timer are not necessarily recognized until SET CPU TIMER (SPT) is issued. CPU timer interruptions are not affected by changes to the logical TOD clock.

The subclass-mask bit is in bit position 53 of control register 0. This bit is initialized to zero.

The CPU-timer condition is indicated by an external-interruption code of 1005 hex.

Emergency Signal

An interruption request for an emergency signal is generated when the CPU accepts the emergency-signal order specified by a SIGNAL PROCESSOR instruction addressing this CPU. The instruction may have been executed by this CPU or by another CPU in the configuration. The request is preserved and remains pending in the receiving CPU until it is

cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Facilities are provided for holding a separate emergency-signal request pending in the receiving CPU for each CPU in the configuration, including the receiving CPU itself.

The subclass-mask bit is in bit position 49 of control register 0. This bit is initialized to zero.

The emergency-signal condition is indicated by an external-interruption code of 1201 hex. The address of the CPU that executed the SIGNAL PROCESSOR instruction is stored at real locations 132-133.

External Call

An interruption request for an external call is generated when the CPU accepts the external-call order specified by a SIGNAL PROCESSOR instruction addressing this CPU. The instruction may have been executed by this CPU or by another CPU in the configuration. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Only one external-call request, along with the processor address, may be held pending in a CPU at a time.

The subclass-mask bit is in bit position 50 of control register 0. This bit is initialized to zero.

The external-call condition is indicated by an external-interruption code of 1202 hex. The address of the CPU that executed the SIGNAL

PROCESSOR instruction is stored at real locations 132-133.

Interrupt Key

An interruption request for the interrupt key is generated when the operator activates that key. The request is preserved and remains pending in the CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

When the interrupt key is activated while the CPU is in the load state, it depends on the model whether an interruption request is generated or the condition is lost.

The subclass-mask bit is in bit position 57 of control register 0. This bit is initialized to one.

The interrupt-key condition is indicated by an external-interruption code of 0040 hex.

Malfunction Alert

An interruption request for a malfunction alert is generated when another CPU in the configuration enters the check-stop state or loses power. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Facilities are provided for holding a separate malfunction-alert request pending in the receiving CPU for each of the other CPUs in the configuration. Removal of a CPU from the configuration does not generate a malfunction-alert condition.

The subclass-mask bit is in bit position 48 of control register 0. This bit is initialized to zero.

The malfunction-alert condition is indicated by an external-interruption code of 1200 hex. The address of the CPU that generated the condition is stored at real locations 132-133.

Service Signal

An interruption request for a service signal is generated upon the completion of certain configuration-control and maintenance functions, such as those initiated by means of the model-dependent DIAGNOSE instruction. A 32-bit parameter is provided with the interruption to assist the program in determining the operation for which the interruption is reported.

Service signal is a floating interruption condition and is presented to the first CPU in the configuration which can perform the interruption. The interruption condition is cleared when it causes an interruption in any one of the CPUs and also by subsystem reset.

The subclass-mask bit is in bit position 54 of control register 0. This bit is initialized to zero.

The service-signal condition is indicated by an external-interruption code of 2401 hex. A 32-bit parameter is stored at real locations 128-131.

Timing Alert

An interruption request for a timing alert is generated when an ETR-timing-alert or STP-timing-alert condition is detected.

If the same timing-alert condition occurs more than once before the interruption occurs, the request is generated only once. The request is generated for each CPU in the configuration and remains pending at each CPU until it is cleared. The pending request is cleared when it causes an interruption at the CPU and by CPU reset.

The subclass-mask bit is in bit position 59 of control register 0. This bit is initialized to zero.

The timing-alert condition is indicated by an external interruption code of 1406 hex. The interruption parameter stored at real locations 128-131 indicate the reason or reasons for the signal.

ETR-Timing-Alert Condition

An ETR-timing-alert condition is detected when a port-availability change occurs at any port in the current CPC-port group or when an ETR alert occurs. The terms specific to the ETR are not defined in this publication.

STP-Timing-Alert Condition

An STP-timing-alert condition is detected when any of the following occur:

Timing Status Change: A timing-status-change condition is detected when a timing status change has occurred for the configuration. Timing status for a configuration includes the timing mode, timing state and the STP-clock-source state.

A timing-state change from the synchronized state to the unsynchronized or stopped state is reported as an ETR- or STP-sync-check machine-check condition rather than a timing-alert external interruption condition.

An STP-clock-source-state change from the usable state to the not-usable is reported as a clock-source-

error machine-check condition rather than a timing-alert external interruption condition.

Link Availability Change: A link-availability-change condition is detected when the availability of an STP link has changed. The condition is detected whenever an STP link state that was unavailable becomes available for STP communication, or when a link that was available becomes unavailable for STP communication.

A timing-alert condition is not generated by a link-availability-change condition if a CTN configuration-change machine check condition is detected concurrently with the link-availability change.

Time Control Parameter Change: The time control-parameter-change condition is detected when any of the time-control parameters change for the configuration, including when a time-control-parameter change is scheduled. Time-control parameters for a configuration include the following:

- Total-time offset (the combination of time-zone offset and DST offset)
- Leap seconds offset
- Time-zone offset
- Daylight-savings-time (DST) offset
- Scheduled changes to any of the above

A 32-bit parameter is associated with the interruption and is stored at real locations 128-131. The field is defined as shown below:

Bit	Meaning
0-13	Reserved
14	Timing-status change
15	Link-availability change
16	Time-control-parameter change
17-31	Reserved

Multiple alert conditions may be indicated concurrently in the external-interruption parameter field.

I/O Interruption

The input/output (I/O) interruption provides a means by which the CPU responds to conditions originating in I/O devices and the channel subsystem.

A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests are preserved and remain pending until accepted by a CPU, or until cleared by some other means, such as subsystem reset.

The I/O interruption occurs at the completion of a unit of operation. Priority is established among requests so that in each CPU only one interruption request is processed at a time. Priority among requests for interruptions of differing I/O-interruption subclasses is according to the numerical value of the I/O-interruption subclass (with zero having the highest priority), in conjunction with the I/O-interruption subclass-mask settings in control register 6. For more details, see "I/O Interruptions" on page 16-1.

When a CPU becomes enabled for I/O interruptions and the channel subsystem has established priority for a pending I/O-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

An I/O interruption causes the old PSW to be stored at real locations 368-383 and a new PSW to be fetched from real locations 496-511. Additional information, in the form of an eight-byte I/O-interruption code, is stored at real locations 184-195. The I/O-interruption code consists of a 32-bit subsystem-identification word, a 32-bit interruption parameter, and a 32-bit I/O-interruption identification word.

An I/O interruption can occur only while a CPU is enabled for the interruption subclass presenting the request. The I/O-mask bit, bit 6 of the PSW, and the I/O-interruption subclass mask in control register 6 determine whether the CPU is enabled for a particular I/O interruption.

I/O interruptions are grouped into eight I/O-interruption subclasses, numbered from 0-7. Each I/O-interruption subclass has an associated I/O-interruption subclass-mask bit in bit positions 32-39 of control register 6. Each subchannel has an I/O-interruption subclass value associated with it. The CPU is enabled for I/O interruptions of a particular I/O-interruption subclass only when PSW bit 6 is one and the associated I/O-interruption subclass-mask bit in control register 6 is also one. If the corresponding I/O-interruption subclass-mask bit is zero, then the CPU is disabled for I/O interruptions with that subclass value. I/O interruptions for all subclasses are disallowed when PSW bit 6 is zero.

Machine-Check Interruption

The machine-check interruption is a means for reporting to the program the occurrence of equipment malfunctions. Information is provided to assist the program in determining the source of the fault and extent of the damage.

A machine-check interruption causes the old PSW to be stored at real locations 352-367 and a new PSW to be fetched from real locations 480-495.

The cause and severity of the malfunction are identified by a 64-bit machine-check-interruption code stored at real locations 232-239. Further information identifying the cause of the interruption and the location of the fault may be stored at real locations 244-255 and 4608-5119.

The interruption action and the storing of the associated information are under the control of PSW bit 13 and bits in control register 14. See Chapter 11, "Machine-Check Handling" for more detailed information.

Program Interruption

Program interruptions are used to report exceptions and events which occur during execution of the program.

A program interruption causes the old PSW to be stored at real locations 336-351 and a new PSW to be fetched from real locations 464-479.

The cause of the interruption is identified by the interruption code. The interruption code is placed at real locations 142-143, the instruction-length code is placed in bit positions 5 and 6 of the byte at real location 141 with the rest of the bits set to zeros, and zeros are stored at real location 140. For some causes, additional information identifying the reason for the interruption is stored at real locations 144-183.

If the PER-3 facility is installed, then, as part of the program interruption action, the contents of the breaking-event-address register are placed in real storage locations 272-279.

Except for PER events and the crypto-operation exception, the condition causing the interruption is indicated by a coded value placed in the rightmost seven bit positions of the interruption code. Only one condition at a time can be indicated. Bits 0-7 of the interruption code are set to zeros.

PER events are indicated by setting bit 8 of the interruption code to one. When this is the only condition, bits 0-7 and 9-15 are also set to zeros. When a PER event is indicated concurrently with another program-interruption condition, bit 8 is one, and bits 0-7 and 9-15 are set as for the other condition.

The crypto-operation exception is indicated by an interruption code of 0119 hex, or 0199 hex if a PER event is also indicated.

When there is a corresponding mask bit, a program interruption can occur only when that mask bit is one. The program mask in the PSW controls four of the exceptions, the IEEE masks in the FPC register control the IEEE exceptions, bit 33 in control register 0 controls whether SET SYSTEM MASK causes a special-operation exception, bits 48-63 in control register 8 control interruptions due to monitor events, and a hierarchy of masks control interruptions due to PER events. When any controlling mask bit is zero, the condition is ignored; the condition does not remain pending.

Programming Notes:

1. When the new PSW for a program interruption has a PSW-format error or causes an exception to be recognized in the process of instruction fetching, a string of program interruptions may occur. See "Priority of Interruptions" on page 6-50 for a description of how such strings are terminated.
2. Some of the conditions indicated as program exceptions may be recognized also by the channel subsystem, in which case the exception is indicated in the subchannel-status word or extended-status word.

Data-Exception Code (DXC)

When a data exception causes a program interruption, a data-exception code (DXC) is stored at location 147, and zeros are stored at locations 144-146. The DXC distinguishes between the various types of

data-exception conditions. When the AFP-register (additional floating-point register) control bit, bit 45 of control register 0, is one, the DXC is also placed in the DXC field of the floating-point-control (FPC) register. The DXC field in the FPC register remains unchanged when any other program exception is reported. The DXC is an 8-bit code indicating the specific cause of a data exception. The data exceptions and data-exception codes are shown in Figure 6-2 and Figure 6-3 on page 6-15.

Priority of Program Interruptions for Data Exceptions

DXC 2 and 3 are mutually exclusive and are of higher priority than any other DXC. Thus, for example, DXC 2 (BFP instruction) takes precedence over any IEEE exception; and DXC 3 (DFP instruction) takes precedence over any IEEE exception or simulated IEEE exception. As another example, if the conditions for both DXC 3 (DFP instruction) and DXC 1 (AFP register) exist, DXC 3 is reported.

When both a specification exception and an AFP-register data exception apply, it is unpredictable which one is reported.

DXC (Hex)	Data Exception
00	Decimal operand
01	AFP register
02	BFP instruction
03	DFP instruction
04	Quantum Exception
07	Simulated Quantum Exception
08	IEEE inexact and truncated
0B	Simulated IEEE inexact
0C	IEEE inexact and incremented
10	IEEE underflow, exact
13	Simulated IEEE underflow, exact
18	IEEE underflow, inexact and truncated
1B	Simulated IEEE underflow, inexact
1C	IEEE underflow, inexact and incremented
20	IEEE overflow, exact
23	Simulated IEEE overflow, exact
28	IEEE overflow, inexact and truncated
2B	Simulated IEEE overflow, inexact
2C	IEEE overflow, inexact and incremented
40	IEEE division by zero
43	Simulated IEEE division by zero
80	IEEE invalid operation
83	Simulated IEEE invalid operation
FF	Compare-and-trap instruction

Figure 6-2. Data-exception codes (DXC)

Exception	Applicable Instruction Types	CR0.45	FPC Mask	FPC Flag	DXC (Binary)	Instruction Action	DXC Placed in Real Loc. 147	DXC Placed in FPC Byte 2
Decimal operand	Decimal ¹	0	none	none	0000 0000	Suppress or Terminate	Yes	No
		1					Yes	Yes
AFP register	FPS & HFP	0*	none	none	0000 0001	Suppress	Yes	No
BFP instruction	BFP	0*	none	none	0000 0010	Suppress	Yes	No
DFP instruction	DFP	0*	none	none	0000 0011	Suppress	Yes	No
IEEE invalid operation	ICMP	1*	0.0	1.0	1000 0000	Suppress ²	Yes	Yes
IEEE division by zero	ICMP	1*	0.1	1.1	0100 0000	Suppress ²	Yes	Yes
IEEE overflow	ICMP	1*	0.2	1.2	0010 xy00	Complete	Yes	Yes
IEEE underflow	ICMP	1*	0.3	1.3	0001 xy00	Complete	Yes	Yes
IEEE inexact	ICMP	1*	0.4	1.4	0000 1y00	Complete	Yes	Yes

Figure 6-3. Data Exceptions

Exception	Applicable Instruction Types	CR0.45	FPC Mask	FPC Flag	DXC (Binary)	Instruction Action	DXC Placed in Real Loc. 147	DXC Placed in FPC Byte 2
Quantum Exception	ICMP	1*	0.5	1.5	0000 0100	Complete	Yes	Yes
Simulated IEEE invalid operation	IXS	1*	0.0	1.0	1000 0011	Complete	Yes	Yes
Simulated IEEE division by zero	IXS	1*	0.1	1.1	0100 0011	Complete	Yes	Yes
Simulated IEEE overflow	IXS	1*	0.2	1.2	0010 w011	Complete	Yes	Yes
Simulated IEEE underflow	IXS	1*	0.3	1.3	0001 w011	Complete	Yes	Yes
Simulated IEEE inexact	IXS	1*	0.4	1.4	0000 1011	Complete	Yes	Yes
Simulated Quantum Exception	IXS	1*	0.5	1.5	0000 0111	Complete	Yes	Yes
Compare-and-trap instruction	CT	0	none	none	1111 1111	Complete	Yes	No
		1					Yes	Yes

Explanation:

¹ Decimal-operand data exception applies to the decimal instructions (Chapter 8), the general instructions COMPRESSION CALL and CONVERT TO BINARY (Chapter 7), and the DFP instructions CONVERT FROM SIGNED PACKED and CONVERT FROM UNSIGNED PACKED (Chapter 20).

² When the FPC mask bit corresponding to the exception condition is one, the DXC is stored in the FPC register, even though the resulting data-exception program interruption is considered to be suppressing.

0* This exception is recognized only when CR0.45 is zero.

1* This exception is recognized only when CR0.45 is one.

xy For IEEE overflow and IEEE underflow, bits 4 and 5 of the DXC are set to 00, 10, or 11 binary, indicating that the result is exact, inexact and truncated, or inexact and incremented, respectively.

y For IEEE inexact, bit 5 of the DXC is set to zero or one, indicating that the result is inexact and truncated or inexact and incremented, respectively.

BFP Binary-floating-point instructions (Chapter 19).

CT COMPARE AND TRAP, COMPARE IMMEDIATE AND TRAP, COMPARE LOGICAL AND TRAP, or COMPARE LOGICAL IMMEDIATE AND TRAP instructions.

DFP Decimal-floating-point instructions (Chapter 20).

FPS Floating-point-support instructions (Chapter 9).

HFP Hexadecimal-floating-point instructions (Chapter 18).

ICMP IEEE computational instructions.

IXS IEEE-exception-simulation instructions (LOAD FPC AND SIGNAL and SET FPC AND SIGNAL).

Figure 6-3. Data Exceptions (Continued)

Programming Note: The data-exception code (DXC) in bits 16-23 of the floating-point control register (FPCR) is primarily intended for use by floating-point applications that rely on the enablement of the advanced-floating-point (AFP) control, bit 45 of control register 0. When a data-exception program interruption occurs as a result of the execution of a compare-and-trap instruction (DXC 255), the DXC in the FPCR may contain an unpredictable value under the following conditions.

- For a control program, an instruction examining the FPCR (for example, EXTRACT FPC) is the first instruction that is subject to the AFP control to be executed since the last initial-CPU reset.
- For a task running under a control program such as z/OS, an instruction examining the FPCR is

the first instruction that is subject to the AFP control to be executed by the task.

This unpredictability is the result of the control program or task being dispatched with the AFP control initially set to zero, and only applies to DXC 255 in the FPCR. Subsequent inspections of the DXC in the FPCR will yield predictable values.

If the program always needs to observe a predictable compare-and-trap DXC value in the FPCR, it should first issue any instruction that is subject to the AFP control (for example EXTRACT FPC) before issuing an instruction that causes the data-exception program interruption for the compare-and-trap condition. Alternatively, the program can inspect the DXC stored in real location 147, as copied into a control-

program-supplied diagnostic work area; the DXC stored at location 147 is predictable in all cases.

Program-Interruption Conditions

The following is a detailed description of each program-interruption condition.

Addressing Exception

An addressing exception is recognized when the CPU attempts to reference a main-storage location that is not available in the configuration. A main-storage location is not available in the configuration when the location is not installed, when the storage unit is not in the configuration, or when power is off in the storage unit. An address designating a storage location that is not available in the configuration is referred to as invalid.

The operation is suppressed when the address of the instruction is invalid. Similarly, the operation is suppressed when the address of the target instruction of an execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG) is invalid. Also, the unit of operation is suppressed when an addressing exception is encountered in accessing a table or table entry. The tables and table entries to which the rule applies are the dispatchable-unit-control table, the primary ASN-second-table entry, and entries in the access list, region first table, region second table, region third table, segment table, page table, linkage table, linkage-first table, linkage-second table, entry table, ASN first table, ASN second table, authority table, linkage stack, and trace table. Addressing exceptions result in suppression when they are encountered for references to the region first table, region second table, region third table, segment table, and page table, in both implicit references for dynamic address translation and references associated with the execution of LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, STORE REAL ADDRESS, and TEST PROTECTION. Similarly, addressing excep-

tions for accesses to the dispatchable-unit-control table, primary ASN-second-table entry, access list, ASN second table, or authority table result in suppression when they are encountered in access-register translation done either implicitly or as part of LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, STORE REAL ADDRESS, TEST ACCESS, or TEST PROTECTION. Except for some specific instructions whose execution is suppressed, the operation is terminated for an operand address that can be translated but designates an unavailable location. See Figure 6-4 on page 6-17.

For termination, changes may occur only to result fields. In this context, the term “result field” includes the condition code, registers, and any storage locations that are provided and that are designated to be changed by the instruction. Therefore, if an instruction is due to change only the contents of a field in storage, and every byte of the field is in a location that is not available in the configuration, the operation is suppressed. When part of an operand location is available in the configuration and part is not, storing may be performed in the part that is available in the configuration.

When an addressing exception occurs during the fetching of an instruction or during the fetching of a DAT table entry associated with an instruction fetch, it is unpredictable whether the ILC is 1, 2, or 3. When the exception is associated with fetching the target of EXECUTE, the ILC is 2. When the exception is associated with fetching the target of EXECUTE RELATIVE LONG, the ILC is 3.

In all cases of addressing exceptions not associated with instruction fetching, the ILC is 1, 2, or 3, indicating the length of the instruction that caused the reference.

An addressing exception is indicated by a program-interruption code of 0005 hex (or 0085 hex if a concurrent PER event is indicated).

Exception	Action on			
	Table-Entry Fetch ¹	Table-Entry Store ²	Instruction Fetch	Operand Reference
Addressing exception	Suppress	Suppress	Suppress	Suppress for IPTE, LASP, LPSW, LPSWE, MSCH, PLO ⁶ , RP, SCKC, SPT, SPX, SSCH, SSM, STCRW, STNSM, STOSM, TPI, and TPROT Terminate for all others. ⁴

Figure 6-4. Summary of Action for Addressing and Protection Exceptions

Exception	Action on			
	Table-Entry Fetch ¹	Table-Entry Store ²	Instruction Fetch	Operand Reference
Protection exception for key-controlled protection	--	--	Suppress	Suppress for IPTE, LASP, LPSW, LPSWE, MSCH, PLO ⁶ , RP, SCKC, SPT, SPX, SSCH, SSM, STCRW, STNSM, STOSM, and TPI ⁵ . Terminate for all others. ⁴
Protection exception for access-list-controlled protection	--	--	--	Suppress
Protection exception for DAT protection	--	Suppress ³	--	Suppress ⁵
Protection exception for low-address protection	--	Suppress	--	Suppress for IPTE, PLO ⁶ , STCRW, STNSM, STOSM, and TPI ⁵ . Terminate for all others. ⁴
Explanation: -- Not applicable ¹ Table entries include region table, segment table, page table, linkage table, linkage-first table, linkage-second table, entry table, ASN first table, ASN second table, authority table, dispatchable-unit-control table, primary ASN-second-table entry, access list, and linkage stack. ² Table entries include linkage stack and trace table. ³ DAT protection applies to the linkage stack but not the trace table. ⁴ For termination, changes may occur only to result fields. In this context, "result field" includes condition code, registers, and storage locations, if any, which are designated to be changed by the instruction. However, no change is made to a storage location or a storage key when the reference causes an access exception. Therefore, if an instruction is due to change only the contents of a field in main storage, and every byte of that field would cause an access exception, the result is the same as if the operation had been suppressed. The action may be, for key-controlled protection and low-address protection, suppression instead of termination; see "Suppression on Protection" in Chapter 3, "Storage." ⁵ When the effective address of TPI is zero, the store access is to implicit real locations 184-191, and key-controlled protection, DAT protection, and low-address protection do not apply. ⁶ Suppression occurs only for the compare-and-load and compare-and-swap operations.				

Figure 6-4. Summary of Action for Addressing and Protection Exceptions (Continued)

AFX-Translation Exception

An AFX-translation exception is recognized when, during ASN translation in the space-switching form of PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, or SET SECONDARY ASN WITH INSTANCE, or during ASN translation in PROGRAM RETURN when the restored SASN does not equal the restored PASN, bit 0 of the ASN-first-table entry used is not zero.

The ASN being translated is stored at real locations 174 and 175, and real locations 172 and 173 are set with zeros.

The operation is nullified.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The AFX-translation exception is indicated by a program-interruption code of 0020 hex (or 00A0 hex if a concurrent PER event is indicated).

ALEN-Translation Exception

An ALEN-translation exception is recognized during access-register translation when either:

1. The access register used contains an access-list-entry number that designates an access-list entry which is beyond the end of the access list designated by the effective access-list designation.

2. Bit 0 of the access-list entry is not zero.

The number of the access register is stored in bit positions 4-7 of real location 160, and bits 0-3 of the location are set to zeros.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ALEN-translation exception is indicated by a program-interruption code of 0029 hex (or 00A9 hex if a concurrent PER event is indicated).

ALE-Sequence Exception

An ALE-sequence exception is recognized during access-register translation when the access register used contains an access-list-entry sequence number (ALESN) which is not equal to the ALESN in the access-list entry that is designated by the access register.

The number of the access register is stored in bit positions 4-7 at real location 160, and bits 0-3 are set to zeros.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ALE-sequence exception is indicated by a program-interruption code of 002A hex (or 00AA hex if a concurrent PER event is indicated).

ALET-Specification Exception

An ALET-specification exception is recognized during access-register translation when bit positions 0-6 of the access-list-entry token in the access register used do not contain all zeros. However, when access-register 0 is used, except in TEST ACCESS, it is treated as containing all zeros, and this exception is not recognized. TEST ACCESS uses the actual contents of access register 0.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

The ALET-specification exception is indicated by a program-interruption code of 0028 hex (or 00A8 hex if a concurrent PER event is indicated).

ASCE-Type Exception

An ASCE-type exception is recognized when any of the following is true during dynamic address translation:

1. The address-space-control element being used is a region-second-table designation, and bits 0-10 of the virtual address being translated are not all zeros.
2. The address-space-control element being used is a region-third-table designation, and bits 0-21 of the virtual address being translated are not all zeros.
3. The address-space-control element being used is a segment-table designation, and bits 0-32 of the virtual address being translated are not all zeros.

The exception is recognized as part of the execution of the instruction that needs the translation of an instruction or operand address, except for the operand address in LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code.

When an interruption occurs, information about the virtual address causing the exception is stored at real locations 168-175 and conditionally at real locations 160 and 162. See "Assigned Storage Locations" on page 3-61 for a detailed description of this information.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during a reference to the target of EXECUTE, the ILC is 2. When the exception occurs during a reference to the target of EXECUTE RELATIVE LONG, the ILC is 3.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The ASCE-type exception is indicated by a program-interruption code of 0038 hex (or 00B8 hex if a concurrent PER event is indicated).

ASTE-Instance Exception

An ASTE-instance exception is recognized when the ASN-and-LX-reuse facility is installed, and any of the following is true:

1. During ASN translation in PROGRAM TRANSFER WITH INSTANCE with space switching, the ASTEIN in bit positions 0-31 of general register R_1 is not equal to the ASTEIN in the located ASTE.
2. During PASN translation in PROGRAM RETURN with space switching, the ASN-and-LX-reuse control in control register 0 is one and the PASTEIN in the linkage-stack state entry is not equal to the ASTEIN in the located ASTE.
3. During ASN translation in SET SECONDARY ASN WITH INSTANCE with space switching, the ASTEIN in bit positions 0-31 of general register R_1 is not equal to the ASTEIN in the located ASTE.
4. During SASN translation in PROGRAM RETURN to current primary or with space switching, the ASN-and-LX-reuse control in control register 0 is one, and the SASTEIN in the linkage-stack state entry is not equal to the ASTEIN in the located ASTE.

Information is stored as follows:

- In cases 1 and 2, bit 2 of real location 160 is set to one, and bits 0, 1, and 3-7 are set to zeros.
- In cases 3 and 4, bit 3 of real location 160 is set to one, and bits 0-2 and 4-7 are set to zeros.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ASTE-instance exception is indicated by a program-interruption code of 002F hex (or 00AF hex if a concurrent PER event is indicated).

ASTE-Sequence Exception

An ASTE-sequence exception is recognized when any of the following is true:

1. During access-register translation, except as in 2, the access-list entry used contains an ASN-second-table-entry sequence number (ASTESN) which is not equal to the ASTESN in the ASN-second-table entry that is designated by the

access-list entry. The access-list entry is the one designated by the access register used.

2. During access-register translation of ALET 1 by BRANCH IN SUBSPACE GROUP, the subspace ASTESN (SSASTESN) in the dispatchable-unit control table (DUCT) is not equal to the ASTESN in the subspace ASTE designated by the subspace-ASTE origin (SSASTE0) in the DUCT.
3. During a subspace-replacement operation, the subspace ASTESN (SSASTESN) in the dispatchable-unit control table (DUCT) is not equal to the ASTESN in the subspace ASTE designated by the subspace-ASTE origin (SSASTE0) in the DUCT.

In the first and second cases, the number of the access register is stored in bit positions 4-7 at real location 160, and bits 0-3 are set to zeros. In the third case, all zeros are stored at real location 160.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ASTE-sequence exception is indicated by a program-interruption code of 002C hex (or 00AC hex if a concurrent PER event is indicated).

Programming Note: The storing of zeros at real location 160 in the case of an ASTE-sequence exception recognized during a subspace-replacement operation is a unique indication since the use of access register 0 in access-register translation cannot result in the exception.

ASTE-Validity Exception

An ASTE-validity exception is recognized when any of the following is true:

1. During access-register translation, except as in 2, the access-list entry used designates an ASN-second-table entry in which bit 0 is not zero. The access-list entry is the one designated by the access register used.
2. During access-register translation of ALET 1 by BRANCH IN SUBSPACE GROUP, the subspace-ASTE origin (SSASTE0) in the dispatchable-unit control table designates an ASN-second-table entry in which bit 0 is not zero.
3. During a subspace-replacement operation, the subspace-ASTE origin (SSASTE0) in the dis-

patchable-unit control table designates an ASN-second-table entry in which bit 0 is not zero.

In the first and second cases, the number of the access register is stored in bit positions 4-7 at real location 160, and bits 0-3 are set to zeros. In the third case, all zeros are stored at real location 160.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The ASTE-validity exception is indicated by a program-interruption code of 002B hex (or 00AB hex if a concurrent PER event is indicated).

Programming Note: The storing of zeros at real location 160 in the case of an ASTE-validity exception recognized during a subspace-replacement operation is a unique indication since the use of access register 0 in access-register translation cannot result in the exception.

ASX-Translation Exception

An ASX-translation exception is recognized when, during execution of the space-switching form of PROGRAM CALL, during ASN translation in the space-switching form of PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, or SET SECONDARY ASN WITH INSTANCE, or during ASN translation in PROGRAM RETURN when the restored SASN does not equal the restored PASN, bit 0 of the ASN-second-table entry used is not zero.

The ASN being translated is stored at real locations 174 and 175, and real locations 172 and 173 are set with zeros.

The operation is nullified.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The ASX-translation exception is indicated by a program-interruption code of 0021 hex (or 00A1 hex if a concurrent PER event is indicated).

Crypto-Operation Exception

A crypto-operation exception is recognized when a crypto-facility instruction is executed while bit 61 of control register 0 is zero on a CPU which has the

crypto facility installed and available. The crypto-operation exception is also recognized when a crypto-facility instruction is executed and the crypto facility is not installed or available on this CPU, but the facility can be made available to the program either on this CPU or another CPU in the configuration.

When a crypto-facility instruction is executed and the crypto facility is not installed on any CPU which is or can be placed in the configuration, it depends on the model whether a crypto-operation exception or an operation exception is recognized.

The operation is nullified when the crypto-operation exception is recognized.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The crypto-operation exception is indicated by a program-interruption code of 0119 hex (or 0199 hex if a concurrent PER event is indicated).

Data Exception

The data exceptions are shown in Figure 6-3 on page 6-15. A mask bit may or may not control whether an interruption occurs, as noted for each exception.

When a non-maskable data exception is recognized, a program interruption for a data exception always occurs.

Each of the IEEE exceptions is controlled by a mask bit in the floating-point-control (FPC) register. The handling of these exceptions is described in the section "IEEE Exceptions" on page 9-17.

A data exception is recognized for the following cases:

- **Decimal-operand** data exception is recognized when an instruction which operates on decimal operands encounters invalid decimal digit or sign codes or has its operands specified improperly. The operation is suppressed, except that, for EDIT and EDIT AND MARK, it is model dependent whether the operation is suppressed or terminated. See the section "Decimal-Operand Data Exception" on page 8-4 for details. A decimal-operand data exception is also recognized

when COMPRESSION CALL encounters errors in its dictionaries, in which case it is model dependent whether the operation is suppressed or terminated. The decimal-operand data exception is reported with DXC 0.

- **AFP-register** data exception is recognized when bit 45 of control register 0 is zero, and a floating-point-support (FPS) instruction or a hexadecimal-floating-point (HFP) instruction specifies a floating-point register other than 0, 2, 4, or 6. AFP-register data exception is also recognized when bit 45 of control register 0 is zero and a PFPO instruction is executed. The operation is suppressed and is reported with DXC 1.
- **BFP-instruction** data exception is recognized when bit 45 of control register 0 is zero and a BFP instruction is executed. The operation is suppressed and is reported with DXC 2.
- **Compare-and-trap-instruction** data exception is recognized when the operands compared by COMPARE AND TRAP, COMPARE IMMEDIATE AND TRAP, COMPARE LOGICAL AND TRAP, or COMPARE LOGICAL IMMEDIATE AND TRAP match the conditions specified by the M_3 field of the instruction. The operation is completed and is reported with DXC FF hex.
- **DFP-instruction** data exception is recognized when bit 45 of control register 0 is zero and a DFP instruction is executed. The operation is suppressed and is reported with DXC 3.
- **IEEE-exception** data exceptions are recognized when an IEEE computational instruction encounters an enabled exceptional condition. The operation is suppressed or completed, depending on the type of exception. See the section “IEEE Exceptions” on page 19-7 for details.
- **Simulated IEEE-exception** data exceptions are recognized when an IEEE-exception-simulation instruction (LOAD FPC AND SIGNAL or SET FPC AND SIGNAL) encounters an enabled signaling flag. The operation is completed. See the section “IEEE Exceptions” on page 19-7 for details.

The instruction-length code is 1, 2, or 3.

The data exception is indicated by a program-interruption code of 0007 hex (or 0087 hex if a concurrent PER event is indicated).

Decimal-Divide Exception

A decimal-divide exception is recognized when in decimal division the divisor is zero or the quotient exceeds the specified data-field size.

The decimal-divide exception is indicated only if the sign codes of both the divisor and dividend are valid and only if the digit or digits used in establishing the exception are valid.

The operation is suppressed.

The instruction-length code is 2 or 3.

The decimal-divide exception is indicated by a program-interruption code of 000B hex (or 008B hex if a concurrent PER event is indicated).

Decimal-Overflow Exception

A decimal-overflow exception is recognized when one or more nonzero digits are lost because the destination field in a decimal operation is too short to contain the result.

The interruption may be disallowed by the decimal-overflow mask (PSW bit 21).

The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set.

The instruction-length code is 2 or 3.

The decimal-overflow exception is indicated by a program-interruption code of 000A hex (or 008A hex if a concurrent PER event is indicated).

Execute Exception

The execute exception is recognized when the target instruction of an execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG) is another execute-type instruction.

The operation is suppressed.

The instruction-length code is 2 or 3 and indicates the length of the execute-type instruction that attempted to execute the target execute-type instruction.

The execute exception is indicated by a program-interruption code of 0003 hex (or 0083 hex if a concurrent PER event is indicated).

EX-Translation Exception

An EX-translation exception is recognized during PC-number translation in PROGRAM CALL when the entry-table entry designated by the entry-index part of the PC number is beyond the end of the entry table as designated by the linkage-table or linkage-second-table entry used.

When ASN-and-LX reuse is not installed or is not enabled by a one value of the ASN-and-LX-reuse control in control register 0, or if it is installed and enabled and bit 44 of the second-operand address used by PROGRAM CALL is zero, bits 44-63 of the address (a 20-bit PC number), with 12 zeros appended on the left, are stored in the word at real location 172. When ASN-and-LX reuse is installed and enabled and bit 44 of the second-operand address is one, bits 32-63 of the address (a 32-bit PC) are stored in the word at real location 172.

The operation is nullified.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The EX-translation exception is indicated by a program-interruption code of 0023 hex (or 00A3 hex if a concurrent PER event is indicated).

Extended-Authority Exception

An extended-authority exception is recognized during access-register translation when all of the following are true:

1. The private bit in the access-list entry used is one.
2. The access-list-entry authorization index (ALEAX) in the access-list entry is not equal to the extended authorization index (EAX) in control register 8.
3. Either of the following is true:
 - a. The authority-table entry designated by the EAX is beyond the length of the authority table used. The authority table is the one

designated by the ASN-second-table entry that is designated by the access-list entry used.

- b. The secondary-authority bit designated by the EAX is zero.

The access-list entry is the one designated by the access register used.

The number of the access register is stored in bit positions 4-7 at real location 160, and bits 0-3 are set to zeros.

The operation is nullified.

The instruction-length code is 1, 2, or 3.

The extended-authority exception is indicated by a program-interruption code of 002D hex (or 00AD hex if a concurrent PER event is indicated).

Fixed-Point-Divide Exception

A fixed-point-divide exception is recognized when any of the following is true:

1. In signed or unsigned binary division when the result is defined to be 32 bits, the divisor is zero, or the quotient cannot be expressed as a 32-bit signed or unsigned, respectively, binary integer.
2. In signed or unsigned binary division when the result is defined to be 64 bits, the divisor is zero, or the quotient cannot be expressed as a 64-bit signed or unsigned, respectively, binary integer.
3. The result of CONVERT TO BINARY cannot be expressed as a 32-bit signed binary integer for a 32-bit result or as a 64-bit signed binary integer for a 64-bit result.

In the case of division, the operation is suppressed. The execution of CONVERT TO BINARY (CVB, CVBY) is completed by ignoring the leftmost bits that cannot be placed in the register. The execution of CONVERT TO BINARY (CVBG) is suppressed.

The instruction-length code is 1, 2, or 3.

The fixed-point-divide exception is indicated by a program-interruption code of 0009 hex (or 0089 hex if a concurrent PER event is indicated).

Fixed-Point-Overflow Exception

A fixed-point-overflow exception is recognized when an overflow occurs during signed binary arithmetic or signed left-shift operations.

The interruption may be disallowed by the fixed-point-overflow mask (PSW bit 20).

The operation is completed. The result is obtained by ignoring the overflow information, and condition code 3 is set.

The instruction-length code is 1, 2, or 3.

The fixed-point-overflow exception is indicated by a program-interruption code of 0008 hex (or 0088 hex if a concurrent PER event is indicated).

HFP-Divide Exception

An HFP-divide exception is recognized when in HFP division the divisor has a zero fraction.

The operation is suppressed.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The HFP-divide exception is indicated by a program-interruption code of 000F hex (or 008F hex if a concurrent PER event is indicated).

HFP-Exponent-Overflow Exception

An HFP-exponent-overflow exception is recognized when the result characteristic of an HFP operation exceeds 127 and the result fraction is not zero.

The operation is completed. The fraction is normalized, and the sign and fraction of the result remain correct. The result characteristic is made 128 smaller than the correct characteristic.

The instruction-length code is 1, 2, or 3.

The HFP-exponent-overflow exception is indicated by a program-interruption code of 000C hex (or 008C hex if a concurrent PER event is indicated).

HFP-Exponent-Underflow Exception

An HFP-exponent-underflow exception is recognized when the result characteristic of an HFP operation is less than zero and the result fraction is not zero. For an extended-format HFP result, HFP-exponent

underflow is indicated only when the high-order characteristic underflows.

The interruption may be disallowed by the HFP-exponent-underflow mask (PSW bit 22).

The operation is completed. The HFP-exponent-underflow mask also affects the result of the operation. When the mask bit is zero, the sign, characteristic, and fraction are set to zero, making the result a true zero. When the mask bit is one, the fraction is normalized, the characteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct.

The instruction-length code is 1, 2, or 3.

The HFP-exponent-underflow exception is indicated by a program-interruption code of 000D hex (or 008D hex if a concurrent PER event is indicated).

HFP-Significance Exception

An HFP-significance exception is recognized when the result fraction in HFP addition or subtraction is zero.

The interruption may be disallowed by the HFP-significance mask (PSW bit 23).

The operation is completed. The HFP-significance mask also affects the result of the operation. When the mask bit is zero, the operation is completed by replacing the result with a true zero. When the mask bit is one, the operation is completed without further change to the characteristic of the result.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The HFP-significance exception is indicated by a program-interruption code of 000E hex (or 008E hex if a concurrent PER event is indicated).

HFP-Square-Root Exception

An HFP-square-root exception is recognized when the second operand of an HFP SQUARE ROOT instruction is less than zero.

The operation is suppressed.

The instruction-length code is 2 or 3.

The HFP-square-root exception is indicated by a program-interruption code of 001D hex (or 009D hex if a concurrent PER event is indicated).

LFX-Translation Exception

An LFX-translation exception may be recognized only when the ASN-and-LX-reuse facility is installed and the ASN-and-LX-reuse control in control register 0 is one.

An LFX-translation exception is recognized during PC-number translation in PROGRAM CALL when either:

1. The linkage-first-table entry specified by the linkage-first-index part of the PC number is beyond the end of the linkage first table as designated by the linkage-first-table designation used.
2. Bit 0 of the linkage-first-table entry is not zero.

When bit 44 of the second-operand address used by PROGRAM CALL is zero, bits 44-63 of the address (a 20-bit PC number), with 12 zeros appended on the left, are stored in the word at real location 172. When bit 44 of the second-operand address is one, bits 32-63 of the address (a 32-bit PC number) are stored in the word at real location 172.

The operation is nullified.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The LFX-translation exception is indicated by a program-interruption code of 0026 hex (or 00A6 hex if a concurrent PER event is indicated).

LSTE-Sequence Exception

An LSTE-sequence exception may be recognized only when the ASN-and-LX-reuse facility is installed and the ASN-and-LX-reuse control in control register 0 is one.

An LSTE-sequence exception is recognized during PC-number translation in PROGRAM CALL when the linkage-second-table-entry sequence number (LSTESN) in the linkage-second-table entry used is nonzero and not equal to the LSTESN in bit positions 0-31 of general register 15.

When bit 44 of the second-operand address used by PROGRAM CALL is zero, bits 44-63 of the address (a 20-bit PC number), with 12 zeros appended on the left, are stored in the word at real location 172. When bit 44 of the second-operand address is one, bits 32-63 of the address (a 32-bit PC number) are stored in the word at real location 172.

The operation is nullified.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The LSTE-sequence exception is indicated by a program-interruption code of 002E hex (or 00AE hex if a concurrent PER event is indicated).

LSX-Translation Exception

An LSX-translation exception may be recognized only when the ASN-and-LX-reuse facility is installed and the ASN-and-LX-reuse control in control register 0 is one.

An LSX-translation exception is recognized during PC-number translation in PROGRAM CALL when bit 0 of the linkage-second-table entry specified by the linkage-second-index part of the PC number is not zero.

When bit 44 of the second-operand address used by PROGRAM CALL is zero, bits 44-63 of the address (a 20-bit PC number), with 12 zeros appended on the left, are stored in the word at real location 172. When bit 44 of the second-operand address is one, bits 32-63 of the address (a 32-bit PC number) are stored in the word at real location 172.

The operation is nullified.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The LSX-translation exception is indicated by a program-interruption code of 0027 hex (or 00A7 hex if a concurrent PER event is indicated).

LX-Translation Exception

An LX-translation exception may be recognized only when the ASN-and-LX-reuse facility is not installed or the ASN-and-LX-reuse control in control register 0 is zero.

An LX-translation exception is recognized during PC-number translation in PROGRAM CALL when either:

1. The linkage-table entry designated by the linkage-index part of the PC number is beyond the end of the linkage table as designated by the linkage-table designation used.
2. Bit 0 of the linkage-table entry is not zero.

Bits 44-63 of the second-operand address used by PROGRAM CALL (a 20-bit PC number), with 12 zeros appended on the left, are stored in the word at real location 172.

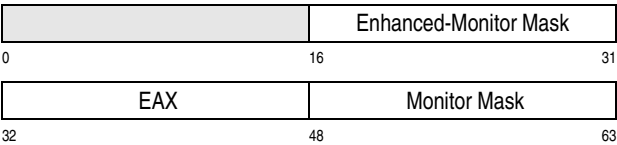
The operation is nullified.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The LX-translation exception is indicated by a program-interruption code of 0022 hex (or 00A2 hex if a concurrent PER event is indicated).

Monitor Event

A monitor event is recognized when MONITOR CALL is executed and the monitor-mask bit in control register 8 corresponding to the monitor class specified by instruction bits 12-15 is one. The information in control register 8 has the following format:



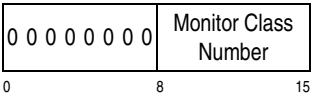
The monitor-mask bits, bits 48-63 of control register 8, correspond to monitor classes 0-15, respectively. Any number of monitor-mask bits may be on at a time; together they specify the classes of monitor events that are monitored at that time. The mask bits are initialized to zeros.

When the enhanced-monitor facility is installed, the enhanced-monitor-mask bits, bits 16-31 of control register 8, correspond to the monitor classes 0-15, respectively. Any number of enhanced-monitor-mask bits may be on at a time; together they specify the classes of monitor events that result in monitor-call program interruptions versus monitor-event counting operations.

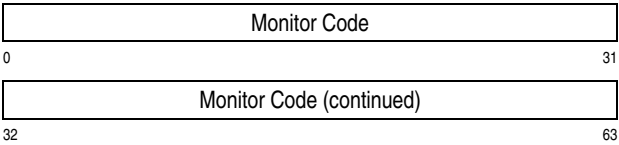
When a monitor event occurs and either (a) the enhanced-monitor facility is not installed, or (b) the facility is installed but the enhanced-monitor-mask bit corresponding to the monitor class specified in bits 12-15 of the MONITOR CALL instruction is zero, then a monitor-event program interruption occurs. When a monitor event occurs, the enhanced-monitor facility is installed, and the enhanced-monitor-mask bit corresponding to the monitor class is one, a monitor-event-counting operation is performed, as described in "Monitor-Event Counting" on page 5-102.

When a monitor-event program interruption occurs, additional information is stored at real locations 148-149 and 176-183. The format of the information stored at these locations is as follows:

Real Locations 148-149



Real Locations 176-183



The contents of bit positions 8-15 of the MONITOR CALL instruction are stored at real location 149 and constitute the monitor-class number. Zeros are stored at real location 148. The effective address specified by the B₁ and D₁ fields of the instruction forms the monitor code, which is stored in the doubleword at real location 176. The value of the address is under control of the addressing mode, bits 31 and 32 of the current PSW. In the 24-bit addressing mode, bits 0-39 of the address are zeros, while in the 31-bit addressing mode, bits 0-32 are zeros.

The operation is completed.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The monitor event is indicated by a program-interruption code of 0040 hex (or 00C0 hex if a concurrent PER event is indicated).

Operand Exception

An operand exception is recognized when any of the following is true:

1. Execution of CLEAR SUBCHANNEL, HALT SUBCHANNEL, MODIFY SUBCHANNEL, RESUME SUBCHANNEL, START SUBCHANNEL, STORE SUBCHANNEL, or TEST SUBCHANNEL is attempted and bit positions 32-47 of general register 1 do not contain 0001 hex. However, an exception due to ones in bit positions 32-39 of the register may or may not be recognized.
2. Execution of MODIFY SUBCHANNEL is attempted, and bits 1 and 6 of word 1 of the SCHIB operand are not zeros or bits 9-10 and 25-30 of word 6 of the SCHIB operand are not all zeros.
3. Execution of MODIFY SUBCHANNEL is attempted, and bits 9 and 10 of word 1 of the SCHIB operand are both one.
4. Execution of RESET CHANNEL PATH is attempted, and bits 40-55 of general register 1 are not all zeros.
5. Execution of SET ADDRESS LIMIT is attempted, and bits 32 and 48-63 of general register 1 are not all zeros.
6. Execution of SET CHANNEL MONITOR is attempted, bit 62 of general register 1 is one, and bits 59-63 of general register 2 are not all zeros.
7. Execution of SET CHANNEL MONITOR is attempted, and bits 36-61 of general register 1 are not all zeros.
8. Execution of START SUBCHANNEL is attempted, and bits 5, 13, and 25-28 of word 1 of the ORB operand are not all zeros.
9. Execution of START SUBCHANNEL is attempted, and bit 11 of word 1 of the ORB operand is not zero. This exception may or may not be recognized.

The operation is suppressed.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The operand exception is indicated by a program-interruption code of 0015 hex (or 0095 hex if a concurrent PER event is indicated).

Operation Exception

An operation exception is recognized when the CPU attempts to execute an instruction with an invalid operation code. The operation code may be unassigned, or the instruction with that operation code may not be installed on the CPU.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

The operation exception is indicated by a program-interruption code of 0001 hex (or 0081 hex if a concurrent PER event is indicated).

Programming Notes:

1. Some models may offer instructions not described in this publication, such as those provided for assists or as part of special or custom features. Consequently, operation codes not described in this publication do not necessarily cause an operation exception to be recognized. Furthermore, these instructions may cause modes of operation to be set up or may otherwise alter the machine so as to affect the execution of subsequent instructions. To avoid causing such an operation, an instruction with an operation code not described in this publication should be executed only when the specific function associated with the operation code is desired.
2. Operation code 00 hex will never be assigned to an instruction implemented in the CPU.

Page-Translation Exception

A page-translation exception is recognized when the page-invalid bit is one.

The exception is recognized as part of the execution of an instruction that needs the page-table entry in the translation of an instruction or operand address, except for the operand address in LOAD REAL ADDRESS and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code, and except for an operand address in MOVE PAGE, in which case the condition is indicated by the setting of the condition code if the condition-code-option bit, bit 55 of general register 0, is one.

When an interruption occurs, information about the virtual address causing the exception is stored at real locations 168-175 and conditionally at real locations 160 and 162. See “Assigned Storage Locations” on page 3-61 for a detailed description of this information.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during a reference to the target of EXECUTE, the ILC is 2. When the exception occurs during a reference to the target of EXECUTE RELATIVE LONG, the ILC is 3.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The page-translation exception is indicated by a program-interruption code of 0011 hex (or 0091 hex if a concurrent PER event is indicated).

PC-Translation-Specification Exception

A PC-translation-specification exception is recognized during PC-number translation in PROGRAM CALL when either of the following is true for the entry-table entry (ETE) used:

1. The PROGRAM CALL operation is the basic operation (bit 128 of the ETE is zero) in the 24-bit or 31-bit addressing mode (bit 31 of the PSW is zero), bit 32 of the ETE is zero (specifying the 24-bit mode), and bits 33-39 of the ETE are not all zeros.
2. The PROGRAM CALL operation is the stacking operation (bit 128 of the ETE is one), bits 32 and 129 of the ETE are zeros (specifying the 24-bit mode), and bits 33-39 of the ETE are not all zeros.

The operation is suppressed.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The PC-translation-specification exception is indicated by a program-interruption code of 001F hex (or 009F hex if a concurrent PER event is indicated).

PER Event

A PER event is recognized when the CPU is enabled for PER and one or more of these events occur.

The PER mask, bit 1 of the PSW, controls whether the CPU is enabled for PER. When the PER mask is zero, PER events are not recognized. When the bit is one, PER events are recognized, subject to the PER-event-mask bits in control register 9.

For a PER instruction-fetching nullification event, the unit of operation is nullified. For other PER events, the unit of operation is completed, unless another condition has caused the unit of operation to be nullified, suppressed, or terminated.

Information identifying the event is stored at real locations 150-159 and conditionally at real location 161.

The instruction-length code is 0, 1, 2, or 3. Code 0 is set for the PER instruction-fetching basic event only if a specification exception is indicated concurrently. Code 0 is always set when the PER instruction-fetching nullification event is indicated.

The PER event is indicated by setting bit 8 of the program-interruption code to one.

See “Program-Event Recording” on page 4-23 for a detailed description of the PER event and the associated interruption information.

Primary-Authority Exception

A primary-authority exception is recognized during ASN authorization in PROGRAM TRANSFER with space switching (PT-ss) or PROGRAM TRANSFER WITH INSTANCE with space switching (PTI-ss) when either:

1. The authority-table entry indicated by the authorization index in control register 4 is beyond the end of the authority table used. The authority table is the one designated by the ASN-second-table entry for the ASN used.
2. The primary-authority bit indicated by the authorization index is zero.

The ASN used is stored at real locations 174-175, and real locations 172-173 are set to zeros.

The operation is nullified.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The primary-authority exception is indicated by a program-interruption code of 0024 hex (or 00A4 hex if a concurrent PER event is indicated).

Privileged-Operation Exception

A privileged-operation exception is recognized when any of the following is true:

1. Execution of a privileged instruction is attempted in the problem state.
2. The value of the rightmost bit of the general register designated by the R_2 field of the PROGRAM TRANSFER or PROGRAM TRANSFER WITH INSTANCE instruction is zero and would cause the PSW problem-state bit to change from the problem state (one) to the supervisor state (zero).
3. In the problem state, the key value specified by the second operand of the SET PSW KEY FROM ADDRESS instruction corresponds to a zero PSW-key-mask bit in control register 3.
4. In the problem state, the key value specified by the rightmost byte of the register designated by the R_3 field of the MOVE WITH KEY instruction corresponds to a zero PSW-key-mask bit in control register 3.
5. In the problem state, the key value specified by the rightmost byte of the register designated by the R_3 field for the instruction MOVE TO PRIMARY, MOVE TO SECONDARY, or MOVE WITH KEY corresponds to a zero PSW-key-mask bit in control register 3.
6. In the problem state, any of the instructions
 - EXTRACT PRIMARY ASN
 - EXTRACT SECONDARY ASN
 - INSERT ADDRESS SPACE CONTROL
 - INSERT PSW KEY
 - INSERT VIRTUAL STORAGE KEY

is encountered, and the extraction-authority control, bit 4 of control register 0, is zero.

7. In the problem state, the result of ANDing the authorization key mask (AKM) with the PSW-key

mask in control register 3 during PROGRAM CALL produces a result of zero.

8. In the problem state, bits 20-23 of the second-operand address of the SET ADDRESS SPACE CONTROL or SET ADDRESS SPACE CONTROL FAST instruction have the value 0011 binary.
9. In the problem state, the key value specified by the rightmost byte of general register 1 for the instruction MOVE WITH SOURCE KEY or MOVE WITH DESTINATION KEY corresponds to a zero PSW-key-mask bit in control register 3.
10. In the problem state, the key value specified by the rightmost byte of the register designated by the R_1 field for the instruction BRANCH AND SET AUTHORITY corresponds to a zero PSW-key-mask bit in control register 3.
11. In the problem state, bits 16 and 17 of the PSW field in the second operand of RESUME PROGRAM have the value 11 binary.
12. Execution of MOVE WITH OPTIONAL SPECIFICATIONS is attempted in the problem state, and any of the following is true:
 - The specified-access-key control for the first operand, bit 46 of general register 0, is one; and the corresponding specified-access key, bits 32-35 of the register, designates a PSW-key-mask (PKM) bit position in control register 3 that contains zero.
 - The specified-access-key control for the second operand, bit 62 of general register 0, is one; and the corresponding specified-access key, bits 48-51 of the register, designates a PKM bit position that contains zero.
 - Either or both bits 46 and 62 of general register 0 are zero, and the PSW key, bits 8-11 of the PSW, designates a PKM bit position that contains zero.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

The privileged-operation exception is indicated by a program-interruption code of 0002 hex (or 0082 hex if a concurrent PER event is indicated).

Protection Exception

A protection exception is recognized when any of the following is true:

Key-Controlled Protection: The CPU attempts to access a storage location that is protected against the type of reference, and the access key does not match the storage key.

Access-List-Controlled Protection: The CPU attempts to store, in the access-register mode, by means of an access-list entry which has the fetch-only bit set to one.

Low-Address Protection: The CPU attempts a store that is subject to low-address protection, the effective address is in the range 0-511 or 4096-4607, and the low-address protection control, bit 35 of control register 0, is one.

DAT Protection: The CPU attempts to store, with DAT on, into any of the following:

- A page which has the DAT-protection bit set to one in the page-table entry used in the translation
- A segment which has the DAT-protection bit set to one in the segment-table entry used in the translation
- When enhanced DAT applies,¹ a region which has the DAT-protection bit set to one in any region-table entry used in the translation.

The operation is suppressed when the location of the instruction is protected against fetching. Similarly, the operation is suppressed when the location of the target instruction of an execute-type instruction is protected against fetching.

For access-list-controlled protection and DAT-protection, the operation is suppressed. For the other two types of protection, except in the case of some specific instructions whose execution is suppressed, the operation is terminated when a protection exception is encountered during a reference to an operand location. See Figure 6-4 on page 6-17. However, the operation may be suppressed as described in "Suppression on Protection" on page 3-14.

For termination, changes may occur only to result fields. In this context, the term "result field" includes

condition code, registers, and storage locations, if any, which are due to be changed by the instruction. However, no change is made to a storage location when a reference to that location causes a protection exception. Therefore, if an instruction is due to change only the contents of a field in storage, and every byte of that field would cause a protection exception, the operation is suppressed. When termination occurs on fetching, the protected information is not loaded into an addressable register nor moved to another storage location.

Information about the address causing the exception is stored at real locations 168-175 and conditionally at real location 160. See "Suppression on Protection" on page 3-14.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2. When the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

For a protected operand location, the instruction-length code (ILC) is 1, 2, or 3, indicating the length of the instruction that caused the reference.

The protection exception is indicated by a program-interruption code of 0004 hex (or 0084 hex if a concurrent PER event is indicated).

Region-First-Translation Exception

A region-first-translation exception is recognized when a region first table is in the translation path for translation of a virtual address and either:

1. The region-first-table entry indicated by the region-first-index portion of the virtual address is outside the region first table.
2. The region-invalid bit is one.

The exception is sometimes called simply a region-translation exception, which term applies also to a region-second-translation exception and a region-third-translation exception.

The exception is recognized as part of the execution of an instruction that needs the region-first-table entry in the translation of an instruction or operand address, except for the operand address in LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL

1. See "Enhanced-DAT Terminology:" on page 3-37 for an explanation of the term "enhanced-DAT applies."

ADDRESS and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code.

When an interruption occurs, information about the virtual address causing the exception is stored at real locations 168-175 and conditionally at real locations 160 and 162. See “Assigned Storage Locations” on page 3-61 for a detailed description of this information.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2. When the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The region-first-translation exception is indicated by a program-interruption code of 0039 hex (or 00B9 hex if a concurrent PER event is indicated).

Region-Second-Translation Exception

A region-second-translation exception is recognized when a region second table is in the translation path for translation of a virtual address and either:

1. The region-second-table entry indicated by the region-second-index portion of the virtual address is outside the region second table.
2. The region-invalid bit is one.

The exception is sometimes called simply a region-translation exception, which term applies also to a region-first-translation exception and a region-third-translation exception.

The exception is recognized as part of the execution of an instruction that needs the region-second-table entry in the translation of an instruction or operand address, except for the operand address in LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code.

When an interruption occurs, information about the virtual address causing the exception is stored at real locations 168-175 and conditionally at real locations 160 and 162. See “Assigned Storage Locations” on page 3-61 for a detailed description of this information.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2. When the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The region-second-translation exception is indicated by a program-interruption code of 003A hex (or 00BA hex if a concurrent PER event is indicated).

Region-Third-Translation Exception

A region-third-translation exception is recognized when a region third table is in the translation path for translation of a virtual address and either:

1. The region-third-table entry indicated by the region-third-index portion of the virtual address is outside the region third table.
2. The region-invalid bit is one.

The exception is sometimes called simply a region-translation exception, which term applies also to a region-first-translation exception and a region-second-translation exception.

The exception is recognized as part of the execution of an instruction that needs the region-third-table entry in the translation of an instruction or operand address, except for the operand address in LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code.

When an interruption occurs, information about the virtual address causing the exception is stored at real locations 168-175 and conditionally at real locations 160 and 162. See “Assigned Storage Locations” on

page 3-61 for a detailed description of this information.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2. When the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The region-third-translation exception is indicated by a program-interruption code of 003B hex (or 00BB hex if a concurrent PER event is indicated).

Secondary-Authority Exception

A secondary-authority exception is recognized during ASN authorization in SET SECONDARY ASN with space switching, SET SECONDARY ASN WITH INSTANCE with space switching, or during ASN authorization in PROGRAM RETURN when the restored SASN does not equal the restored PASN, when either:

1. The authority-table entry indicated by the authorization index in control register 4 is beyond the end of the authority table used. The authority table is the one designated by the ASN-second-table entry for the ASN used. For PROGRAM RETURN, the ASN is the SASN being restored from the linkage-stack state entry used.
2. The secondary-authority bit indicated by the authorization index is zero.

The ASN used is stored at real locations 174-175, and real locations 172-173 are set to zeros.

The operation is nullified.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The secondary-authority exception is indicated by a program-interruption code of 0025 hex (or 00A5 hex if a concurrent PER event is indicated).

Segment-Translation Exception

A segment-translation exception is recognized when either:

1. The segment-table entry indicated by the segment-index portion of a virtual address is outside the segment table.
2. The segment-invalid bit is one.

The exception is recognized as part of the execution of an instruction that needs the segment-table entry in the translation of an instruction or operand address, except for the operand address in LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code.

When an interruption occurs, information about the virtual address causing the exception is stored at real locations 168-175 and conditionally at real locations 160 and 162. See "Assigned Storage Locations" on page 3-61 for a detailed description of this information.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2. When the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The segment-translation exception is indicated by a program-interruption code of 0010 hex (or 0090 hex if a concurrent PER event is indicated).

Space-Switch Event

A space-switch event is recognized at the completion of the operation in each of the following cases:

1. The space-switching form of PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER,

or PROGRAM TRANSFER WITH INSTANCE is executed and any of the following is true:

- a. The primary space-switch-event-control bit, bit 57 of control register 1, is one before the operation.
 - b. The primary space-switch-event-control bit is one after the operation.
 - c. A PER event is indicated.
2. RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST is executed, the CPU is in the home-space mode either before or after the operation, but not both before and after the operation, and any of the following is true:
- a. The primary space-switch-event-control bit, bit 57 of control register 1, is one.
 - b. The home space-switch-event-control bit, bit 57 of control register 13, is one.
 - c. A PER event is indicated.

For PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, and PROGRAM TRANSFER WITH INSTANCE, and for a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes the translation mode to the home-space mode, the old PASN, which is in bit positions 48-63 of control register 4 before the operation, is stored at real locations 174-175, and the old primary space-switch-event-control bit is placed in bit position 0 and zeros are placed in bit positions 1-15 at real locations 172-173.

For a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes the translation mode away from the home-space mode, zeros are stored at real locations 174-175, and the home space-switch-event-control bit is placed in bit position 0 and zeros are placed in bit positions 1-15 at real locations 172-173.

For a PROGRAM RETURN instruction that introduces a PSW-format error, it is unpredictable whether the instruction-length code is 0 or 1, or 0 or 2 if EXECUTE was used, or 0 or 3 if EXECUTE RELATIVE LONG was used.

The operation is completed.

The instruction-length code is 0, 1, or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 0 or 3.

The space-switch event is indicated by a program-interruption code of 001C hex (or 009C hex if a concurrent PER event is indicated).

Programming Notes:

1. The space-switch event permits the control program to gain control whenever a program enters or leaves a particular address space. The primary space-switch-event-control bit is loaded into control register 1, along with the remaining bits of the primary address-space-control element, whenever control register 1 is loaded.
2. The space-switch event may be useful in obtaining programmed authorization checking, in causing additional trace information to be recorded, or in enabling or disabling the CPU for PER or tracing.
3. Bit 121 of the ASN-second-table entry (ASTE) is loaded into bit position 57 of control register 1 as part of the PC-ss, PR-ss, PT-ss, and PTI-ss operations. If bit 121 of the ASTE for a particular address space is set to one, then a space-switch event is recognized when a program enters or leaves the address space by means of any of PC-ss, PR-ss, PT-ss, or PTI-ss.
4. The occurrence of a space-switch event at the completion of a PC-ss, PR-ss, PT-ss, or PTI-ss operation when any PER event is indicated, or at the completion of execution of a RESUME PROGRAM, SET ADDRESS SPACE CONTROL, or SET ADDRESS SPACE CONTROL FAST instruction that changes to or from the home-space mode when any PER event is indicated, permits the control program to determine the address space from which the instruction causing the PER event was fetched.

Special-Operation Exception

A special-operation exception is recognized when any of the following is true:

1. Execution of any of the following instructions is attempted with DAT off:
 - EXTRACT PRIMARY ASN
 - EXTRACT SECONDARY ASN

- INSERT ADDRESS SPACE CONTROL
 - INSERT VIRTUAL STORAGE KEY
 - MOVE WITH OPTIONAL SPECIFICATIONS
 - SET ADDRESS SPACE CONTROL
 - SET SECONDARY ASN
2. Execution of BRANCH AND SET AUTHORITY is attempted, and the R_2 field is zero in the base-authority state or nonzero in the reduced-authority state.
 3. Execution of BRANCH AND STACK, stacking PROGRAM CALL, PROGRAM RETURN, or TRAP is attempted, and the CPU is not in the primary-space or access-register mode.
 4. Execution of BRANCH IN SUBSPACE GROUP is attempted, and any of the following is true:
 - a. The current primary address space is not in a subspace group associated with the current dispatchable unit, that is, the primary-ASTE origin (PASTE0) in control register 5 does not equal the base-ASTE origin (BASTE0) in the dispatchable-unit control table (DUCT).
 - b. The access-list-entry token (ALET) in access register R_2 is ALET 1, but a subspace has not previously been entered by the dispatchable unit by means of BRANCH IN SUBSPACE GROUP, that is, the subspace-ASTE origin (SSASTE0) in the DUCT is all zeros.
 - c. The ALET used is other than ALET 0 and ALET 1, and the destination ASTE (DASTE) does not specify the base space or a subspace of the subspace group, that is, the DASTE origin (DASTE0) obtained from an access-list entry does not equal the BASTE0 in the DUCT, and either the subspace-group bit (G) in the address-space-control element in the DASTE is zero or the base-space bit (B) in the DASTE is one.
 5. Execution of EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE or MODIFY STACKED STATE is attempted, and the CPU is not in the primary-space, access-register, or home-space mode.
 6. Execution of LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), PROGRAM TRANSFER WITH INSTANCE with space switching (PT-ss), SET SECONDARY ASN, or SET SECONDARY ASN WITH INSTANCE is attempted, or execution of a PROGRAM RETURN instruction requiring PASN or SASN translation is attempted, and the ASN-translation control, bit 44 of control register 14, is zero.
 7. Execution of LOAD PAGE-TABLE-ENTRY ADDRESS is attempted and the effective address-space-control element specified by the M_4 field of the instruction designates the real space.
 8. Execution of LOAD REAL ADDRESS (LRA) is attempted in the 24-bit or 31-bit addressing mode, and bits 0-32 of the resulting real or absolute address are not all zeros.
 9. Execution of MOVE TO PRIMARY or MOVE TO SECONDARY is attempted, and the CPU is not in the primary-space or secondary-space mode.
 10. Execution of MOVE TO PRIMARY, MOVE TO SECONDARY, or SET ADDRESS SPACE CONTROL is attempted, and the secondary-space control, bit 37 of control register 0, is zero. The exception may be recognized for this reason when execution of SET ADDRESS SPACE CONTROL FAST is attempted.
 11. Execution of MOVE WITH OPTIONAL SPECIFICATIONS is attempted, and either of the following conditions are true:
 - Bit 47 of general register 0 is one, bits 40-41 of the register contain 11 binary, and the problem-state bit, bit 15 of the current PSW, is one.
 - The secondary-space control, bit 37 of control register 0 is zero, and any of the following conditions are true:
 - Bit 47 of general register 0 is zero, and bits 16-17 of the current PSW are 10 binary.
 - Bit 47 of general register 0 is one, and bits 40-41 of the register are 10 binary.
 - Bit 63 of general register 0 is zero, and bits 16-17 of the current PSW are 10 binary.

- Bit 63 of general register 0 is one, and bits 56-57 of the register are 10 binary.

12. Execution of basic PROGRAM CALL, PROGRAM TRANSFER, or PROGRAM TRANSFER WITH INSTANCE is attempted, and the CPU is not in the primary-space mode.
13. Execution of basic PROGRAM CALL is attempted, and the extended-addressing-mode bit, bit 31 of the current PSW, does not equal the entry-extended-addressing-mode bit, bit 129, in the entry-table entry.
14. Execution of PROGRAM CALL, PROGRAM TRANSFER, or PROGRAM TRANSFER WITH INSTANCE is attempted, and the subsystem-linkage control, bit 192 of the primary ASN-second-table entry, is zero.
15. Execution of the space-switching form of PROGRAM TRANSFER or SET SECONDARY ASN is attempted, the ASN-and-LX-reuse control in control register 0 is one, and the reusable-ASN bit in the located ASTE is one.
16. Execution of the space-switching form of PROGRAM TRANSFER WITH INSTANCE or SET SECONDARY ASN WITH INSTANCE is attempted, the controlled-ASN bit is one in the ASN-second-table entry used, and the CPU is in the problem state at the beginning of the operation.
17. Execution of SET SYSTEM MASK is attempted in the supervisor state, and the SSM-suppression control, bit 33 of control register 0, is one.
18. Execution of TRAP is attempted, and either of the following is true:
 - a. The TRAP-enabled bit, bit 31 in bytes 44-47 of the dispatchable-unit control table, is zero.
 - b. The PSW control, bit 13 of bytes 0-3 of the trap control table, is zero and bit 31 of the current PSW, the extended-addressing-mode bit, is one.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

The special-operation exception is indicated by a program-interruption code of 0013 hex (or 0093 hex if a concurrent PER event is indicated).

Specification Exception

A specification exception is recognized when any of the following is true:

1. A one is introduced into an unassigned bit position of the PSW (that is, any of bit positions 0, 2-4, 24-30, or 33-63). This is handled as an early PSW specification exception.
2. A one is introduced into bit position 12 of the PSW. This is handled as an early PSW specification exception.
3. The PSW is invalid in any of the following ways:
 - a. Bit 31 of the PSW is one and bit 32 is zero.
 - b. Bits 31 and 32 of the PSW are zero, indicating the 24-bit addressing mode, and bits 64-103 of the PSW are not all zeros.
 - c. Bit 31 of the PSW is zero and bit 32 is one, indicating the 31-bit addressing mode, and bits 64-96 of the PSW are not all zeros.

This is handled as an early PSW specification exception. Note, items b and c, above, do not apply to SAM24 and SAM31, respectively. See items 41 and 42 in this list for further explanation.

4. The PSW contains an odd instruction address.
5. An operand address does not designate an integral boundary in an instruction requiring such integral-boundary designation.
6. An odd-numbered general register is designated by an R field of an instruction that requires an even-numbered register designation.
7. A floating-point register other than 0, 1, 4, 5, 8, 9, 12, or 13 is designated for an extended operand.
8. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.
9. The length of the first-operand field is less than or equal to the length of the second-operand field in decimal multiplication or division.
10. Execution of CIPHER MESSAGE, CIPHER MESSAGE WITH CHAINING, COMPUTE INTERMEDIATE MESSAGE DIGEST, COMPUTE LAST MESSAGE DIGEST, or COMPUTE MESSAGE AUTHENTICATION CODE is attempted, and the function code in bits 57-63 of general register 0 contain an unassigned or uninstalled function code.

11. Execution of CIPHER MESSAGE or CIPHER MESSAGE WITH CHAINING is attempted, and the R_1 or R_2 field designates an odd-numbered register or general register 0.
12. Execution of CIPHER MESSAGE, CIPHER MESSAGE WITH CHAINING, COMPUTE INTERMEDIATE MESSAGE DIGEST or COMPUTE MESSAGE AUTHENTICATION CODE is attempted, and the second operand length is not a multiple of the data block size of the designated function. This specification-exception condition does not apply to the query functions.
13. Execution of COMPARE AND FORM CODE-WORD is attempted, and general registers 1, 2, and 3 do not initially contain even values.
14. Execution of COMPARE AND SWAP AND STORE is attempted and any of the following conditions exist:
 - The function code specifies an unassigned value.
 - The store characteristic specifies an unassigned value.
 - The function code is 0, and the first operand is not designated on a word boundary.
 - The function code is 1, and the first operand is not designated on a doubleword boundary.
 - The function code is 2, and any of the following is true:
 - The compare-and-swap-and-store facility 2 is not installed.
 - The first operand is not designated on a quadword boundary.
 - The R_3 field does not designate the even-numbered register of an even-odd register pair.
 - The second operand is not designated on an integral boundary corresponding to the size of the store value.
15. Execution of COMPARE LOGICAL LONG UNICODE or MOVE LONG UNICODE is attempted, and the contents of either general register $R_1 + 1$ or $R_3 + 1$ do not specify an even number of bytes.
16. Execution of COMPARE LOGICAL STRING, MOVE STRING or SEARCH STRING is attempted, and bits 32-55 of general register 0 are not all zeros.
17. Execution of COMPRESSION CALL is attempted, and bits 48-51 of general register 0 have any of the values 0000 and 0110-1111 binary.
18. Execution of COMPUTE INTERMEDIATE MESSAGE DIGEST, COMPUTE LAST MESSAGE DIGEST, or COMPUTE MESSAGE AUTHENTICATION CODE is attempted, and either of the following is true:
 - The R_2 field designates an odd-numbered register or general register 0.
 - Bit 56 of general register 0 is not zero.
19. Execution of CONVERT HFP TO BFP, CONVERT TO FIXED (BFP or HFP), or LOAD FP INTEGER (BFP) is attempted, and the M_3 field does not designate a valid modifier.
20. Execution of DIVIDE TO INTEGER is attempted, and the M_4 field does not designate a valid modifier.
21. Execution of EXECUTE is attempted, and the target address is odd.
22. Execution of EXTRACT STACKED STATE is attempted, and the code in bit positions 56-63 of general register R_2 is greater than 4 when the ASN-and-LX-reuse facility is not installed or is greater than 5 when the facility is installed.
23. Execution of FIND LEFTMOST ONE is attempted, and the R_1 field designates an odd-numbered register.
24. Execution of INVALIDATE DAT TABLE ENTRY is attempted, and bits 44-51 of general register R_2 are not all zeros.
25. Execution of INVALIDATE PAGE TABLE ENTRY is attempted, the IPTE-range facility is installed, the R_3 field is nonzero, and the page index in general register R_2 plus the additional-entry count in general register R_3 is greater than 255.
26. Execution of LOAD FPC or LOAD FPC AND SIGNAL is attempted, and one or more bits of the second operand corresponding to unsupported bits in the FPC register are one.

27. Execution of LOAD PAGE-TABLE-ENTRY ADDRESS is attempted and the M_4 field of the instruction contains any value other than 0000-0100 binary.
28. Execution of LOAD PSW is attempted and bit 12 of the doubleword at the second-operand address is zero. It is model dependent whether or not this exception is recognized.
29. Execution of MONITOR CALL is attempted, and bit positions 8-11 of the instruction do not contain zeros.
30. Execution of MOVE PAGE is attempted, and bit positions 48-51 of general register 0 do not contain zeros or bits 52 and 53 of the register are both one.
31. Execution of PACK ASCII is attempted, and the L_2 field is greater than 31.
32. Execution of PACK UNICODE is attempted, and the L_2 field is greater than 63 or is even.
33. Execution of PERFORM FLOATING POINT OPERATION is attempted, bit 32 of general register 0 is zero, and one or more fields in bits 33-63 are invalid or designate an uninstalled function.
34. Execution of PERFORM FRAME MANAGEMENT FUNCTION is attempted, and either or both of the following are true:
 - Bits 32-45, 52, 55, or 63 of general register R_1 are not zero.
 - The frame-size code in general register R_1 specifies a reserved value.
35. Execution of PERFORM LOCKED OPERATION is attempted, and any of the following is true:
 - The T bit, bit 55 of general register 0 is zero, and the function code in bits 56-63 of the register is invalid.
 - Bits 32-54 of general register 0 are not all zeros.
 - In the access-register mode, for function codes that cause use of a parameter list containing an ALET, the R_3 field is zero.
36. Execution of PERFORM TIMING FACILITY FUNCTION is attempted, and either of the following is true:
 - Bit 56 of general register 0 is not zero.
 - Bits 57-63 of general register 0 specify an unassigned or uninstalled function code.
37. Execution of PROGRAM TRANSFER or PROGRAM TRANSFER WITH INSTANCE is attempted, and all of the following are true:
 - The extended-addressing-mode bit in the PSW is zero.
 - The basic-addressing-mode bit, bit 32, in the general register designated by the R_2 field of the instruction is zero.
 - Bits 33-39 of the instruction address in the same register are not all zeros.
38. Execution of RESUME PROGRAM is attempted, and either of the following is true:
 - Bits 31, 32, and 64-127 of the PSW field in the second operand are not valid for placement in the current PSW. The exception is recognized if any of the following is true:
 - Bits 31 and 32 are both zero and bits 64-103 are not all zeros.
 - Bits 31 and 32 are zero and one, respectively, and bits 64-96 are not all zeros.
 - Bits 31 and 32 are one and zero, respectively.
 - Bit 127 is one.
 - Bits 0-12 of the parameter list are not all zeros.
39. Execution of SEARCH STRING UNICODE is attempted, and bits 32-47 of general register 0 are not all zeros.
40. Execution of SET ADDRESS SPACE CONTROL or SET ADDRESS SPACE CONTROL FAST is attempted, and bits 52 and 53 of the second-operand address are not both zeros.
41. Execution of SET ADDRESSING MODE (SAM24) is attempted, and bits 0-39 of the unupdated instruction address in the PSW, bits 64-103 of the PSW, are not all zeros.
42. Execution of SET ADDRESSING MODE (SAM31) is attempted, and bits 0-32 of the unupdated instruction address in the PSW, bits 64-96 of the PSW, are not all zeros.

43. Execution of SET CLOCK PROGRAMMABLE FIELD is attempted, and bits 32-47 of general register 0 are not all zeros.
44. Execution of SET FPC or SET FPC AND SIGNAL is attempted, and one or more bits of the first operand corresponding to unsupported bits in the FPC register are one.
45. Execution of STORE SYSTEM INFORMATION is attempted, the function code in general register 0 is valid, and either of the following is true:
 - Bits 36-55 of general register 0 and bits 32-47 of general register 1 are not all zeros.
 - The second-operand address is not aligned on a 4K-byte boundary.
46. Execution of TRANSLATE AND TEST EXTENDED or TRANSLATE AND TEST REVERSE EXTENDED is attempted, the A bit, bit 0 of the M_3 field, is one, and the argument-character length in general register $R_1 + 1$ is not an even number.
47. Execution of TRANSLATE TWO TO ONE or TRANSLATE TWO TO TWO is attempted, and the length in general register $R_1 + 1$ does not specify an even number of bytes.
48. Execution of UNPACK ASCII is attempted, and the L_1 field is greater than 31.
49. Execution of UNPACK UNICODE is attempted, and the L_1 field is greater than 63 or is even.
50. Execution of UPDATE TREE is attempted, and the initial contents of general registers 4 and 5 are not a multiple of 8 in the 24-bit or 31-bit addressing mode or are not a multiple of 16 in the 64-bit addressing mode.

The execution of the instruction identified by the old PSW is suppressed. However, for early PSW specification exceptions (causes 1-3) the operation that introduces the new PSW is completed, but an interruption occurs immediately thereafter.

Except as noted below, the instruction-length code (ILC) is 1, 2, or 3, indicating the length of the instruction causing the exception.

When the instruction address is odd (cause 4 on page 6-35), it is unpredictable whether the ILC is 1, 2, or 3.

When the exception is recognized because of an early PSW specification exception (causes 1-3) and the exception has been introduced by LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, or an interruption, the ILC is 0. When the exception is introduced by SET ADDRESSING MODE (SAM24, SAM31), the ILC is 1, or it is 2 if SET ADDRESSING MODE was the target of EXECUTE, or 3 if the SET ADDRESSING MODE was the target of EXECUTE RELATIVE LONG. When the exception is introduced by SET SYSTEM MASK or by STORE THEN OR SYSTEM MASK, the ILC is 2.

The specification exception is indicated by a program-interruption code of 0006 hex (or 0086 hex if a concurrent PER event is indicated).

Programming Note: See “Exceptions Associated with the PSW” on page 6-8 for a definition of when the exceptions associated with the PSW are recognized.

Stack-Empty Exception

A stack-empty exception is recognized during the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN when the current linkage-stack entry is a header entry and the backward stack-entry validity bit in the header entry is zero.

The operation is nullified.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The stack-empty exception is indicated by a program-interruption code of 0031 hex (or 00B1 hex if a concurrent PER event is indicated).

Stack-Full Exception

A stack-full exception is recognized during the stacking process in BRANCH AND STACK or stacking PROGRAM CALL when there is not enough remaining free space in the current linkage-stack section and the forward-section validity bit in the trailer entry of the section is zero.

The operation is nullified.

The instruction-length code is 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The stack-full exception is indicated by a program-interruption code of 0030 hex (or 00B0 hex if a concurrent PER event is indicated).

Stack-Operation Exception

A stack-operation exception is recognized during the unstacking process in PROGRAM RETURN when the unstack-suppression bit is one in any linkage-stack state entry or header entry encountered during the process.

The operation is nullified.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The stack-operation exception is indicated by a program-interruption code of 0034 hex (or 00B4 hex if a concurrent PER event is indicated).

Stack-Specification Exception

A stack-specification exception is recognized in each of the following cases:

1. During the stacking process in BRANCH AND STACK or stacking PROGRAM CALL when there is not enough remaining free space in the current linkage-stack section and either of the following is true:
 - a. The remaining-free-space value used to locate the trailer entry of the current section is not a multiple of 8.
 - b. There is not enough remaining free space in the next section.
2. During the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN when the current linkage-stack entry is a header entry in which the backward stack-entry address designates another header entry.

The operation is nullified.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The stack-specification exception is indicated by a program-interruption code of 0032 hex (or 00B2 hex if a concurrent PER event is indicated).

Stack-Type Exception

A stack-type exception is recognized during the unstacking process in EXTRACT STACKED REGISTERS, EXTRACT STACKED STATE, MODIFY STACKED STATE, or PROGRAM RETURN in each of the following cases:

1. The current linkage-stack entry is not a header entry or a state entry.
2. When the current linkage-stack entry is a header entry, the preceding entry, designated by the backward stack-entry address in the header entry, is not a header entry or a state entry. (A stack-specification exception is recognized if the preceding entry is a header entry.)

The operation is nullified.

The instruction-length code is 1 or 2, except that when the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

The stack-type exception is indicated by a program-interruption code of 0033 hex (or 00B3 hex if a concurrent PER event is indicated).

Trace-Table Exception

A trace-table exception is recognized when the CPU attempts to store a trace-table entry which would reach or cross the next 4K-byte block boundary. For the purpose of recognizing this exception in the TRACE instruction, the explicit trace entry is treated as being 76 bytes long for TRACE (TRACE) and as 144 bytes long for TRACE (TRACG). For a PROGRAM CALL instruction that would cause storing of both a PROGRAM CALL trace entry and a mode-switch trace entry, the exception is recognized for the first entry when either the first or the second entry would reach or cross the boundary.

The operation is nullified.

The instruction-length code is 1, 2, or 3, indicating the length of the instruction causing the exception.

The trace-table exception is indicated by a program-interruption code of 0016 hex (or 0096 hex if a concurrent PER event is indicated).

Translation-Specification Exception

A translation-specification exception is recognized when translation of a virtual address is attempted and any of the following is true:

1. In the lookup in the table designated by the address-space-control element used for the translation, the table-type bits in the selected table entry do not equal the designation-type bits in the address-space-control element.
2. In a lookup in a table designated by an entry in a region first table, region second table, or region third table, the value of the table-type bits in the selected table entry is not one less than the value of the same bits in the designating table entry.
3. The private-space control, bit 55 in the address-space-control element used for the translation, is one, the segment-table entry used for the translation is valid, and the common-segment bit, bit 59, in the segment-table entry is one.
4. The page-table entry used for the translation is valid, and bit position 52 does not contain zero.
5. The page-table entry used for the translation is valid, enhanced DAT does not apply, and bit position 55 does not contain zero.

Any of the above reasons is referred to by saying that the DAT-table entry has a format error.

The exception is recognized only as part of the execution of an instruction using address translation, that is, when DAT is on and a logical address, instruction address, or virtual address must be translated, or when LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS or STORE REAL ADDRESS is executed.

The unit of operation is suppressed.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2. When the exception occurs during the fetching of the target of EXECUTE RELATIVE LONG, the ILC is 3.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The translation-specification exception is indicated by a program-interruption code of 0012 hex (or 0092 hex if a concurrent PER event is indicated).

Programming Note: When a translation-specification exception is recognized in the process of translating an instruction address, the operation is suppressed. In this case, the instruction-length code (ILC) is needed to derive the address of the instruction, as the instruction address in the old PSW has been incremented by the amount indicated by the ILC. In the case of region-first-translation, region-second-translation, region-third-translation, segment-translation, and page-translation exceptions, the operation is nullified, the instruction address in the old PSW identifies the instruction, and the ILC may be arbitrarily set to 1, 2, or 3.

Collective Program-Interruption Names

For the sake of convenience, certain program exceptions are grouped together under a single collective name. These collective names are used when it is necessary to refer to the complete set of exceptions, such as in instruction definitions. Four collective names are used:

- Access exceptions
- ASN-translation exceptions
- Subspace-replacement exceptions
- Trace exceptions

The individual exceptions and their priorities are listed in “Multiple Program-Interruption Conditions” on page 6-44.

Recognition of Access Exceptions

Figure 6-5 on page 6-41 summarizes the conditions that can cause access exceptions and the action taken when they are encountered.

Any access exception is recognized as part of the execution of the instruction with which the exception is associated. An access exception is not recognized when the CPU attempts to prefetch from an unavail-

Condition ²	Translation for Virtual Address of LPTEA		Translation for Virtual Address of LOAD REAL ADDRESS		Translation for TAR and TPROT, and Access for Logical Address of TPROT ¹		Translation and Access for Any Other Address	
	Indi-cation	Action	Indi-cation	Action	Indi-cation	Action	Indi-cation	Action
<u>Access register</u> ³ Bits 0-6 not all zeros	cc3	Complete	cc3	Complete	cc3	Complete	AS	Suppress
<u>Effective access-list designation</u> Invalid address of designation	A	Suppress	A	Suppress	A	Suppress	A	Suppress
<u>Access-list entry</u> ³ Access-list-length violation	cc3	Complete	cc3	Complete	cc3	Complete	AT	Nullify
Invalid address of entry	A	Suppress	A	Suppress	A	Suppress	A	Suppress
I bit on	cc3	Complete	cc3	Complete	cc3	Complete	AT	Nullify
Sequence number in access register not equal to sequence number in entry	cc3	Complete	cc3	Complete	cc3	Complete	ALQ	Nullify
<u>ASN-second-table entry</u> ³ Invalid address of entry	A	Suppress	A	Suppress	A	Suppress	A	Suppress
I bit on	cc3	Complete	cc3	Complete	cc3	Complete	AV	Nullify
Sequence number in access-list entry not equal to sequence number in entry	cc3	Complete	cc3	Complete	cc3	Complete	ASQ	Nullify
<u>Authority-table entry</u> ^{3,4} Authority-table-length violation	cc3	Complete	cc3	Complete	cc3	Complete	EA	Nullify
Invalid address of entry	A	Suppress	A	Suppress	A	Suppress	A	Suppress
Second-authority bit not one	cc3	Complete	cc3	Complete	cc3	Complete	EA	Nullify
<u>Address-space-control element</u> Bits 0-10, 0-21, or 0-32 of instruction or operand address not all zeros when address-space-control element is a region-second-table designation, region-third-table designation, or segment-table designation, respectively	cc3	Complete	cc3	Complete	cc3	Complete	ATY	Nullify
<u>Region-table-entry designation by address-space-control element or region-table-entry</u> Entry outside of table	cc3	Complete	cc3	Complete	cc3	Complete	RT	Nullify
Invalid address of entry	A	Suppress	A	Suppress	A	Suppress	A	Suppress
I bit on	cc2	Complete	cc3	Complete	cc3	Complete	RT	Nullify
TT in entry not equal DT in address-space-control element or not one less than TT in next-higher-level entry	TS	Suppress	TS	Suppress	TS	Suppress	TS	Suppress

Figure 6-5. Handling of Access Exceptions (Part 1 of 3)

Condition ²	Translation for Virtual Address of LPTEA		Translation for Virtual Address of LOAD REAL ADDRESS		Translation for TAR and TPROT, and Access for Logical Address of TPROT ¹		Translation and Access for Any Other Address	
	Indication	Action	Indication	Action	Indication	Action	Indication	Action
<u>Segment-table entry designation by address-space-control element or region-table-entry</u>								
Entry outside of table	cc3	Complete	cc3	Complete	cc3	Complete	ST	Nullify
Invalid address of entry	A	Suppress	A	Suppress	A	Suppress	A	Suppress
l bit on (except as follows)	cc2	Complete	cc1	Complete	cc3	Complete	ST	Nullify
l bit on (LRA in 24-bit or 31-bit mode when bits 0-32 of entry address not all zeros)	-	-	cc3	Complete	-	-	-	-
One in a bit position which is checked for zero ⁵	TS	Suppress	TS	Suppress	TS	Suppress	TS	Suppress
TT in entry not equal DT in address-space-control element or not one less than TT in next-higher-level entry (TT not zero)	TS	Suppress	TS	Suppress	TS	Suppress	TS	Suppress
<u>Page-table entry</u>								
Invalid address of entry	-	-	A	Suppress	A	Suppress	A	Suppress
l bit on (except as follow)	-	-	cc2	Complete	cc3	Complete	PT	Nullify
l bit on (LRA in 24-bit or 31-bit mode when bits 0-32 of entry address not all zeros)	-	-	cc3	Complete	cc3	Complete	PT	Nullify
One in a bit position which is checked for zeros ⁵	-	-	TS	Suppress	TS	Suppress	TS	Suppress
<u>Access for instruction fetch</u>								
Location protected (key-controlled protection)	-	-	-	-	-	-	P	Suppress
Invalid address	-	-	-	-	-	-	A	Suppress
<u>Access for operand</u>								
Location protected (low-address, DAT, or key-controlled protection)	-	-	-	-	cc set ⁶	Complete	P	Term.*
Invalid address	-	-	-	-	A	Suppress	A	Term.*
Explanation - The condition does not apply. * Action is to terminate except where otherwise specified in this publication. For access-list-controlled protection and DAT protection, the action is always to suppress. ¹ TAR does not have a logical address. The rows "Address-space-control element" through "Access for operands" apply only to TPROT, not to TAR. ² Protection applies only to accesses for instruction fetch and for operands. It does not apply to the fetching of the effective access-list designation or any of the listed entries. ³ Exceptions related to an access register, effective access-list designation, access-list entry, ASN-second-table entry, or authority-table entry are recognized only in the access-register mode except that, for LOAD REAL ADDRESS and STORE REAL ADDRESS, they are recognized when PSW bits 16 and 17 are 01 binary, and, for TEST ACCESS, they are recognized regardless of the translation mode.								

Figure 6-5. Handling of Access Exceptions (Part 2 of 3)

Condition ²	Translation for Virtual Address of LPTEA		Translation for Virtual Address of LOAD REAL ADDRESS		Translation for TAR and TPROT, and Access for Logical Address of TPROT ¹		Translation and Access for Any Other Address	
	Indi-cation	Action	Indi-cation	Action	Indi-cation	Action	Indi-cation	Action
⁴	Authority table is not accessed and secondary-authority bit is not checked if the private bit in the access-list entry is zero or the access-list- entry authorization index in the access-list entry is equal to the extended authorization index in control register 8.							
⁵	A translation-specification exception for a format error in a table entry is recognized only when the execution of an instruction requires the entry for translation of an address.							
⁶	The condition code is set as follows: 0 Operand location not protected. 1 Fetches permitted, but stores not permitted. 2 Neither fetches nor stores permitted.							
A	Addressing exception.							
ALQ	ALE-sequence exception.							
AS	ALET-specification exception.							
ASQ	ASTE-sequence exception.							
AT	ALen-translation exception.							
ATY	ASCE-type exception.							
AV	ASTE-validity exception.							
cc1	Condition code 1 set.							
cc2	Condition code 2 set.							
cc3	Condition code 3 set.							
EA	Extended-authority exception.							
P	Protection exception.							
PT	Page-translation exception.							
RT	Region-first-translation, region-second-translation, or region-third translation exception, depending on the level of the table.							
ST	Segment-translation exception.							
TS	Translation-specification exception.							

Figure 6-5. Handling of Access Exceptions (Part 3 of 3)

able location or detects some other access-exception condition, but a branch instruction or an interruption changes the instruction sequence such that the instruction is not executed.

Every instruction can cause an access exception to be recognized because of instruction fetch. Additionally, access exceptions associated with instruction execution may occur because of an access to an operand in storage.

An access exception due to fetching an instruction is indicated when the first instruction halfword cannot be fetched without encountering the exception. When the first halfword of the instruction has no access exceptions, access exceptions may be indicated for

additional halfwords according to the instruction length specified by the first two bits of the instruction; however, when the operation can be performed without accessing the second or third halfwords of the instruction, it is unpredictable whether the access exception is indicated for the unused part. Since the indication of access exceptions for instruction fetch is common to all instructions, it is not covered in the individual instruction definitions.

Except where otherwise indicated in the individual instruction description, the following rules apply for exceptions associated with an access to an operand location. For a fetch-type operand, access exceptions are necessarily indicated only for that portion of the operand which is required for completing the opera-

tion. It is unpredictable whether access exceptions are indicated for those portions of a fetch-type operand which are not required for completing the operation. For a store-type operand, access exceptions are recognized for the entire operand even if the operation could be completed without the use of the inaccessible part of the operand. In situations where the value of a store-type operand is defined to be unpredictable, it is unpredictable whether an access exception is indicated.

Whenever an access to an operand location can cause an access exception to be recognized, the word “access” is included in the list of program exceptions in the description of the instruction. This entry also indicates which operand can cause the exception to be recognized and whether the exception is recognized on a fetch or store access to that operand location. Access exceptions are recognized only for the portion of the operand as defined for each particular instruction.

Multiple Program-Interruption Conditions

Except for PER basic events, only one program-interruption condition is indicated with a program interruption. The existence of one condition, however, does not preclude the existence of other conditions. When more than one program-interruption condition exists, only the condition having the highest priority is identified in the interruption code.

When multiple conditions of the same priority apply, it is unpredictable which is indicated.

When multiple parts of the same storage operand are subject to separate access controls, the priority of access exceptions associated with the parts is unpredictable and is not necessarily related to the sequence specified for the access of bytes within the operand. For example, when (a) the first operand of a MOVE (MVC) instruction crosses a segment boundary, (b) the invalid bit is one in the segment-table entry used to translate the leftmost part of the operand, and (c) the DAT-protection bit is one in a valid page-table entry used to translate the rightmost part of the operand, then it is unpredictable whether a segment-translation exception or protection exception is recognized.

When an instruction has two storage operands and access-exception conditions exist for both operands, it is unpredictable which condition is recognized. A subsequent execution of the same instruction (with the same exception conditions) may result in the exception condition being recognized for the same operand as the first execution, or for the other operand.

The type of ending which occurs (nullification, suppression, or termination) is that which is defined for the type of exception that is indicated in the interruption code. However, if a condition is indicated which permits termination, and another condition also exists which would cause either nullification or suppression, then the unit of operation is suppressed.

Figure 6-6 on page 6-45 lists the priorities of all program-interruption conditions other than PER basic events and exceptions associated with some of the more complex control instructions. All exceptions associated with references to storage for a particular instruction halfword or a particular operand byte are grouped as a single entry called “access.” Figure 6-7 on page 6-47 lists the priority of access exceptions for a single access. Thus, the second figure specifies which of several exceptions, encountered either in the access of a particular portion of an instruction or in any particular access associated with an operand, has highest priority, and the first figure specifies the priority of this condition in relation to other conditions detected in the operation. Similarly, the priorities for exceptions occurring as part of ASN translation and tracing are covered in Figure 6-8 on page 6-49 and Figure 6-10 on page 6-49, respectively.

For some instructions, the priority is shown in the individual instruction description.

The relative priorities of any two conditions listed in the figure can be found by comparing the priority numbers, as found in the figure, from left to right until a mismatch is found. If the first inequality is between numeric characters, either the two conditions are mutually exclusive or, if both can occur, the condition with the smaller number is indicated. If the first inequality is between alphabetic characters, then the two conditions are not exclusive, and it is unpredictable which is indicated when both occur.

To understand the use of the table, consider an example involving the instruction ADD DECIMAL, which is a six-byte instruction. Assume that the first four bytes of the instruction can be accessed but that

the instruction crosses a boundary so that an addressing exception exists for the last two bytes. Additionally, assume that the first operand addressed by the instruction contains invalid decimal digits and is in a location that can be fetched from, but not stored into, because of key-controlled protection. The three exceptions which could result from attempted execution of the ADD DECIMAL are:

Priority Number	Exception
6.2.B	Access exceptions for third instruction halfword
8.B	Access exceptions (operand 1).

Priority Number	Exception
8.D	Data exception

Since the first inequality ($6 \neq 8$) is between numeric characters, the addressing exception would be indicated. If, however, the entire ADD DECIMAL instruction can be fetched, and only the second two exceptions listed above exist, then the inequality ($B \neq D$) is between alphabetic characters, and it is unpredictable whether the protection exception or the data exception would be indicated.

1.	Specification exception due to any PSW error of the type that causes an immediate interruption. ¹
2.	Specification exception due to an odd instruction address in the PSW.
3.	Access exceptions for first halfword of an execute-type instruction. ²
4.A	Access exceptions for second halfword of an execute-type instruction. ²
4.B	Access exceptions for third halfword of EXECUTE RELATIVE LONG. ²
5.	Specification exception due to target instruction of EXECUTE not being specified on halfword boundary. ² (Without PER-3)
5.1	PER event due to instruction-fetching nullification on EXECUTE or EXECUTE RELATIVE LONG. (With PER-3)
5.2	Specification exception due to target instruction of EXECUTE not being specified on halfword boundary. ² (With PER-3)
6.	Access exceptions for first instruction halfword. (Without PER-3)
6.1	Access exceptions for first instruction halfword. (With PER-3)
6.2.A	Access exceptions for second instruction halfword. ³ (With PER-3)
6.2.B	Access exceptions for third instruction halfword. ³ (With PER-3)
6.3	PER event due to instruction-fetching nullification. (PER-3)
7.A	Access exceptions for second instruction halfword. ³ (Without PER-3)
7.B	Access exceptions for third instruction halfword. ³ (Without PER-3)
7.C.1	Operation exception.
7.C.2	Privileged-operation exception for privileged instructions.
7.C.3	Execute exception.
7.C.4	Special-operation exception. ⁴
8.A	Specification exception due to conditions other than those included in 1, 2, and 5 above.
8.B ⁵	Access exceptions for an access to an operand in storage. ⁶
8.C ⁵	Access exceptions for any other access to an operand in storage. ⁶
8.D	Data exception. ⁷
8.E	Decimal-divide exception. ⁸
8.F	Trace exceptions.

Figure 6-6. Priority of Program-Interruption Conditions

9.	<p>Events other than PER events, exceptions which result in completion, and the following exceptions:</p> <ul style="list-style-type: none"> • Fixed-point divide, • Floating-point divide, • Operand, • Special-operation recognized by any of the following: <ul style="list-style-type: none"> – LOAD PAGE-TABLE-ENTRY ADDRESS due to a real-space designation – LOAD REAL ADDRESS <p>See the definition of the TRAP instruction for other priorities of the special-operation exception.</p> <ul style="list-style-type: none"> • Square root. <p>Either these exceptions and events are mutually exclusive or their priority is specified in the corresponding definitions.</p>
<p>Explanation:</p> <p>Numbers indicate priority, with “1” being the highest priority; letters indicate no priority.</p>	
1	PSW errors which cause an immediate interruption may be introduced by a new PSW loaded as a result of an interruption or by the instructions LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, SET SYSTEM MASK, and STORE THEN OR SYSTEM MASK. The priority shown in the chart is for a PSW error introduced by an interruption and may also be considered as the priority for a PSW error introduced by the previous instruction. The error is introduced only if the instruction encounters no other exceptions. The resulting interruption has a higher priority than any interruption caused by the instruction which would have been executed next; it has lower priority, however, than any interruption caused by the instruction which introduced the erroneous PSW.
2	Priorities 3, 4.A, and 5.1 are for both EXECUTE and EXECUTE RELATIVE LONG, priority 4.B is for EXECUTE RELATIVE LONG, priority 5.2 is for EXECUTE; priorities starting with 6 are for the target instruction. When no execute-type instruction is encountered, priorities 3-5 do not apply.
3	Separate accesses may occur for each halfword of an instruction. The second instruction halfword is accessed only if bits 0-1 of the instruction are not both zeros. The third instruction halfword is accessed only if bits 0-1 of the instruction are both ones. Access exceptions for one of these halfwords are not necessarily recognized if the instruction can be completed without use of the contents of the halfword or if an exception of lower priority can be determined without the use of the halfword.
4	Except for the special-operation-exception conditions listed in priority 9.
5	As in instruction fetching, separate accesses may occur for each portion of an operand. Each of these accesses, and also accesses for different operands, are of equal priority, and the two entries 8.B and 8.C are listed to represent the relative priorities of exceptions associated with any two of these accesses. Access exceptions for INSERT STORAGE KEY EXTENDED, INSERT VIRTUAL STORAGE KEY, INVALIDATE PAGE TABLE ENTRY, LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, PERFORM FRAME MANAGEMENT FUNCTION, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY EXTENDED, STORE REAL ADDRESS, and TEST PROTECTION are also included in 8.B.
6	For MOVE LONG, MOVE LONG EXTENDED, COMPARE LOGICAL LONG, and COMPARE LOGICAL LONG EXTENDED, an access exception for a particular operand can be indicated only if the R field for that operand designates an even-numbered register.
7	The exception can be indicated only if the sign, digit, or digits responsible for the exception were fetched without encountering an access exception.
8	The exception can be indicated only if the digits used in establishing the exception, and also the signs, were fetched without encountering an access exception, only if the signs are valid, and only if the digits used in establishing the exception are valid.

Figure 6-6. Priority of Program-Interruption Conditions (Continued)

Access Exceptions

The access exceptions consist of those exceptions which can be encountered while using an absolute, instruction, logical, real, or virtual address to access

storage. Thus, in the access-register mode, the exceptions are:

1. ALET specification
2. ALEN translation
3. ALE sequence
4. ASTE validity
5. ASTE sequence
6. Extended authority
7. Addressing (the ART tables)
8. ASCE type
9. Region first translation
10. Region second translation
11. Region third translation
12. Segment translation
13. Page translation
14. Translation specification
15. Addressing (the DAT tables)
16. Addressing (the operand or instruction)
17. Protection (key-controlled, access-list-controlled, DAT, and low-address)

With DAT on but in other than the access-register mode, exceptions 8-17 in the above list, except for access-list-controlled protection, can be encountered.

With DAT off, the exceptions are:

18. Addressing (the operand or instruction)
19. Protection (key-controlled and low-address)

Additionally, even with DAT off, the instruction STORE REAL ADDRESS can encounter exceptions 1-17, the instructions LOAD PAGE-TABLE-ENTRY ADDRESS and LOAD REAL ADDRESS can encounter exceptions 7, 14, and 15, and the instructions INVALIDATE DAT TABLE ENTRY and INVALIDATE PAGE TABLE ENTRY can encounter exception 15.

The access exceptions are listed in more detail in Figure 6-7 on page 6-47.

Programming Note: The priorities in Figure 6-7 on page 6-47 could be renumbered, but they are kept as they are to allow easier comparison to the corresponding ESA/390 priorities. Specifically, B.1.A.1-B.1.A.9 could be changed to B.1-B.9, but ESA/390 contains “B.1.B Translation-specification exception due to invalid encoding of bits 8-12 of control register 0.”

A.	Protection exception (low-address protection) due to a store-type operand reference with an effective address in the range 0-511 or 4096-4607. Not recognized if DAT is on and the address-space-control element to be used in the translation cannot be obtained because of another exception.
B.1.A.1	ALET-specification exception due to bits 0-6 of access register not being all zeros. ¹
B.1.A.2	Addressing exception for access to effective access-list designation. ²
B.1.A.3	ALEN-translation exception due to access-list entry being outside the list. ¹
B.1.A.4	Addressing exception for access to access-list entry. ²
B.1.A.5	ALEN-translation exception due to I bit in access-list entry having the value one. ¹
B.1.A.6	ALE-sequence exception due to access-list-entry sequence number (ALESN) in access register not being equal to ALESN in access-list entry. ¹
B.1.A.7	Addressing exception for access to ASN-second-table entry. ²
B.1.A.8	ASTE-validity exception due to I bit in ASN-second-table entry having the value one. ¹
B.1.A.9	ASTE-sequence exception due to ASN-second-table-entry sequence number (ASTESN) in access-list entry not being equal to ASTESN in ASN-second-table entry. ¹
	Note: Exceptions B.1.A.10 through B.1.A.12 are recognized only when the private bit in the access-list entry is one and the ALEAX in the entry is not equal to the EAX in control register 8.
B.1.A.10	Extended-authority exception due to authority-table entry being outside table. ¹
B.1.A.11	Addressing exception for access to authority-table entry. ²

Figure 6-7. Priority of Access Exceptions (Part 1 of 3)

B.1.A.12	Extended-authority exception due to (1) private bit in access-list entry not being zero, (2) access-list-entry authorization index in access-list entry not being equal to extended authorization index in control register 8, and (3) secondary-authority bit selected by extended authorization index not being one. ¹
B.2.A	Protection exception (access-list-controlled protection) due to store-type operand reference to a virtual address which is protected against stores. ¹
B.2.B.1	ASCE-type exception due to bits 0-10, 0-21, or 0-32 of instruction or operand address not being zeros when address-space-control element is a region-second-table designation, region-third-table designation, or segment-table designation, respectively. ³
B.2.B.2	Region-first-, region-second-, region-third-, or segment-translation exception due to required entry in table designated by address-space-control element being outside of table. ³ Note: Exceptions B.2.B.3 through B.2.B.6 are recognized for a region-first-table, region-second-table, region-third-table, and segment-table entry in the order in which the entries are used.
B.2.B.3	Addressing exception for access to table entry. ⁴
B.2.B.4	Region-first-, region-second-, region-third-, or segment-translation exception due to I bit in table entry having the value one. ³
B.2.B.5	Translation-specification exception due to (1) TT in table entry not equal to DT in designating address-space-control element or not one less than TT in designating next-higher-level table entry or (2) invalid one in segment-table entry if this entry is a segment-table entry (common-segment bit if private-space bit in address-space-control element is one).
B.2.B.6	Region-second-, region-third-, or segment-translation exception due to required entry in next-lower-level table entry, if any, being outside of table. ³
B.2.B.7	Addressing exception for access to page-table entry. ⁵
B.2.B.8	Page-translation exception due to I bit in page-table entry having the value one. ^{3, 7}
B.2.B.9	Translation-specification exception due to invalid ones in page-table entry in which I bit is zero. ⁸ A one in bit position 52 of a valid PTE always causes the exception. When enhanced DAT does not apply, a one in bit position 55 of a valid PTE causes the exception.
B.3.A	Protection exception (DAT protection) due to a store-type operand reference to a virtual address which is protected against stores. ⁶
B.3.B	Addressing exception for access to instruction or operand.
B.4	Protection exception (key-controlled protection) due to attempt to access a protected instruction or operand location.
Explanation:	
¹	Not applicable when not in the access-register mode; not applicable for execution of TEST ACCESS and for translation of operand address of LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS, and TEST PROTECTION.
²	Not applicable when not in the access-register mode, except applicable for execution of TEST ACCESS and, when PSW bits 16 and 17 are 01 binary, for translation of operand address of LOAD REAL ADDRESS and second-operand address of STORE REAL ADDRESS; also applicable for LOAD PAGE-TABLE-ENTRY ADDRESS when the M ₄ field is 0001 binary or when the M ₄ field is 0100 binary and PSW bits 16-17 are 01 binary.
³	Not applicable when DAT is off except for translation of second-operand address of STORE REAL ADDRESS; not applicable to operand addresses of LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS and TEST PROTECTION.

Figure 6-7. Priority of Access Exceptions (Part 2 of 3)

4	Not applicable when DAT is off except for translation of operand address of LOAD PAGE-TABLE-ENTRY ADDRESS, LOAD REAL ADDRESS and second-operand address of STORE REAL ADDRESS.
5	Not applicable when DAT is off, except for execution of INVALIDATE PAGE TABLE ENTRY and for translation of operand address of LOAD PAGE-TABLE-ENTRY ADDRESS and LOAD REAL ADDRESS and second-operand address of STORE REAL ADDRESS.
6	Not applicable when DAT is off.
7	For MOVE PAGE, if the condition is true for both operands, the exception is recognized for the second operand. Also, if the condition-code-option bit is one, the exception is not recognized. Instead, condition code 1 is set if the condition is true for only the first operand, or condition code 2 is set if the condition is true for the second operand or both operands.
8	Not applicable when DAT is off except for translation of operand address of LOAD REAL ADDRESS and second-operand address of STORE REAL ADDRESS.

Figure 6-7. Priority of Access Exceptions (Part 3 of 3)

ASN-Translation Exceptions

The ASN-translation exceptions are those exceptions which are common to the process of translating an ASN in the instructions PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, and SET SECONDARY ASN WITH INSTANCE. The exceptions and the priority in which they are detected are shown in Figure 6-8.

1. Addressing exception for access to ASN- first-table entry.
2. AFX-translation exception due to I bit (bit 0) in ASN-first-table entry being one.
3. Addressing exception for access to ASN-second-table entry.
4. ASX-translation exception due to I bit (bit 0) in ASN-second-table entry being one.

Figure 6-8. Priority of ASN-Translation Exceptions

Subspace-Replacement Exceptions

The subspace-replacement exceptions are those exceptions which can be recognized during a subspace-replacement operation in PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, or SET SECONDARY ASN WITH INSTANCE. The exceptions and their priority are shown in Figure 6-9 on page 6-49.

1. Addressing exception for access to dispatchable-unit control table.

Figure 6-9. Priority of Subspace-Replacement Exceptions

2. Addressing exception for access to subspace ASN-second-table entry.
3. ASTE-validity exception due to bit 0 being one in subspace ASN-second-table entry.
4. ASTE-sequence exception due to subspace ASN-second-table-entry sequence number in dispatchable-unit control table not being equal to ASN-second-table-entry sequence number in subspace ASN-second-table entry.

Figure 6-9. Priority of Subspace-Replacement Exceptions

Trace Exceptions

The trace exceptions are those exceptions which can be encountered while forming a trace-table entry. The exceptions and their priority are shown in Figure 6-10 on page 6-49.

- A. Protection exception (low-address protection) due to entry address being in the range 0-511 or 4096-4607.
- B.1 Trace-table exception due to new entry reaching or crossing next 4K-byte boundary.
- B.2 Addressing exception for access to trace-table entry.

Figure 6-10. Priority of Trace Exceptions

Restart Interruption

The restart interruption provides a means for the operator or another CPU to invoke the execution of a specified program. The CPU cannot be disabled for this interruption.

A restart interruption causes the old PSW to be stored at real locations 288-303 and a new PSW, designating the start of the program to be executed, to be fetched from real 416-431. The instruction-length code and interruption code are not stored.

If the CPU is in the operating state, the exchange of the PSWs occurs at the completion of the current unit of operation and after all other pending interruption conditions for which the CPU is enabled have been honored. If the CPU is in the stopped state, the CPU enters the operating state and exchanges the PSWs without first honoring any other pending interruptions.

The restart interruption is initiated by activating the restart key. The operation can also be initiated at the addressed CPU by executing a SIGNAL PROCESSOR instruction which specifies the restart order.

When the rate control is set to the instruction-step position, it is unpredictable whether restart causes a unit of operation or additional interruptions to be performed after the PSWs have been exchanged.

Programming Note: To perform a restart when the CPU is in the check-stop state, the CPU has to be reset. Resetting with loss of the least amount of information can be accomplished by means of the system-reset-normal key, which does not clear the contents of program-addressable registers, including the control registers, but causes the channel subsystem to be reset. The CPU-reset SIGNAL PROCESSOR order can be used to clear the CPU without affecting the channel subsystem.

Supervisor-Call Interruption

The supervisor-call interruption occurs when the instruction SUPERVISOR CALL is executed. The CPU cannot be disabled for the interruption, and the interruption occurs immediately upon the execution of the instruction.

The supervisor-call interruption causes the old PSW to be stored at real locations 320-335 and a new PSW to be fetched from real locations 448-463.

The contents of bit positions 8-15 of the SUPERVISOR CALL instruction are placed in the rightmost byte of the interruption code. The leftmost byte of the interruption code is set to zero. The instruction-length code is 1, unless the instruction was executed by

means of an execute-type instruction. When the SUPERVISOR CALL instruction was executed by means of EXECUTE, the instruction-length code is 2; when it is executed by means of EXECUTE RELATIVE LONG, the instruction-length code is 3.

The interruption code is placed at real locations 138-139; the instruction-length code is placed in bit positions 5 and 6 of the byte at real location 137, with the other bits set to zeros; and zeros are stored at real location 136.

Priority of Interruptions

During the execution of an instruction, several interruption-causing events may occur simultaneously. The instruction may give rise to a program interruption, a request for an external interruption may be received, equipment malfunctioning may be detected, an I/O-interruption request may be made, and the restart key may be activated. Instead of the program interruption, a supervisor-call interruption might occur; or both can occur if PER is active. Simultaneous interruption requests are honored in a predetermined order.

An exigent machine-check condition has the highest priority. When it occurs, the current operation is terminated or nullified. Program and supervisor-call interruptions that would have occurred as a result of the current operation may be eliminated. Any pending repressible machine-check conditions may be indicated with the exigent machine-check interruption. Every reasonable attempt is made to limit the side effects of an exigent machine-check condition, and requests for external, I/O, and restart interruptions normally remain unaffected.

In the absence of an exigent machine-check condition, interruption requests existing concurrently at the end of a unit of operation are honored, in descending order of priority, as follows:

- Supervisor call
- Program
- Repressible machine check
- External
- Input/output
- Restart

The processing of multiple simultaneous interruption requests consists in storing the old PSW and fetching

the new PSW belonging to the interruption first honored. This new PSW is subsequently stored without the execution of any instructions, and the new PSW associated with the next interruption is fetched. Storing and fetching of PSWs continues until no more interruptions are to be serviced. The priority is reevaluated after each new PSW is loaded. Each evaluation takes into consideration any additional interruptions which may have become pending. Additionally, external and I/O interruptions, as well as machine-check interruptions due to repressible conditions, occur only if the current PSW at the instant of evaluation indicates that the CPU is interruptible for the cause.

Instruction execution is resumed using the last-fetched PSW. The order of executing interruption subroutines is, therefore, the reverse of the order in which the PSWs are fetched.

If the new PSW for a program interruption does not specify the wait state and has an odd instruction address, or causes an access exception to be recognized, another program interruption occurs. Since this second interruption introduces the same unacceptable PSW, a string of interruptions is established. These program exceptions are recognized as part of the execution of the following instruction, and the string may be broken by an external, I/O, machine-check, or restart interruption or by the stop function.

If the new PSW for a program interruption contains a one in bit position 12 or in an unassigned bit position, if the leftmost 40 bits of the instruction address are not zeros when bit 31 and 32 indicate 24-bit addressing, or the leftmost 33 bits are not zeros when bits 31 and 32 indicate 31-bit addressing, or if bit 32 is zero when bit 31 is one, another program interruption occurs. This condition is of higher priority than restart, I/O, external, or repressible machine-check

conditions, or the stop function, and CPU reset has to be used to break the string of interruptions.

A string of interruptions for other interruption classes can also exist if the new PSW allows the interruption which has just occurred. These include machine-check interruptions, external interruptions, and I/O interruptions due to PCI conditions generated because of CCWs which form a loop. Furthermore, a string of interruptions involving more than one interruption class can exist. For example, assume that the CPU timer is negative and the CPU-timer subclass mask is one. If the external new PSW has a one in an unassigned bit position, and the program new PSW is enabled for external interruptions, then a string of interruptions occurs, alternating between external and program. Even more complex strings of interruptions are possible. As long as more interruptions must be serviced, the string of interruptions cannot be broken by employing the stop function; CPU reset is required.

Similarly, CPU reset has to be invoked to terminate the condition that exists when an interruption is attempted with a prefix value designating a storage location that is not available to the CPU.

Interruptions for all requests for which the CPU is enabled occur before the CPU is placed in the stopped state. When the CPU is in the stopped state, restart has the highest priority.

Programming Note: The order in which concurrent interruption requests are honored can be changed to some extent by masking.

Chapter 7. General Instructions

Data Format	7-5	KM-TDEA-192 (KM Function Code 3)	7-53
Binary-Integer Representation	7-5	KM-Encrypted-TDEA-192 (KM Function Code 11)	7-53
Binary Arithmetic	7-6	KM-AES-128 (KM Function Code 18)	7-54
Signed Binary Arithmetic	7-6	KM-Encrypted-AES-128 (KM Function Code 26)	7-54
Addition and Subtraction	7-6	KM-AES-192 (KM Function Code 19)	7-55
Fixed-Point Overflow	7-7	KM-Encrypted-AES-192 (KM Function Code 27)	7-55
Unsigned Binary Arithmetic	7-7	KM-AES-256 (KM Function Code 20)	7-56
Signed and Logical Comparison	7-8	KM-Encrypted-AES-256 (KM Function Code 28)	7-56
Instructions	7-8	KM-XTS-AES-128 (KM Function Code 50)	7-57
ADD	7-24	KM-XTS-Encrypted-AES-128 (KM Function Code 58)	7-57
ADD IMMEDIATE	7-25	KM-XTS-AES-256 (KM Function Code 52)	7-59
ADD HALFWORD	7-27	KM-XTS-Encrypted-AES-256 (KM Function Code 60)	7-59
ADD HALFWORD IMMEDIATE	7-27	KMC-Query (KMC Function Code 0)	7-60
ADD HIGH	7-27	KMC-DEA (KMC Function Code 1)	7-61
ADD IMMEDIATE HIGH	7-28	KMC-Encrypted-DEA (KMC Function Code 9)	7-61
ADD LOGICAL	7-28	KMC-TDEA-128 (KMC Function Code 2)	7-62
ADD LOGICAL IMMEDIATE	7-28	KMC-Encrypted-TDEA-128 (KMC Function Code 10)	7-62
ADD LOGICAL HIGH	7-29	KMC-TDEA-192 (KMC Function Code 3)	7-63
ADD LOGICAL WITH CARRY	7-29	KMC-Encrypted-TDEA-192 (KMC Function Code 11)	7-63
ADD LOGICAL WITH SIGNED IMMEDIATE	7-30	KMC-AES-128 (KMC Function Code 18)	7-65
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH	7-31	KMC-Encrypted-AES-128 (KMC Function Code 26)	7-65
AND	7-31	KMC-AES-192 (KMC Function Code 19)	7-66
AND IMMEDIATE	7-32	KMC-Encrypted-AES-192 (KMC Function Code 27)	7-66
BRANCH AND LINK	7-33	KMC-AES-256 (KMC Function Code 20)	7-67
BRANCH AND SAVE	7-34	KMC-Encrypted-AES-256 (KMC Function Code 28)	7-67
BRANCH AND SAVE AND SET MODE	7-35	KMC-PRNG (KMC Function Code 67)	7-69
BRANCH AND SET MODE	7-36	CIPHER MESSAGE WITH CFB	7-71
BRANCH ON CONDITION	7-36	KMF-Query (Function Code 0)	7-75
BRANCH ON COUNT	7-37	KMF-DEA (Function Code 1)	7-75
BRANCH ON INDEX HIGH	7-38	KMF-Encrypted-DEA (Function Code 9)	7-75
BRANCH ON INDEX LOW OR EQUAL	7-38	KMF-TDEA-128 (Function Code 2)	7-77
BRANCH RELATIVE AND SAVE	7-39	KMF-Encrypted-TDEA-128 (Function Code 10)	7-77
BRANCH RELATIVE AND SAVE LONG	7-39	KMF-TDEA-192 (Function Code 3)	7-78
BRANCH RELATIVE ON CONDITION	7-40	KMF-Encrypted-TDEA-192 (Function Code 11)	7-78
BRANCH RELATIVE ON CONDITION LONG	7-40	KMF-AES-128 (Function Code 18)	7-80
BRANCH RELATIVE ON COUNT	7-41		
BRANCH RELATIVE ON COUNT HIGH	7-41		
BRANCH RELATIVE ON INDEX HIGH	7-41		
BRANCH RELATIVE ON INDEX LOW OR EQUAL	7-41		
CHECKSUM	7-43		
CIPHER MESSAGE	7-46		
CIPHER MESSAGE WITH CHAINING	7-46		
KM-Query (KM Function Code 0)	7-50		
KM-DEA (KM Function Code 1)	7-51		
KM-Encrypted-DEA (KM Function Code 9)	7-51		
KM-TDEA-128 (KM Function Code 2)	7-52		
KM-Encrypted-TDEA-128 (KM Function Code 10)	7-52		

KMF-Encrypted-AES-128 (Function Code 26)	7-80	COMPARE AND FORM CODEWORD	7-115
KMF-AES-192 (Function Code 19)	7-82	COMPARE AND SWAP	7-122
KMF-Encrypted-AES-192 (Function Code 27)	7-82	COMPARE DOUBLE AND SWAP	7-122
KMF-AES-256 (Function Code 20)	7-83	COMPARE AND SWAP AND STORE	7-124
KMF-Encrypted-AES-256 (Function Code 28)	7-83	COMPARE AND TRAP	7-127
CIPHER MESSAGE WITH COUNTER	7-86	COMPARE IMMEDIATE AND TRAP	7-127
KMCTR-Query (Function Code 0)	7-90	COMPARE HALFWORD	7-128
KMCTR-DEA (Function Code 1)	7-90	COMPARE HALFWORD IMMEDIATE	7-128
KMCTR-Encrypted-DEA (Function Code 9)	7-90	COMPARE HALFWORD RELATIVE LONG	7-129
KMCTR-TDEA-128 (Function Code 2)	7-91	COMPARE HIGH	7-129
KMCTR-Encrypted-TDEA-128 (Function Code 10)	7-91	COMPARE IMMEDIATE HIGH	7-130
KMCTR-TDEA-192 (Function Code 3)	7-92	COMPARE LOGICAL	7-130
KMCTR-Encrypted-TDEA-192 (Function Code 11)	7-92	COMPARE LOGICAL IMMEDIATE	7-130
KMCTR-AES-128 (Function Code 18)	7-94	COMPARE LOGICAL RELATIVE LONG	7-131
KMCTR-Encrypted-AES-128 (Function Code 26)	7-94	COMPARE LOGICAL AND BRANCH	7-132
KMCTR-AES-192 (Function Code 19)	7-95	COMPARE LOGICAL AND BRANCH RELATIVE	7-132
KMCTR-Encrypted-AES-192 (Function Code 27)	7-95	COMPARE LOGICAL IMMEDIATE AND BRANCH	7-132
KMCTR-AES-256 (Function Code 20)	7-96	COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE	7-132
KMCTR-Encrypted-AES-256 (Function Code 28)	7-96	COMPARE LOGICAL AND TRAP	7-133
CIPHER MESSAGE WITH OFB	7-98	COMPARE LOGICAL IMMEDIATE AND TRAP	7-133
KMO-Query (Function Code 0)	7-102	COMPARE LOGICAL CHARACTERS UNDER MASK	7-134
KMO-DEA (Function Code 1)	7-103	COMPARE LOGICAL HIGH	7-135
KMO-Encrypted-DEA (Function Code 9)	7-103	COMPARE LOGICAL IMMEDIATE HIGH	7-135
KMO-TDEA-128 (Function Code 2)	7-104	COMPARE LOGICAL LONG	7-136
KMO-Encrypted-TDEA-128 (Function Code 10)	7-104	COMPARE LOGICAL LONG EXTENDED	7-138
KMO-TDEA-192 (Function Code 3)	7-105	COMPARE LOGICAL LONG UNICODE	7-140
KMO-Encrypted-TDEA-192 (Function Code 11)	7-105	COMPARE LOGICAL STRING	7-143
KMO-AES-128 (Function Code 18)	7-107	COMPARE UNTIL SUBSTRING EQUAL	7-144
KMO-Encrypted-AES-128 (Function Code 26)	7-107	COMPRESSION CALL	7-148
KMO-AES-192 (Function Code 19)	7-108	COMPUTE INTERMEDIATE MESSAGE DIGEST	7-160
KMO-Encrypted-AES-192 (Function Code 27)	7-108	COMPUTE LAST MESSAGE DIGEST	7-160
KMO-AES-256 (Function Code 20)	7-110	KIMD-Query (KIMD Function Code 0)	7-163
KMO-Encrypted-AES-256 (Function Code 28)	7-110	KIMD-SHA-1 (KIMD Function Code 1)	7-163
COMPARE	7-112	KIMD-SHA-256 (KIMD Function Code 2)	7-164
COMPARE IMMEDIATE	7-112	KIMD-SHA-512 (KIMD Function Code 3)	7-164
COMPARE RELATIVE LONG	7-112	KIMD-GHASH (Function Code 65)	7-165
COMPARE AND BRANCH	7-113	KLMD-Query (KLMD Function Code 0)	7-165
COMPARE AND BRANCH RELATIVE	7-113	KLMD-SHA-1 (KLMD Function Code 1)	7-166
COMPARE IMMEDIATE AND BRANCH	7-113	KLMD-SHA-256 (KLMD Function Code 2)	7-167
COMPARE IMMEDIATE AND BRANCH RELATIVE	7-114	KLMD-SHA-512 (KLMD Function Code 3)	7-169
		COMPUTE MESSAGE AUTHENTICATION CODE	7-172
		KMAC-Query (Function Code 0)	7-176
		KMAC-DEA (Function Code 1)	7-176
		KMAC-Encrypted-DEA (Function Code 9)	7-176
		KMAC-TDEA-128 (Function Code 2)	7-177
		KMAC-Encrypted-TDEA-128 (Function Code 10)	7-177

KMAC-TDEA-192 (Function Code 3) . . .	7-178	LOAD COMPLEMENT	7-222
KMAC-Encrypted-TDEA-192 (Function Code 11)	7-178	LOAD HALFWORD	7-222
KMAC-AES-128 (Function Code 18) . . .	7-179	LOAD HALFWORD IMMEDIATE	7-223
KMAC-Encrypted-AES-128 (Function Code 26)	7-179	LOAD HALFWORD RELATIVE LONG	7-223
KMAC-AES-192 (Function Code 19) . . .	7-180	LOAD HALFWORD HIGH	7-224
KMAC-Encrypted-AES-192 (Function Code 27)	7-180	LOAD HIGH	7-224
KMAC-AES-256 (Function Code 20) . . .	7-180	LOAD LOGICAL	7-224
KMAC-Encrypted-AES-256 (Function Code 28)	7-180	LOAD LOGICAL RELATIVE LONG	7-224
CONVERT TO BINARY	7-182	LOAD LOGICAL CHARACTER	7-225
CONVERT TO DECIMAL	7-183	LOAD LOGICAL CHARACTER HIGH	7-225
CONVERT UTF-16 TO UTF-32	7-184	LOAD LOGICAL HALFWORD	7-225
CONVERT UTF-16 TO UTF-8	7-187	LOAD LOGICAL HALFWORD RELATIVE LONG	7-226
CONVERT UNICODE TO UTF-8	7-187	LOAD LOGICAL HALFWORD HIGH	7-226
CONVERT UTF-32 TO UTF-16	7-190	LOAD LOGICAL IMMEDIATE	7-226
CONVERT UTF-32 TO UTF-8	7-193	LOAD LOGICAL THIRTY ONE BITS	7-227
CONVERT UTF-8 TO UTF-16	7-196	LOAD MULTIPLE	7-227
CONVERT UTF-8 TO UNICODE	7-196	LOAD MULTIPLE DISJOINT	7-228
CONVERT UTF-8 TO UTF-32	7-200	LOAD MULTIPLE HIGH	7-228
COPY ACCESS	7-204	LOAD NEGATIVE	7-229
DIVIDE	7-204	LOAD ON CONDITION	7-229
DIVIDE LOGICAL	7-205	LOAD PAIR DISJOINT	7-230
DIVIDE SINGLE	7-205	LOAD PAIR FROM QUADWORD	7-231
EXCLUSIVE OR	7-206	LOAD POSITIVE	7-231
EXCLUSIVE OR IMMEDIATE	7-207	LOAD REVERSED	7-232
EXECUTE	7-208	MONITOR CALL	7-233
EXECUTE RELATIVE LONG	7-208	Monitor-Event Program Interruption . . .	7-233
EXTRACT ACCESS	7-209	Monitor-Event Counting Operation	7-233
EXTRACT CACHE ATTRIBUTE	7-209	MOVE	7-234
EXTRACT CPU TIME	7-211	MOVE INVERSE	7-234
EXTRACT PSW	7-211	MOVE LONG	7-235
FIND LEFTMOST ONE	7-212	MOVE LONG EXTENDED	7-239
INSERT CHARACTER	7-212	MOVE LONG UNICODE	7-242
INSERT CHARACTERS UNDER MASK . . .	7-213	MOVE NUMERICS	7-245
INSERT IMMEDIATE	7-213	MOVE STRING	7-246
INSERT PROGRAM MASK	7-214	MOVE WITH OFFSET	7-247
LOAD	7-214	MOVE ZONES	7-248
LOAD IMMEDIATE	7-215	MULTIPLY	7-248
LOAD RELATIVE LONG	7-215	MULTIPLY HALFWORD	7-249
LOAD ACCESS MULTIPLE	7-215	MULTIPLY HALFWORD IMMEDIATE	7-249
LOAD ADDRESS	7-216	MULTIPLY LOGICAL	7-250
LOAD ADDRESS EXTENDED	7-216	MULTIPLY SINGLE	7-250
LOAD ADDRESS RELATIVE LONG	7-217	MULTIPLY SINGLE IMMEDIATE	7-251
LOAD AND ADD	7-218	OR	7-251
LOAD AND ADD LOGICAL	7-218	OR IMMEDIATE	7-252
LOAD AND AND	7-219	PACK	7-253
LOAD AND EXCLUSIVE OR	7-219	PACK ASCII	7-254
LOAD AND OR	7-220	PACK UNICODE	7-255
LOAD AND TEST	7-221	PERFORM CRYPTOGRAPHIC COMPUTATION	7-256
LOAD BYTE	7-221	PCC-Query (Function Code 0)	7-259
LOAD BYTE HIGH	7-222	PCC-Compute-Last-Block-CMAC-Using- DEA (Function Code 1)	7-259

PCC-Compute-Last-Block-CMAC-Using-Encrypted-DEA (Function Code 9)	7-259	SEARCH STRING	7-292
PCC-Compute-Last-Block-CMAC-Using-TDEA-128 (Function Code 2)	7-260	SEARCH STRING UNICODE	7-294
PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-128 (Function Code 10)	7-260	SET ACCESS	7-296
PCC-Compute-Last-Block-CMAC-Using-TDEA-192 (Function Code 3)	7-262	SET ADDRESSING MODE	7-296
PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-192 (Function Code 11)	7-262	SET PROGRAM MASK	7-297
PCC-Compute-Last-Block-CMAC-Using-AES-128 (Function Code 18)	7-263	SHIFT LEFT DOUBLE	7-297
PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-128 (Function Code 26)	7-263	SHIFT LEFT DOUBLE LOGICAL	7-298
PCC-Compute-Last-Block-CMAC-Using-AES-192 (Function Code 19)	7-264	SHIFT LEFT SINGLE	7-298
PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-192 (Function Code 27)	7-264	SHIFT LEFT SINGLE LOGICAL	7-299
PCC-Compute-Last-Block-CMAC-Using-AES-256 (Function Code 20)	7-266	SHIFT RIGHT DOUBLE	7-300
PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-256 (Function Code 28)	7-266	SHIFT RIGHT DOUBLE LOGICAL	7-300
PCC-Compute-XTS-Parameter-Using-AES-128 (Function Code 50)	7-267	SHIFT RIGHT SINGLE	7-301
PCC-Compute-XTS-Parameter-Using-Encrypted-AES-128 (Function Code 58)	7-267	SHIFT RIGHT SINGLE LOGICAL	7-302
PCC-Compute-XTS-Parameter-Using-AES-256 (Function Code 52)	7-269	STORE	7-302
PCC-Compute-XTS-Parameter-Using-Encrypted-AES-256 (Function Code 60)	7-269	STORE RELATIVE LONG	7-302
PERFORM LOCKED OPERATION	7-271	STORE ACCESS MULTIPLE	7-303
Function Codes 0-3 (Compare and Load)	7-276	STORE CHARACTER	7-303
Function Codes 4-7 (Compare and Swap)	7-276	STORE CHARACTER HIGH	7-304
Function Codes 8-11 (Double Compare and Swap)	7-277	STORE CHARACTERS UNDER MASK	7-304
Function Codes 12-15 (Compare and Swap and Store)	7-277	STORE CLOCK	7-305
Function Codes 16-19 (Compare and Swap and Double Store)	7-278	STORE CLOCK FAST	7-305
Function Codes 20-23 (Compare and Swap and Triple Store)	7-279	STORE CLOCK EXTENDED	7-306
POPULATION COUNT	7-285	STORE FACILITY LIST EXTENDED	7-307
PREFETCH DATA	7-286	STORE HALFWORD	7-308
PREFETCH DATA RELATIVE LONG	7-286	STORE HALFWORD RELATIVE LONG	7-308
ROTATE LEFT SINGLE LOGICAL	7-287	STORE HALFWORD HIGH	7-309
ROTATE THEN AND SELECTED BITS	7-288	STORE HIGH	7-309
ROTATE THEN EXCLUSIVE OR SELECTED BITS	7-288	STORE MULTIPLE	7-309
ROTATE THEN OR SELECTED BITS	7-288	STORE MULTIPLE HIGH	7-310
ROTATE THEN INSERT SELECTED BITS	7-289	STORE ON CONDITION	7-310
ROTATE THEN INSERT SELECTED BITS HIGH	7-291	STORE PAIR TO QUADWORD	7-311
ROTATE THEN INSERT SELECTED BITS LOW	7-291	STORE REVERSED	7-311
		SUBTRACT	7-312
		SUBTRACT HALFWORD	7-313
		SUBTRACT HIGH	7-313
		SUBTRACT LOGICAL	7-314
		SUBTRACT LOGICAL IMMEDIATE	7-314
		SUBTRACT LOGICAL HIGH	7-315
		SUBTRACT LOGICAL WITH BORROW	7-315
		SUPERVISOR CALL	7-316
		TEST ADDRESSING MODE	7-316
		TEST AND SET	7-317
		TEST UNDER MASK (TEST UNDER MASK HIGH, TEST UNDER MASK LOW)	7-317
		TRANSLATE	7-318
		TRANSLATE AND TEST	7-319
		TRANSLATE AND TEST EXTENDED	7-320
		TRANSLATE AND TEST REVERSE EXTENDED	7-320
		TRANSLATE AND TEST REVERSE	7-324
		TRANSLATE EXTENDED	7-325
		TRANSLATE ONE TO ONE	7-328
		TRANSLATE ONE TO TWO	7-328
		TRANSLATE TWO TO ONE	7-328

TRANSLATE TWO TO TWO.....	7-328
UNPACK	7-332
UNPACK ASCII	7-333
UNPACK UNICODE	7-334

UPDATE TREE.....	7-335
Protection of Cryptographic Key.....	7-340
Protection of DES Keys	7-341
Protection of AES Keys.....	7-342

This chapter includes most of the unprivileged instructions described in this publication. Other unprivileged instructions are described in chapters 8-9 and 18-20.

Data Format

The general instructions treat data as being of four types: signed binary integers, unsigned binary integers, unstructured logical data, and decimal data. Data is treated as decimal by the conversion, packing, and unpacking instructions. Decimal data is described in Chapter 8, "Decimal Instructions."

The general instructions manipulate data which resides in general registers or in storage or is introduced from the instruction stream. Some general instructions operate on data which resides in the PSW or the TOD clock.

In a storage-and-storage operation the operand fields may be defined in such a way that they overlap. The effect of this overlap depends upon the operation. When the operands remain unchanged, as in COMPARE or TRANSLATE AND TEST, overlapping does not affect the execution of the operation. For instructions such as MOVE and TRANSLATE, one operand is replaced by new data, and the execution of the operation may be affected by the amount of overlap and the manner in which data is fetched or stored. For purposes of evaluating the effect of overlapped operands, data is considered to be handled one eight-bit byte at a time. Special rules apply to the operands of MOVE LONG and MOVE INVERSE. See "Interlocks within a Single Instruction" on page 5-87 for how overlap is detected in the access-register mode.

Binary-Integer Representation

Binary integers are treated as signed or unsigned.

In an unsigned binary integer, all bits are used to express the absolute value of the number. When two unsigned binary integers of different lengths are added, the shorter number is considered to be extended on the left with zeros.

In some operations, the result is achieved by the use of the one's complement of the number. The one's complement of a number is obtained by inverting each bit of the number, including the sign.

For signed binary integers, the leftmost bit represents the sign, which is followed by the numeric field. Positive numbers are represented in true binary notation with the sign bit set to zero. When the value is zero, all bits are zeros, including the sign bit. Negative numbers are represented in two's-complement binary notation with a one in the sign-bit position.

Specifically, a negative number is represented by the two's complement of the positive number of the same absolute value. The two's complement of a number is obtained by forming the one's complement of the number, adding a value of one in the rightmost bit position, allowing a carry into the sign position, and ignoring any carry out of the sign position.

This number representation can be considered the rightmost portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number are zeros. When the number is negative, these bits are ones. Therefore, when a signed operand must be extended with bits on the left, the extension is achieved by setting these bits equal to the sign bit of the operand.

The notation for signed binary integers does not include a negative zero. It has a number range in which, for a given length, the set of negative nonzero numbers is one larger than the set of positive nonzero numbers. The maximum positive number consists of a sign bit of zero followed by all ones, whereas the maximum negative number (the negative number with the greatest absolute value) consists of a sign bit of one followed by all zeros.

A signed binary integer of either sign, except for zero and the maximum negative number, can be changed to a number of the same magnitude but opposite sign by forming its two's complement. Forming the two's complement of a number is equivalent to subtracting the number from zero. The two's complement of zero is zero.

The two's complement of the maximum negative number cannot be represented in the same number of bits. When an operation, such as `LOAD COMPLEMENT`, attempts to produce the two's complement of the maximum negative number, the result is the maximum negative number, and a fixed-point-overflow exception is recognized. An overflow does not result, however, when the maximum negative number is complemented as an intermediate result but the final result is within the representable range. An example of this case is a subtraction of the maximum negative number from -1. The product of two maximum negative numbers of a given length is representable as a positive number of double that length.

In discussions of signed binary integers in this publication, a signed binary integer includes the sign bit. Thus, the expression "32-bit signed binary integer" denotes an integer with 31 numeric bits and a sign bit, and the expression "64-bit signed binary integer" denotes an integer with 63 numeric bits and a sign bit.

In an arithmetic operation, a carry out of the numeric field of a signed binary integer is carried into the sign bit. However, in algebraic left-shifting, the sign bit does not change even if significant numeric bits are shifted out.

Programming Notes:

1. An alternate way of forming the two's complement of a signed binary integer is to invert all bits to the left of the rightmost one bit, leaving the rightmost one bit and all zero bits to the right of it unchanged.
2. The numeric bits of a signed binary integer may be considered to represent a positive value, with the sign representing a value of either zero or the maximum negative number.

Binary Arithmetic

Many of the instructions that perform a register-and-storage or register-and-register binary-arithmetic operation are provided in sets of three instructions corresponding to three different combinations of operand lengths. These three instructions have the same name but different operation codes and mnemonics. For example, `ADD (A)` operates on 32-bit operands and produces a 32-bit result, `ADD (AG)` operates on 64-bit operands and produces a 64-bit result, and `ADD (AGF)` operates on a 64-bit operand and a 32-bit operand and produces a 64-bit result. The letter "G" alone in the mnemonic indicates a completely 64-bit operation, and the letters "GF" indicate a 32-to-64-bit operation.

In a 32-to-64-bit operation, the intermediate result is 64 bits. `LOAD COMPLEMENT (LCGFR)` forms the two's complement of the maximum negative 32-bit number without recognizing overflow.

A 32-bit operand in a general register is in bit positions 32-63 of the register. In an operation on the operand, such as by `ADD (A)`, bits 0-31 of the register are unused and remain unchanged. A 64-bit operand in a general register is in bit positions 0-63 of the register, and all of the bits participate in an operation on the operand, such as by `ADD (AG)`. However, some instructions, which do not have "G" in their mnemonics, use a 64-bit operand of which the leftmost 32 bits are in bit positions 32-63 of the even register of an even-odd general-register pair, and the rightmost 32 bits are in bit positions 32-63 of the odd register of the pair.

The bits of a 32-bit operand in storage are numbered 0-31. When the operand is in bit positions 32-63 of a general register, the bits are numbered 32-63.

Signed Binary Arithmetic

Addition and Subtraction

Addition of signed binary integers is performed by adding all bits of each operand, including the sign bits. When one of the operands is shorter, the shorter operand is considered to be extended on the left to the length of the longer operand by propagating the sign-bit value.

For a 32-bit signed binary integer in a general register, the sign bit is bit 32 of the register. For a 64-bit signed binary integer in a general register, the sign bit is bit 0 of the register.

Subtraction is performed by adding the one's complement of the second operand and a value of one to the first operand.

Fixed-Point Overflow

A fixed-point-overflow condition exists for signed binary addition or subtraction when the carry out of the sign-bit position and the carry out of the leftmost numeric bit position disagree. Detection of an overflow does not affect the result produced by the addition. In mathematical terms, signed addition and subtraction produce a fixed-point overflow when the result is outside the range of representation for signed binary integers. Specifically, for ADD (A) and SUBTRACT (S), which operate on 32-bit signed binary integers, there is an overflow when the proper result would be greater than or equal to $+2^{31}$ or less than -2^{31} . The actual result placed in the general register after an overflow differs from the proper result by 2^{32} . A fixed-point overflow causes a program interruption if allowed by the program mask. Similarly, for ADD (AG) and SUBTRACT (SG), which operate on 64-bit signed binary integers, there is an overflow when the proper result would be greater than or equal to $+2^{63}$ or less than -2^{63} , and the actual result placed in the general register after an overflow differs from the proper result by 2^{64} . ADD (AGF) and SUBTRACT (SGF) have the same 64-bit result and overflow rules as ADD (AG) and SUBTRACT (SG).

The instructions SHIFT LEFT SINGLE and SHIFT LEFT DOUBLE produce an overflow when the result is outside the range of representation for signed binary integers. The actual result differs from that for addition and subtraction in that the sign of the result remains the same as the original sign.

Unsigned Binary Arithmetic

Addition of unsigned binary integers is performed by adding all bits of each operand. Subtraction is performed by adding the one's complement of the second operand (the subtrahend) and a value of one to the first operand (the minuend). In any case, when one of the operands is shorter, the shorter operand is considered to be extended on the left with zeros. During subtraction, this extension applies before an

operand is complemented, and it applies to the value of one.

Unsigned binary arithmetic is used in address arithmetic for adding the X, B, and D fields. (See "Address Generation" on page 5-8.) It is also used to obtain the addresses of the function bytes in TRANSLATE and TRANSLATE AND TEST. Furthermore, unsigned binary arithmetic is used on 32-bit or 64-bit unsigned binary integers by ADD LOGICAL, ADD LOGICAL WITH CARRY, DIVIDE LOGICAL, MULTIPLY LOGICAL, SUBTRACT LOGICAL, and SUBTRACT LOGICAL WITH BORROW.

Given the same length operands, ADD (A, AG, AGF) and ADD LOGICAL (AL, ALG, ALGF) produce the same 32-bit or 64-bit result. The instructions differ only in the interpretation of this result. ADD interprets the result as a signed binary integer and inspects it for sign, magnitude, and overflow to set the condition code accordingly. ADD LOGICAL interprets the result as an unsigned binary integer and sets the condition code according to whether the result is zero and whether there was a carry out of bit position 32, for a 32-bit integer, or out of bit position 0 for a 64-bit integer. Such a carry is not considered an overflow, and no program interruption for overflow can occur for ADD LOGICAL.

SUBTRACT LOGICAL differs from ADD LOGICAL in that the one's complement of the second operand and a value of one are added to the first operand.

For ADD LOGICAL WITH CARRY, a carry from a previous operation is represented by a one value of bit 18 of the current PSW. Bit 18 is the leftmost bit of the two-bit condition code in the PSW. For SUBTRACT LOGICAL WITH BORROW, a borrow from a previous operation is represented by a zero value of bit 18. A borrow is equivalent to the absence of a carry.

Programming Notes:

1. Logical addition and subtraction may be used to perform arithmetic on multiple-precision binary-integer operands. Thus, for multiple-precision addition, ADD LOGICAL can be used to add the lowest-order corresponding parts of the operands, and ADD LOGICAL WITH CARRY can be used to add the other corresponding parts of the operands, moving from right to left in the operands. If the multiple-precision operands are signed, ADD should be used on the highest-

order parts. The condition code then indicates any overflow or the proper sign and magnitude of the entire result; an overflow is also indicated by a program interruption for fixed-point overflow if allowed by the program mask. When ADD is used, a value of one must be added to the sum of the highest-order parts if the condition code indicated there was a carry from the addition of the next-lower parts.

2. Another use for ADD LOGICAL is to increment values representing binary counters, which are allowed to wrap around from all ones to all zeros without indicating overflow.

Signed and Logical Comparison

Comparison operations determine whether two operands are equal or not and, for most operations, which of two unequal operands is the greater (high). Signed-binary-comparison operations are provided which treat the operands as signed binary integers, and logical-comparison operations are provided which treat the operands as unsigned binary integers or as unstructured data.

COMPARE (C, CG, CGF) and COMPARE HALFWORD are signed-binary-comparison operations. These instructions are equivalent to SUBTRACT (S, SG, SGF) and SUBTRACT HALFWORD without replacing either operand, the resulting difference being used only to set the condition code. The operations permit comparison of numbers of opposite sign which differ by 2^{63} or more. Thus, unlike SUBTRACT, COMPARE cannot cause overflow.

Logical comparison of two operands is performed byte by byte, in a left-to-right sequence. The operands are equal when all their bytes are equal. When the operands are unequal, the comparison result is determined by a left-to-right comparison of corresponding bit positions in the first unequal pair of bytes: the zero bit in the first unequal pair of bits indicates the low operand, and the one bit the high operand. Since the remaining bit and byte positions do not change the comparison, it is not necessary to continue comparing unequal operands beyond the first unequal bit pair.

Instructions

The general instructions and their mnemonics, formats, and operation codes are listed in Figure 7-1 on page 7-13. The figure also indicates which instructions are new in z/Architecture as compared to ESA/390, when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

The instructions that are new in z/Architecture are indicated in Figure 7-1 by “N.” A few of the instructions that are new in z/Architecture have also been added to ESA/390, and these are indicated by “N3.”

When the operands of an instruction are 32-bit operands, the mnemonic for the instruction normally does not include a letter indicating the operand length; however some mnemonics for instructions with 32-bit operands contain the letter “F”. If there is an instruction with the same name but with 64-bit operands, its mnemonic includes the letter “G.” If there is an instruction with the same name but with a 64-bit first operand and a 32-bit second operand, its mnemonic includes the letters “GF.” Certain instruction mnemonics indicate the length of the first and second operands with combinations of G (64-bit), F (32-bit), or H (16-bit), although the F is usually omitted for the first operand.

In Figure 7-1, when there is an instruction with 32-bit operands and other instructions with the same name but with “G” or “GF” added in their mnemonics, the first instruction has “(32)” after its name, and the other instructions have “(64)” or “(64←32),” respectively, after their names. Some 32-bit operand-length instructions do not have 64-bit operand-length counterparts, and they do not have “(32)” after their names. However, all instructions for multiplication or division are marked to show, or approximately show, operand lengths.

A detailed definition of instruction formats, operand designation and length, and address generation is contained in “Instructions” on page 5-2. Exceptions to the general rules stated in that section are explicitly identified in the individual instruction descriptions.

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designations for the assembler language are

shown with each instruction. For LOAD AND TEST with 32-bit operands, for example, LTR is the mnemonic and R_1, R_2 the operand designation.

Certain fields of various instructions are considered to be optional, as indicated by the field being contained within brackets [] in the assembler syntax. When an optional field is not specified, the assembler places zeros in that field of the instruction.

Programming Notes:

1. Trimodal addressing affects the general instructions only in the manner in which logical storage addresses are handled, except as follows.

The instructions BRANCH AND LINK (BAL, BALR), BRANCH AND SAVE (BAS, BASR), BRANCH AND SAVE AND SET MODE, BRANCH AND SET MODE, BRANCH RELATIVE AND SAVE, and BRANCH RELATIVE AND SAVE LONG place information in bit positions 32-39 of general register R_1 as in ESA/390 in the 24-bit or 31-bit addressing mode or place address bits in those bit positions in the 64-bit addressing mode.

The instruction BRANCH AND SAVE AND SET MODE places a zero in bit position 63 of general register R_1 in the 24-bit or 31-bit addressing mode or places a one in that bit position in the 64-bit addressing mode.

The instruction BRANCH AND SET MODE leaves the contents of bit position 63 of general register R_1 unchanged in the 24-bit or 31-bit addressing mode or places a one in that bit position in the 64-bit addressing mode.

The following instructions leave bits 0-31 of a general register unchanged in the 24-bit or 31-bit addressing mode but place or update address or length information in them in the 64-bit addressing mode. Also, the leftmost byte of the results in registers may be handled differently depending on whether the addressing mode is the 24-bit or the 31-bit mode.

- BRANCH AND LINK (BAL, BALR)
- BRANCH AND SAVE (BAS, BASR)
- BRANCH AND SAVE AND SET MODE
- BRANCH RELATIVE AND SAVE
- BRANCH RELATIVE AND SAVE LONG
- CHECKSUM
- COMPARE AND FORM CODEWORD

- COMPARE LOGICAL LONG
- COMPARE LOGICAL LONG EXTENDED
- COMPARE LOGICAL LONG UNICODE
- COMPARE LOGICAL STRING
- COMPARE UNTIL SUBSTRING EQUAL
- COMPRESSION CALL
- CONVERT UNICODE TO UTF-8
- CONVERT UTF-8 TO UNICODE
- LOAD ADDRESS
- LOAD ADDRESS EXTENDED
- LOAD ADDRESS RELATIVE LONG
- MOVE LONG
- MOVE LONG EXTENDED
- MOVE LONG UNICODE
- MOVE STRING
- SEARCH STRING
- TRANSLATE EXTENDED
- TRANSLATE AND TEST
- TRANSLATE AND TEST EXTENDED
- TRANSLATE AND TEST REVERSE
- TRANSLATE AND TEST REVERSE EXTENDED
- TRANSLATE ONE TO ONE
- TRANSLATE ONE TO TWO
- TRANSLATE TWO TO ONE
- TRANSLATE TWO TO TWO
- UPDATE TREE

The instructions in the preceding list are sometimes called modal instructions.

2. Bits 0-31 of general registers are changed by two types of instructions. The first type is a modal instruction, as listed in the preceding note, when the instruction is executed in the 64-bit addressing mode. The second type is an instruction having, independent of the addressing mode, either a 64-bit result operand in a single general register or a 128-bit result operand in an even-odd general-register pair.

Most of the instructions of the second type are indicated by a "G," either alone or in "GF," in their mnemonics. The other instructions that change or may change bits 0-31 of a general register regardless of the current addressing mode are:

- AND IMMEDIATE (NIHF, NIHH, NIHL only)
- EXCLUSIVE OR IMMEDIATE (XIHF only)
- INSERT CHARACTERS UNDER MASK (ICMH only)
- INSERT IMMEDIATE (IIHF, IIHH, IIHL only)
- LOAD LOGICAL IMMEDIATE (LLIHF, LLIHH, LLIHL only)
- LOAD MULTIPLE DISJOINT

- LOAD MULTIPLE HIGH
- LOAD PAIR FROM QUADWORD
- OR IMMEDIATE (OIHF, OIHH, OIHL only)

All of the instructions of the second type are sometimes referred to as “G-type” instructions.

If a program is not executed in the 64-bit addressing mode and does not contain a G-type instruction, it cannot change bits 0-31 of any general register.

3. It is not intended or expected that old programs not containing G-type instructions will be able to be executed successfully in the 64-bit addressing mode. However, this may be possible, particularly if, by programming convention, bits 0-31 of the general registers are always all zeros when an old program is given control.
4. The following additional general instructions are available when the extended-translation facility 2 is installed:
 - COMPARE LOGICAL LONG UNICODE
 - MOVE LONG UNICODE
 - PACK ASCII
 - PACK UNICODE
 - TRANSLATE ONE TO ONE
 - TRANSLATE ONE TO TWO
 - TRANSLATE TWO TO ONE
 - TRANSLATE TWO TO TWO
 - UNPACK ASCII
 - UNPACK UNICODE
5. The long-displacement facility uses new instruction formats, named RSY, RXY, and SIY, to provide 20-bit signed displacements. In connection with the long-displacement facility, all previously existing general instructions of the RSE or RXE format are changed to be of format RSY or RXY, respectively, where the new formats differ from the old by using a previously unused byte, now named DH, in the instructions. When the long-displacement facility is installed, the displacement for an instruction operand address is formed by appending DH on the left of the previous displacement field, now named DL, of the instruction. When the long-displacement facility is not installed, eight zero bits are appended on the left of DL, and DH is ignored.

The following additional general instruction is available when the long-displacement facility is installed.

- LOAD BYTE

The following additional RSY-format versions of general instructions are available when the long-displacement facility is installed.

- COMPARE AND SWAP
- COMPARE DOUBLE AND SWAP
- COMPARE LOGICAL CHARACTERS UNDER MASK
- LOAD ACCESS MULTIPLE
- LOAD MULTIPLE
- STORE ACCESS MULTIPLE
- STORE CHARACTERS UNDER MASK
- STORE MULTIPLE

The following additional RXY-format versions of general instructions are available when the long-displacement facility is installed.

- ADD
- ADD HALFWORD
- ADD LOGICAL
- AND
- COMPARE
- COMPARE HALFWORD
- COMPARE LOGICAL
- CONVERT TO BINARY
- CONVERT TO DECIMAL
- EXCLUSIVE OR
- INSERT CHARACTER
- INSERT CHARACTER UNDER MASK
- LOAD
- LOAD ADDRESS
- LOAD HALFWORD
- MULTIPLY SINGLE
- OR
- STORE
- STORE CHARACTER
- STORE HALFWORD
- SUBTRACT
- SUBTRACT HALFWORD
- SUBTRACT LOGICAL

The following additional SIY-format versions of general instructions are available when the long-displacement facility is installed.

- AND
- COMPARE LOGICAL
- EXCLUSIVE OR
- MOVE
- OR
- TEST UNDER MASK

6. The following additional general instructions are available when the message-security assist is installed:

- CIPHER MESSAGE
- CIPHER MESSAGE WITH CHAINING
- COMPUTE INTERMEDIATE MESSAGE DIGEST
- COMPUTE LAST MESSAGE DIGEST
- COMPUTE MESSAGE AUTHENTICATION CODE

7. The following additional general instructions are available when the extended-translation facility 3 is installed:

- CONVERT UTF-16 TO UTF-32
- CONVERT UTF-32 TO UTF-16
- CONVERT UTF-32 TO UTF-8
- CONVERT UTF-8 TO UTF-32
- SEARCH STRING UNICODE
- TRANSLATE AND TEST REVERSE

Additionally, CONVERT UNICODE TO UTF-8 (CUUTF) and CONVERT UTF-8 TO UNICODE (CUTFU) are renamed to CONVERT UTF-16 TO UTF-8 (CU21) and CONVERT UTF-8 TO UTF-16 (CU12), respectively, in order to conform to the names used by this facility. The old names continue to be recognized.

8. The following additional general instructions are available when the extended-immediate facility is installed:

- ADD IMMEDIATE (AFI, AGFI)
- ADD LOGICAL IMMEDIATE (ALFI, ALGFI)
- AND IMMEDIATE (NIHF, NILF)
- COMPARE IMMEDIATE (CFI, CGFI)
- COMPARE LOGICAL IMMEDIATE (CLFI, CLGFI)
- EXCLUSIVE OR IMMEDIATE (XIHF, XILF)
- FIND LEFTMOST ONE (FLOGR)
- INSERT IMMEDIATE (IIHF, IILF)
- LOAD AND TEST (LT, LTG)
- LOAD BYTE (LBR, LGBR)
- LOAD HALFWORD (LHR, LGHR)
- LOAD IMMEDIATE (LGFI)
- LOAD LOGICAL CHARACTER (LLC, LLCR, LLGCR)
- LOAD LOGICAL HALFWORD (LLGHR, LLH, LLHR)
- LOAD LOGICAL IMMEDIATE (LLIHF, LLILF)
- OR IMMEDIATE (OIHF, OILF)
- SUBTRACT LOGICAL IMMEDIATE (SLFI, SLGFI)

The instructions that are part of the extended-immediate facility are indicated in Figure 7-1 by “EI.”

9. The STORE CLOCK FAST general instruction is available when the store-clock-fast facility is installed.

10. The STORE FACILITY LIST EXTENDED general instruction is available when the store-facility-list-extended facility is installed.

11. When the ETF2-enhancement facility is installed, new functions are provided for the TRANSLATE ONE TO ONE, TRANSLATE ONE TO TWO, TRANSLATE TWO TO ONE, and TRANSLATE TWO TO TWO instructions.

12. When the ETF3-enhancement facility is installed, improved well-formedness checking is provided for the CONVERT UTF-16 TO UTF-32, CONVERT UTF-16 TO UTF-8, CONVERT UTF-8 TO UTF-16, and CONVERT UTF-8 TO UTF-32 instructions.

13. The COMPARE AND SWAP AND STORE general instruction is available if the compare-and-swap-and-store facility is installed.

14. The EXTRACT CPU TIME general instruction is available if the extract-CPU-time facility is installed.

15. The following additional general instructions are available when the parsing-enhancement facility is installed:

- TRANSLATE AND TEST EXTENDED
- TRANSLATE AND TEST REVERSE EXTENDED

16. The EXECUTE RELATIVE LONG general instruction is available when the execute-extension facility is installed.

17. The following additional general instructions are available when the general-instructions-extension facility is installed:

- ADD IMMEDIATE (AGSI, ASI)
- ADD LOGICAL WITH SIGNED IMMEDIATE (ALGSI, ALSI)
- COMPARE AND BRANCH (CGRB, CRB)
- COMPARE AND BRANCH RELATIVE (CGRJ, CRJ)
- COMPARE AND TRAP (CGRT, CRT)

- COMPARE HALFWORD (CGH)
- COMPARE HALFWORD IMMEDIATE (CGHSI, CHHSI, CHSI)
- COMPARE HALFWORD RELATIVE LONG (CGHRL, CHRL)
- COMPARE IMMEDIATE AND BRANCH (CGIB, CIB)
- COMPARE IMMEDIATE AND BRANCH RELATIVE (CGIJ, CIJ)
- COMPARE IMMEDIATE AND TRAP (CIT, CGIT)
- COMPARE LOGICAL AND BRANCH (CLGRB, CLRB)
- COMPARE LOGICAL AND BRANCH RELATIVE (CLGRJ, CLRJ)
- COMPARE LOGICAL AND TRAP (CLGRT, CLRT)
- COMPARE LOGICAL IMMEDIATE (CLFHSI, CLGHSI, CLHHSI)
- COMPARE LOGICAL IMMEDIATE AND BRANCH (CLIB, CLGIB)
- COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (CLGIJ, CLIJ)
- COMPARE LOGICAL IMMEDIATE AND TRAP (CLFIT, CLGIT)
- COMPARE LOGICAL RELATIVE LONG (CLGFRL, CLGHRL, CLGRL, CLHRL, CLRL)
- COMPARE RELATIVE LONG (CGFRL, CGRL, CRL)
- EXTRACT CACHE ATTRIBUTE (ECAG)
- LOAD ADDRESS EXTENDED (LAEY)
- LOAD AND TEST (LTGF)
- LOAD HALFWORD RELATIVE LONG (LGHRL, LHRL)
- LOAD LOGICAL HALFWORD RELATIVE LONG (LLGHRL, LLHRL)
- LOAD LOGICAL RELATIVE LONG (LLGFRL)
- LOAD RELATIVE LONG (LGFRL, LGRL, LRL)
- MOVE (MVHI, MVGHI, MVHHI)
- MULTIPLY (MFY)
- MULTIPLY HALFWORD (MHY)
- MULTIPLY SINGLE IMMEDIATE (MSFI, MSGFI)
- PREFETCH DATA (PFD)
- PREFETCH DATA RELATIVE LONG (PFDRL)
- ROTATE THEN AND SELECTED BITS (RNSBG)
- ROTATE THEN EXCLUSIVE OR SELECTED BITS (RXSBG)

- ROTATE THEN INSERT SELECTED BITS (RISBG)
- ROTATE THEN OR SELECTED BITS (ROSBG)
- STORE HALFWORD RELATIVE LONG (STHRL)
- STORE RELATIVE LONG (STGRL, STRL)

18. The following additional general instructions are available when the high-word facility is installed:

- ADD HIGH (AHHHR, AHHLR)
- ADD IMMEDIATE HIGH (AIH)
- ADD LOGICAL HIGH (ALHHHR, ALHHLR)
- ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (ALSIH, ALSIHN)
- BRANCH RELATIVE ON COUNT HIGH (BRCTH)
- COMPARE HIGH (CHHR, CHLR, CHF)
- COMPARE IMMEDIATE HIGH (CIH)
- COMPARE LOGICAL HIGH (CLHHR, CLHLR, CLHF)
- COMPARE LOGICAL IMMEDIATE HIGH (CLIH)
- LOAD BYTE HIGH (LBH)
- LOAD HALFWORD HIGH (LHH)
- LOAD HIGH (LFH)
- LOAD LOGICAL CHARACTER HIGH (LLCH)
- LOAD LOGICAL HALFWORD HIGH (LLHH)
- ROTATE THEN INSERT SELECTED BITS HIGH (RISBHG)
- ROTATE THEN INSERT SELECTED BITS LOW (RISBLG)
- STORE CHARACTER HIGH (STCH)
- STORE HALFWORD HIGH (STHH)
- STORE HIGH (STFH)
- SUBTRACT HIGH (SHHHR, SHHLR)
- SUBTRACT LOGICAL HIGH (SLHHHR, SLHHLR)

19. The following additional general instructions are available when the interlocked-access facility is installed:

- LOAD AND ADD (LAA, LAAG)
- LOAD AND ADD LOGICAL (LAAL, LAALG)
- LOAD AND AND (LAN, LANG)
- LOAD AND EXCLUSIVE OR (LAX, LAXG)
- LOAD AND OR (LAO, LAOG)
- LOAD PAIR DISJOINT (LPD, LPDG)

20. The following additional general instructions are available when the load/store-on-condition facility is installed:

- LOAD ON CONDITION (LOC, LOCG, LOCGR, LOCR)
- STORE ON CONDITION (STOC, STOCG)

21. The following additional general instructions are available when the distinct-operands facility is installed:

- ADD (ARK, AGRK)
- ADD IMMEDIATE (AHIK, AGHIK)
- ADD LOGICAL (ALRK, ALGRK)
- ADD LOGICAL WITH SIGNED IMMEDIATE (ALHSIK, ALGHSIK)
- AND (NRK, NGRK)
- EXCLUSIVE OR (XRK, XGRK)
- OR (ORK, OGRK)
- SHIFT LEFT SINGLE (SLAK)
- SHIFT LEFT SINGLE LOGICAL (SLLK)
- SHIFT RIGHT SINGLE (SRAK)

- SHIFT RIGHT SINGLE LOGICAL (SRLK)
- SUBTRACT (SRK, SGRK)
- SUBTRACT LOGICAL (SLRK, SLGRK)

22. The POPULATION COUNT general instruction is available when the population-count facility is installed.

23. The following additional general instructions are available when the message-security-assist extension 4 is installed:

- CIPHER MESSAGE WITH CFB
- CIPHER MESSAGE WITH COUNTER
- CIPHER MESSAGE WITH OFB
- PERFORM CRYPTOGRAPHIC COMPUTATION

Name	Mnemonic	Characteristics						Op-code	Page
ADD (32)	A	RX-a C	A	IF			B ₂	5A	7-25
ADD (32)	AR	RR C		IF				1A	7-24
ADD (32)	ARK	RRF-a C DO		IF				B9F8	7-25
ADD (32)	AY	RXY-a C LD	A	IF			B ₂	E35A	7-25
ADD (64)	AG	RXY-a C N	A	IF			B ₂	E308	7-25
ADD (64)	AGR	RRE C N		IF				B908	7-25
ADD (64)	AGRK	RRF-a C DO		IF				B9E8	7-25
ADD (64←32)	AGF	RXY-a C N	A	IF			B ₂	E318	7-25
ADD (64←32)	AGFR	RRE C N		IF				B918	7-25
ADD HALFWORD	AH	RX-a C	A	IF			B ₂	4A	7-27
ADD HALFWORD	AHY	RXY-a C LD	A	IF			B ₂	E37A	7-27
ADD HALFWORD IMMEDIATE (32)	AHI	RI-a C		IF				A7A	7-27
ADD HALFWORD IMMEDIATE (64)	AGHI	RI-a C N		IF				A7B	7-27
ADD HIGH (32)	AHHHR	RRF-a C HW		IF				B9C8	7-27
ADD HIGH (32)	AHHLR	RRF-a C HW		IF				B9D8	7-27
ADD IMMEDIATE (32)	AFI	RIL-a C EI		IF				C29	7-25
ADD IMMEDIATE (32←16)	AHIK	RIE-d C DO		IF				ECD8	7-25
ADD IMMEDIATE (32←8)	ASI	SIY C GE	A	IF		ST	B ₁	EB6A	7-25
ADD IMMEDIATE (64←16)	AGHIK	RIE-d C DO		IF				ECD9	7-25
ADD IMMEDIATE (64←32)	AGFI	RIL-a C EI		IF				C28	7-25
ADD IMMEDIATE (64←8)	AGSI	SIY C GE	A	IF		ST	B ₁	EB7A	7-25
ADD IMMEDIATE HIGH (32)	AIH	RIL-a C HW		IF				CC8	7-25
ADD LOGICAL (32)	AL	RX-a C	A				B ₂	5E	7-28
ADD LOGICAL (32)	ALR	RR C						1E	7-28
ADD LOGICAL (32)	ALRK	RRF-a C DO						B9FA	7-28
ADD LOGICAL (32)	ALY	RXY-a C LD	A				B ₂	E35E	7-28
ADD LOGICAL (64)	ALG	RXY-a C N	A				B ₂	E30A	7-28
ADD LOGICAL (64)	ALGR	RRE C N						B90A	7-28
ADD LOGICAL (64)	ALGRK	RRF-a C DO						B9EA	7-28
ADD LOGICAL (64←32)	ALGF	RXY-a C N	A				B ₂	E31A	7-28
ADD LOGICAL (64←32)	ALGFR	RRE C N						B91A	7-28
ADD LOGICAL HIGH (32)	ALHHHR	RRF-a C HW						B9CA	7-29
ADD LOGICAL HIGH (32)	ALHHLR	RRF-a C HW						B9DA	7-29
ADD LOGICAL IMMEDIATE (32)	ALFI	RIL-a C EI						C2B	7-28

Figure 7-1. Summary of General Instructions (Part 1 of 12)

Name	Mne- monic	Characteristics						Op- code	Page
ADD LOGICAL IMMEDIATE (64←32)	ALGFI	RIL-a C EI						C2A	7-28
ADD LOGICAL WITH CARRY (32)	ALC	RXY-a C N3	A				B ₂	E398	7-29
ADD LOGICAL WITH CARRY (32)	ALCR	RRE C N3						B998	7-29
ADD LOGICAL WITH CARRY (64)	ALCG	RXY-a C N	A				B ₂	E388	7-29
ADD LOGICAL WITH CARRY (64)	ALCGR	RRE C N						B988	7-29
ADD LOGICAL WITH SIGNED IMMEDIATE (32←16)	ALHSIK	RIE-d C DO						ECDA	7-30
ADD LOGICAL WITH SIGNED IMMEDIATE (32←8)	ALSI	SIY C GE	A			ST	B ₁	EB6E	7-30
ADD LOGICAL WITH SIGNED IMMEDIATE (64←16)	ALGHSIK	RIE-d C DO				ST	B ₁	ECDB	7-30
ADD LOGICAL WITH SIGNED IMMEDIATE (64←8)	ALGSI	SIY C GE	A			ST	B ₁	EB7E	7-30
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	ALSIH	RIL-a C HW						CCA	7-31
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	ALSIHN	RIL-a HW						CCB	7-31
AND (32)	N	RX-a C	A				B ₂	54	7-31
AND (32)	NR	RR C						14	7-31
AND (32)	NRK	RRF-a C DO						B9F4	7-31
AND (32)	NY	RXY-a C LD	A				B ₂	E354	7-32
AND (64)	NG	RXY-a C N	A				B ₂	E380	7-32
AND (64)	NGR	RRE C N						B980	7-31
AND (64)	NGRK	RRF-a C DO						B9E4	7-31
AND (character)	NC	SS-a C	A			ST	B ₁ B ₂	D4	7-32
AND (immediate)	NI	SI C	A			ST	B ₁	94	7-32
AND (immediate)	NIY	SIY C LD	A			ST	B ₁	EB54	7-32
AND IMMEDIATE (high high)	NIHH	RI-a C N						A54	7-32
AND IMMEDIATE (high low)	NIHL	RI-a C N						A55	7-33
AND IMMEDIATE (high)	NIHF	RIL-a C EI						C0A	7-32
AND IMMEDIATE (low high)	NILH	RI-a C N						A56	7-33
AND IMMEDIATE (low low)	NILL	RI-a C N						A57	7-33
AND IMMEDIATE (low)	NILF	RIL-a C EI						C0B	7-33
BRANCH AND LINK	BAL	RX-a				B		45	7-33
BRANCH AND LINK	BALR	RR		T		B		05	7-33
BRANCH AND SAVE	BAS	RX-a				B		4D	7-34
BRANCH AND SAVE	BASR	RR		T		B		0D	7-34
BRANCH AND SAVE AND SET MODE	BASSM	RR		T		B		0C	7-35
BRANCH AND SET MODE	BSM	RR		T		B		0B	7-36
BRANCH ON CONDITION	BC	RX-b				B		47	7-36
BRANCH ON CONDITION	BCR	RR		⊕ ¹		B		07	7-36
BRANCH ON COUNT (32)	BCT	RX-a				B		46	7-37
BRANCH ON COUNT (32)	BCTR	RR				B		06	7-37
BRANCH ON COUNT (64)	BCTG	RXY-a N				B		E346	7-37
BRANCH ON COUNT (64)	BCTGR	RRE N				B		B946	7-37
BRANCH ON INDEX HIGH (32)	BXH	RS-a				B		86	7-38
BRANCH ON INDEX HIGH (64)	BXHG	RSY-a N				B		EB44	7-38
BRANCH ON INDEX LOW OR EQUAL (32)	BXLE	RS-a				B		87	7-38
BRANCH ON INDEX LOW OR EQUAL (64)	BXLEG	RSY-a N				B		EB45	7-38
BRANCH RELATIVE AND SAVE	BRAS	RI-b				B		A75	7-39
BRANCH RELATIVE AND SAVE LONG	BRASL	RIL-b N3				B		C05	7-39
BRANCH RELATIVE ON CONDITION	BRC	RI-c				B		A74	7-40
BRANCH RELATIVE ON CONDITION LONG	BRCL	RIL-c N3				B		C04	7-40
BRANCH RELATIVE ON COUNT (32)	BRCT	RI-b				B		A76	7-41
BRANCH RELATIVE ON COUNT (64)	BRCTG	RI-b N				B		A77	7-41
BRANCH RELATIVE ON COUNT HIGH (32)	BRCTH	RIL-b HW				B		CC6	7-41
BRANCH RELATIVE ON INDEX HIGH (32)	BRXH	RSI				B		84	7-41
BRANCH RELATIVE ON INDEX HIGH (64)	BRXHG	RIE-e N				B		EC44	7-41
BRANCH RELATIVE ON INDEX LOW OR EQ. (32)	BRXLE	RSI				B		85	7-41
BRANCH RELATIVE ON INDEX LOW OR EQ. (64)	BRXLG	RIE-e N				B		EC45	7-42
CHECKSUM	CKSM	RRE C	A SP IC				R ₂	B241	7-43

Figure 7-1. Summary of General Instructions (Part 2 of 12)

Name	Mne- monic	Characteristics										Op- code	Page	
CIPHER MESSAGE	KM	RRE	C	MS	A	SP	IC	GM	I1	ST	R ₁	R ₂	B92E	7-46
CIPHER MESSAGE WITH CHAINING	KMC	RRE	C	MS	A	SP	IC	GM	I1	ST	R ₁	R ₂	B92F	7-46
CIPHER MESSAGE WITH CFB	KMF	RRE	C	M4	A	SP	IC	GM	I1	ST	R ₁	R ₂	B92A	7-71
CIPHER MESSAGE WITH COUNTER	KMCTR	RRF-b	C	M4	A	SP	IC	GM	I1	ST	R ₁ ,R ₂ ,R ₃		B92D	7-86
CIPHER MESSAGE WITH OFB	KMO	RRE	C	M4	A	SP	IC	GM	I1	ST	R ₁	R ₂	B92B	7-98
COMPARE (32)	C	RX-a	C		A							B ₂	59	7-112
COMPARE (32)	CR	RR	C										19	7-112
COMPARE (32)	CY	RXY-a	C	LD	A							B ₂	E359	7-112
COMPARE (64)	CG	RXY-a	C	N	A							B ₂	E320	7-112
COMPARE (64)	CGR	RRE	C	N									B920	7-112
COMPARE (64←32)	CGF	RXY-a	C	N	A							B ₂	E330	7-112
COMPARE (64←32)	CGFR	RRE	C	N									B930	7-112
COMPARE AND BRANCH (32)	CRB	RRS		GE						B			ECF6	7-113
COMPARE AND BRANCH (64)	CGRB	RRS		GE						B			ECE4	7-113
COMPARE AND BRANCH RELATIVE (32)	CRJ	RIE-b		GE						B			EC76	7-113
COMPARE AND BRANCH RELATIVE (64)	CGRJ	RIE-b		GE						B			EC64	7-113
COMPARE AND FORM CODEWORD	CFC	S	C		A	SP	II	GM	I1				B21A	7-115
COMPARE AND SWAP (32)	CS	RS-a	C		A	SP		\$		ST		B ₂	BA	7-122
COMPARE AND SWAP (32)	CSY	RSY-a	C	LD	A	SP		\$		ST		B ₂	EB14	7-122
COMPARE AND SWAP (64)	CSG	RSY-a	C	N	A	SP		\$		ST		B ₂	EB30	7-122
COMPARE AND SWAP AND STORE	CSST	SSF	C	CS	A	SP		\$	GM	ST	B ₁	B ₂	C82	7-124
COMPARE AND TRAP (32)	CRT	RRF-c		GE				Dc					B972	7-127
COMPARE AND TRAP (64)	CGRT	RRF-c		GE				Dc					B960	7-127
COMPARE DOUBLE AND SWAP (32)	CDS	RS-a	C		A	SP		\$		ST		B ₂	BB	7-122
COMPARE DOUBLE AND SWAP (32)	CDSY	RSY-a	C	LD	A	SP		\$		ST		B ₂	EB31	7-122
COMPARE DOUBLE AND SWAP (64)	CDSG	RSY-a	C	N	A	SP		\$		ST		B ₂	EB3E	7-122
COMPARE HALFWORD (32←16)	CH	RX-a	C		A							B ₂	49	7-128
COMPARE HALFWORD (32←16)	CHY	RXY-a	C	LD	A							B ₂	E379	7-128
COMPARE HALFWORD (64←16)	CGH	RXY-a	C	GE	A							B ₂	E334	7-128
COMPARE HALFWORD IMMEDIATE (16)	CHHSI	SIL	C	GE	A						B ₁		E554	7-128
COMPARE HALFWORD IMMEDIATE (32←16)	CHI	RI-a	C										A7E	7-128
COMPARE HALFWORD IMMEDIATE (32←16)	CHSI	SIL	C	GE	A						B ₁		E55C	7-128
COMPARE HALFWORD IMMEDIATE (64←16)	CGHI	RI-a	C	N									A7F	7-128
COMPARE HALFWORD IMMEDIATE (64←16)	CGHSI	SIL	C	GE	A						B ₁		E558	7-128
COMPARE HALFWORD RELATIVE LONG (32←16)	CHRL	RIL-b	C	GE	A								C65	7-128
COMPARE HALFWORD RELATIVE LONG (64←16)	CGHRL	RIL-b	C	GE	A								C64	7-128
COMPARE HIGH (32)	CHF	RXY-a	C	HW	A							B ₂	E3CD	7-129
COMPARE HIGH (32)	CHHR	RRE	C	HW									B9CD	7-129
COMPARE HIGH (32)	CHLR	RRE	C	HW									B9DD	7-129
COMPARE IMMEDIATE (32)	CFI	RIL-a	C	EI									C2D	7-112
COMPARE IMMEDIATE (64←32)	CGFI	RIL-a	C	EI									C2C	7-112
COMPARE IMMEDIATE AND BRANCH (32←8)	CIB	RIS		GE						B			ECFE	7-113
COMPARE IMMEDIATE AND BRANCH (64←8)	CGIB	RIS		GE						B			ECFC	7-113
COMPARE IMMEDIATE AND BRANCH RELATIVE (32←8)	CIJ	RIE-c		GE						B			EC7E	7-114
COMPARE IMMEDIATE AND BRANCH RELATIVE (64←8)	CGIJ	RIE-c		GE						B			EC7C	7-114
COMPARE IMMEDIATE AND TRAP (32←16)	CIT	RIE-a		GE				Dc					EC72	7-127
COMPARE IMMEDIATE AND TRAP (64←16)	CGIT	RIE-a		GE				Dc					EC70	7-127
COMPARE IMMEDIATE HIGH (32)	CIH	RIL-a	C	HW								B ₂	CCD	7-129
COMPARE LOGICAL (32)	CL	RX-a	C		A								55	7-130
COMPARE LOGICAL (32)	CLR	RR	C										15	7-130
COMPARE LOGICAL (32)	CLY	RXY-a	C	LD	A							B ₂	E355	7-130
COMPARE LOGICAL (64)	CLG	RXY-a	C	N	A							B ₂	E321	7-130
COMPARE LOGICAL (64)	CLGR	RRE	C	N									B921	7-130

Figure 7-1. Summary of General Instructions (Part 3 of 12)

Name	Mne- monic	Characteristics						Op- code	Page
COMPARE LOGICAL (64←32)	CLGF	RXY-a C N	A				B ₂	E331	7-130
COMPARE LOGICAL (64←32)	CLGFR	RRE C N						B931	7-130
COMPARE LOGICAL (character)	CLC	SS-a C	A				B ₁ B ₂	D5	7-130
COMPARE LOGICAL (immediate)	CLI	SI C	A				B ₁	95	7-130
COMPARE LOGICAL (immediate)	CLiy	SIY C LD	A				B ₁	EB55	7-130
COMPARE LOGICAL AND BRANCH (32)	CLRB	RRS GE				B		ECF7	7-132
COMPARE LOGICAL AND BRANCH (64)	CLGRB	RRS GE				B		ECE5	7-132
COMPARE LOGICAL AND BRANCH RELATIVE (32)	CLRJ	RIE-b GE				B		EC77	7-132
COMPARE LOGICAL AND BRANCH RELATIVE (64)	CLGRJ	RIE-b GE				B		EC65	7-132
COMPARE LOGICAL AND TRAP (32)	CLRT	RRF-c GE			Dc			B973	7-133
COMPARE LOGICAL AND TRAP (64)	CLGRT	RRF-c GE			Dc			B961	7-133
COMPARE LOGICAL CHAR. UNDER MASK (high)	CLMH	RSY-b C N	A				B ₂	EB20	7-134
COMPARE LOGICAL CHAR. UNDER MASK (low)	CLM	RS-b C	A				B ₂	BD	7-134
COMPARE LOGICAL CHAR. UNDER MASK (low)	CLMY	RSY-b C LD	A				B ₂	EB21	7-134
COMPARE LOGICAL HIGH (32)	CLHF	RXY-a C HW	A				B ₂	E3CF	7-135
COMPARE LOGICAL HIGH (32)	CLHHR	RRE C HW						B9CF	7-135
COMPARE LOGICAL HIGH (32)	CLHLR	RRE C HW						B9DF	7-135
COMPARE LOGICAL IMMEDIATE (16)	CLHHSI	SIL C GE	A				B ₁	E555	7-130
COMPARE LOGICAL IMMEDIATE (32)	CLFI	RIL-a C EI						C2F	7-130
COMPARE LOGICAL IMMEDIATE (32←16)	CLFHSI	SIL C GE	A				B ₁	E55D	7-130
COMPARE LOGICAL IMMEDIATE (64←16)	CLGHSI	SIL C GE	A				B ₁	E559	7-130
COMPARE LOGICAL IMMEDIATE (64←32)	CLGFI	RIL-a C EI						C2E	7-130
COMPARE LOGICAL IMMEDIATE AND BRANCH (32←8)	CLIB	RIS GE				B		ECFF	7-132
COMPARE LOGICAL IMMEDIATE AND BRANCH (64←8)	CLGIB	RIS GE				B		ECFD	7-132
COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (32←8)	CLIJ	RIE-c GE				B		EC7F	7-132
COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (64←8)	CLGIJ	RIE-c GE				B		EC7D	7-132
COMPARE LOGICAL IMMEDIATE AND TRAP (32←16)	CLFIT	RIE-a GE			Dc			EC73	7-133
COMPARE LOGICAL IMMEDIATE AND TRAP (64←16)	CLGIT	RIE-a GE			Dc			EC71	7-133
COMPARE LOGICAL IMMEDIATE HIGH (32)	CLIH	RIL-a C HW						CCF	7-135
COMPARE LOGICAL LONG	CLCL	RR C	A SP	II			R ₁ R ₂	0F	7-135
COMPARE LOGICAL LONG EXTENDED	CLCLE	RS-a C	A SP	IC			R ₁ R ₃	A9	7-138
COMPARE LOGICAL LONG UNICODE	CLCLU	RSY-a C E2	A SP	IC			R ₁ R ₂	EB8F	7-140
COMPARE LOGICAL RELATIVE LONG (32)	CLRL	RIL-b C GE	A SP					C6F	7-130
COMPARE LOGICAL RELATIVE LONG (32←16)	CLHRL	RIL-b C GE	A					C67	7-131
COMPARE LOGICAL RELATIVE LONG (64)	CLGRL	RIL-b C GE	A SP					C6A	7-131
COMPARE LOGICAL RELATIVE LONG (64←16)	CLGHRL	RIL-b C GE	A					C66	7-131
COMPARE LOGICAL RELATIVE LONG (64←32)	CLGFRL	RIL-b C GE	A SP					C6E	7-131
COMPARE LOGICAL STRING	CLST	RRE C	A SP	IC	G0		R ₁ R ₂	B25D	7-143
COMPARE RELATIVE LONG (32)	CRL	RIL-b C GE	A SP					C6D	7-112
COMPARE RELATIVE LONG (64)	CGRL	RIL-b C GE	A SP					C68	7-112
COMPARE RELATIVE LONG (64←32)	CGFRL	RIL-b C GE	A SP					C6C	7-112
COMPARE UNTIL SUBSTRING EQUAL	CUSE	RRE C	A SP	II	GM		R ₁ R ₂	B257	7-144
COMPRESSION CALL	CMPSC	RRE C	A SP	II	Dd GM	ST	R ₁ R ₂	B263	7-147
COMPUTE INTERMEDIATE MESSAGE DIGEST	KIMD	RRE C MS	A SP	IC	GM I1	ST	R ₂	B93E	7-159
COMPUTE LAST MESSAGE DIGEST	KLMD	RRE C MS	A SP	IC	GM I1	ST	R ₂	B93F	7-159
COMPUTE MESSAGE AUTHENTICATION CODE	KMAC	RRE C MS	A SP	IC	GM I1	ST	R ₂	B91E	7-171
CONVERT TO BINARY (32)	CVB	RX-a	A		Dd IK		B ₂	4F	7-181
CONVERT TO BINARY (32)	CVBY	RXY-a LD	A		Dd IK		B ₂	E306	7-181
CONVERT TO BINARY (64)	CVBG	RXY-a N	A		Dd IK		B ₂	E30E	7-181
CONVERT TO DECIMAL (32)	CVD	RX-a	A			ST	B ₂	4E	7-182

Figure 7-1. Summary of General Instructions (Part 4 of 12)

Name	Mnemonic	Characteristics						Op-code	Page
CONVERT TO DECIMAL (32)	CVDY	RXY-a	LD	A		ST	B ₂	E326	7-182
CONVERT TO DECIMAL (64)	CVDG	RXY-a	N	A		ST	B ₂	E32E	7-182
CONVERT UNICODE TO UTF-8	CUUTF	RRF-c	C	A	SP	ST	R ₁	B2A6	7-186
CONVERT UTF-16 TO UTF-32	CU24	RRF-c	C	E3	A	SP	IC	B9B1	7-183
CONVERT UTF-16 TO UTF-8	CU21	RRF-c	C		A	SP	IC	B2A6	7-186
CONVERT UTF-32 TO UTF-16	CU42	RRE	C	E3	A	SP	IC	B9B3	7-189
CONVERT UTF-32 TO UTF-8	CU41	RRE	C	E3	A	SP	IC	B9B2	7-192
CONVERT UTF-8 TO UNICODE	CUTFU	RRF-c	C		A	SP	IC	B2A7	7-195
CONVERT UTF-8 TO UTF-16	CU12	RRF-c	C		A	SP	IC	B2A7	7-195
CONVERT UTF-8 TO UTF-32	CU14	RRF-c	C	E3	A	SP	IC	B9B0	7-199
COPY ACCESS	CPYA	RRE					U ₁	B24D	7-203
DIVIDE (32←64)	D	RX-a		A	SP		B ₂	5D	7-203
DIVIDE (32←64)	DR	RR			SP			1D	7-203
DIVIDE LOGICAL (32←64)	DL	RXY-a	N3	A	SP		B ₂	E397	7-204
DIVIDE LOGICAL (32←64)	DLR	RRE	N3		SP			B997	7-204
DIVIDE LOGICAL (64←128)	DLG	RXY-a	N	A	SP		B ₂	E387	7-204
DIVIDE LOGICAL (64←128)	DLGR	RRE	N		SP			B987	7-204
DIVIDE SINGLE (64)	DSG	RXY-a	N	A	SP		B ₂	E30D	7-205
DIVIDE SINGLE (64)	DSGR	RRE	N		SP			B90D	7-204
DIVIDE SINGLE (64←32)	DSGF	RXY-a	N	A	SP		B ₂	E31D	7-205
DIVIDE SINGLE (64←32)	DSGFR	RRE	N		SP			B91D	7-204
EXCLUSIVE OR (32)	X	RX-a	C	A			B ₂	57	7-205
EXCLUSIVE OR (32)	XR	RR	C					17	7-205
EXCLUSIVE OR (32)	XRK	RRF-a	C	DO				B9F7	7-205
EXCLUSIVE OR (32)	XY	RXY-a	C	LD	A		B ₂	E357	7-205
EXCLUSIVE OR (64)	XG	RXY-a	C	N	A		B ₂	E382	7-205
EXCLUSIVE OR (64)	XGR	RRE	C	N				B982	7-205
EXCLUSIVE OR (64)	XGRK	RRF-a	C	DO				B9E7	7-205
EXCLUSIVE OR (character)	XC	SS-a	C		A	ST	B ₁ B ₂	D7	7-206
EXCLUSIVE OR (immediate)	XI	SI	C		A	ST	B ₁	97	7-205
EXCLUSIVE OR (immediate)	XIY	SIY	C	LD	A	ST	B ₁	EB57	7-205
EXCLUSIVE OR IMMEDIATE (high)	XIHF	RIL-a	C	EI				C06	7-206
EXCLUSIVE OR IMMEDIATE (low)	XILF	RIL-a	C	EI				C07	7-206
EXECUTE	EX	RX-a		AI	SP			44	7-207
EXECUTE RELATIVE LONG	EXRL	RIL-b	XX	AI				C60	7-207
EXTRACT ACCESS	EAR	RRE					U ₂	B24F	7-208
EXTRACT CACHE ATTRIBUTE	ECAG	RSY-a	GE					EB4C	7-208
EXTRACT CPU TIME	ECTG	SSF	ET	A		R ₃	B ₁ B ₂	C81	7-210
EXTRACT PSW	EPSW	RRE	N3					B98D	7-210
FIND LEFTMOST ONE	FLOGR	RRE	C	EI	SP			B983	7-211
INSERT CHARACTER	IC	RX-a		A			B ₂	43	7-211
INSERT CHARACTER	ICY	RXY-a	LD	A			B ₂	E373	7-211
INSERT CHARACTERS UNDER MASK (high)	ICMH	RSY-b	C	N	A		B ₂	EB80	7-212
INSERT CHARACTERS UNDER MASK (low)	ICM	RS-b	C		A		B ₂	BF	7-212
INSERT CHARACTERS UNDER MASK (low)	ICMY	RSY-b	C	LD	A		B ₂	EB81	7-212
INSERT IMMEDIATE (high high)	IIHH	RI-a	N					A50	7-212
INSERT IMMEDIATE (high low)	IIHL	RI-a	N					A51	7-213
INSERT IMMEDIATE (high)	IIHF	RIL-a	EI					C08	7-212
INSERT IMMEDIATE (low high)	IILH	RI-a	N					A52	7-213
INSERT IMMEDIATE (low low)	IILL	RI-a	N					A53	7-213
INSERT IMMEDIATE (low)	IILF	RIL-a	EI					C09	7-213
INSERT PROGRAM MASK	IPM	RRE						B222	7-213
LOAD (32)	L	RX-a		A			B ₂	58	7-213
LOAD (32)	LR	RR						18	7-213
LOAD (32)	LY	RXY-a	LD	A			B ₂	E358	7-213

Figure 7-1. Summary of General Instructions (Part 5 of 12)

Name	Mne- monic	Characteristics						Op- code	Page
LOAD (64)	LG	RXY-a	N	A			B ₂	E304	7-213
LOAD (64)	LGR	RRE	N					B904	7-213
LOAD (64←32)	LGF	RXY-a	N	A			B ₂	E314	7-213
LOAD (64←32)	LGFR	RRE	N					B914	7-213
LOAD ACCESS MULTIPLE	LAM	RS-a		A	SP		UB	9A	7-214
LOAD ACCESS MULTIPLE	LAMY	RSY-a	LD	A	SP		UB	EB9A	7-214
LOAD ADDRESS	LA	RX-a						41	7-215
LOAD ADDRESS	LAY	RXY-a	LD					E371	7-215
LOAD ADDRESS EXTENDED	LAE	RX-a					U ₁	BP	51
LOAD ADDRESS EXTENDED	LAEY	RXY-a	GE				U ₁	BP	E375
LOAD ADDRESS RELATIVE LONG	LARL	RIL-b	N3					C00	7-216
LOAD AND ADD (32)	LAA	RSY-a	C	IA	A	SP	IF	ST	B ₂
LOAD AND ADD (64)	LAAG	RSY-a	C	IA	A	SP	IF	ST	B ₂
LOAD AND ADD LOGICAL (32)	LAAL	RSY-a	C	IA	A	SP		ST	B ₂
LOAD AND ADD LOGICAL (64)	LAALG	RSY-a	C	IA	A	SP		ST	B ₂
LOAD AND AND (32)	LAN	RSY-a	C	IA	A	SP		ST	B ₂
LOAD AND AND (64)	LANG	RSY-a	C	IA	A	SP		ST	B ₂
LOAD AND EXCLUSIVE OR (32)	LAX	RSY-a	C	IA	A	SP		ST	B ₂
LOAD AND EXCLUSIVE OR (64)	LAXG	RSY-a	C	IA	A	SP		ST	B ₂
LOAD AND OR (32)	LAO	RSY-a	C	IA	A	SP		ST	B ₂
LOAD AND OR (64)	LAOG	RSY-a	C	IA	A	SP		ST	B ₂
LOAD AND TEST (32)	LT	RXY-a	C	EI	A			B ₂	E312
LOAD AND TEST (32)	LTR	RR	C					12	7-219
LOAD AND TEST (64)	LTG	RXY-a	C	EI	A			B ₂	E302
LOAD AND TEST (64)	LTGR	RRE	C	N					B902
LOAD AND TEST (64←32)	LTGF	RXY-a	C	GE	A			B ₂	E332
LOAD AND TEST (64←32)	LTGFR	RRE	C	N					B912
LOAD BYTE (32)	LB	RXY-a	LD	A					E376
LOAD BYTE (32)	LBR	RRE	EI						B926
LOAD BYTE (64)	LGB	RXY-a	LD	A					E377
LOAD BYTE (64)	LGBR	RRE	EI						B906
LOAD BYTE HIGH (32←8)	LBH	RXY-a	HW	A				B ₂	E3C0
LOAD COMPLEMENT (32)	LCR	RR	C			IF		13	7-221
LOAD COMPLEMENT (64)	LCGR	RRE	C	N		IF			B903
LOAD COMPLEMENT (64←32)	LCGFR	RRE	C	N					B913
LOAD HALFWORD (32)	LH	RX-a		A				B ₂	48
LOAD HALFWORD (32)	LHR	RRE	EI						B927
LOAD HALFWORD (32)	LHY	RXY-a	LD	A				B ₂	E378
LOAD HALFWORD (64)	LGH	RXY-a	N	A				B ₂	E315
LOAD HALFWORD (64)	LGHR	RRE	EI						B907
LOAD HALFWORD HIGH (32←16)	LHH	RXY-a	HW	A				B ₂	E3C4
LOAD HALFWORD IMMEDIATE (32)	LHI	RI-a							A78
LOAD HALFWORD IMMEDIATE (64)	LGHI	RI-a	N						A79
LOAD HALFWORD RELATIVE LONG (32←16)	LHRL	RIL-b	GE	A					C45
LOAD HALFWORD RELATIVE LONG (64←16)	LGHRL	RIL-b	GE	A					C44
LOAD HIGH (32)	LFH	RXY-a	HW	A				B ₂	E3CA
LOAD IMMEDIATE (64←32)	LGFI	RIL-a	EI						C01
LOAD LOGICAL (64←32)	LLGF	RXY-a	N	A				B ₂	E316
LOAD LOGICAL (64←32)	LLGFR	RRE	N						B916
LOAD LOGICAL CHARACTER (32)	LLC	RXY-a	EI	A				B ₂	E394
LOAD LOGICAL CHARACTER (32)	LLCR	RRE	EI						B994
LOAD LOGICAL CHARACTER (64)	LLGC	RXY-a	N	A				B ₂	E390
LOAD LOGICAL CHARACTER (64)	LLGCR	RRE	EI						B984
LOAD LOGICAL CHARACTER HIGH (32←8)	LLCH	RXY-a	HW	A				B ₂	E3C2
LOAD LOGICAL HALFWORD (32)	LLH	RXY-a	EI	A				B ₂	E395

Figure 7-1. Summary of General Instructions (Part 6 of 12)

Name	Mnemonic	Characteristics						Op-code	Page
LOAD LOGICAL HALFWORD (32)	LLHR	RRE	EI					B995	7-224
LOAD LOGICAL HALFWORD (64)	LLGH	RXY-a	N	A			B ₂	E391	7-224
LOAD LOGICAL HALFWORD (64)	LLGHR	RRE	EI					B985	7-224
LOAD LOGICAL HALFWORD HIGH (32←16)	LLHH	RXY-a	HW	A			B ₂	E3C6	7-225
LOAD LOGICAL HALFWORD RELATIVE LONG (32←16)	LLHRL	RIL-b	GE	A				C42	7-224
LOAD LOGICAL HALFWORD RELATIVE LONG (64←16)	LLGHRL	RIL-b	GE	A				C46	7-224
LOAD LOGICAL IMMEDIATE (high high)	LLIHH	RI-a	N					A5C	7-225
LOAD LOGICAL IMMEDIATE (high low)	LLIHL	RI-a	N					A5D	7-225
LOAD LOGICAL IMMEDIATE (high)	LLIHF	RIL-a	EI					C0E	7-225
LOAD LOGICAL IMMEDIATE (low high)	LLILH	RI-a	N					A5E	7-225
LOAD LOGICAL IMMEDIATE (low low)	LLILL	RI-a	N					A5F	7-225
LOAD LOGICAL IMMEDIATE (low)	LLILF	RIL-a	EI					C0F	7-225
LOAD LOGICAL RELATIVE LONG (64←32)	LLGFRL	RIL-b	GE	A	SP			C4E	7-223
LOAD LOGICAL THIRTY ONE BITS	LLGT	RXY-a	N	A			B ₂	E317	7-226
LOAD LOGICAL THIRTY ONE BITS	LLGTR	RRE	N					B917	7-226
LOAD MULTIPLE (32)	LM	RS-a		A			B ₂	98	7-226
LOAD MULTIPLE (32)	LMY	RSY-a	LD	A			B ₂	EB98	7-226
LOAD MULTIPLE (64)	LMG	RSY-a	N	A			B ₂	EB04	7-226
LOAD MULTIPLE DISJOINT	LMD	SS-e	N	A			B ₂ B ₄	EF	7-227
LOAD MULTIPLE HIGH	LMH	RSY-a	N	A			B ₂	EB96	7-227
LOAD NEGATIVE (32)	LNR	RR	C					11	7-227
LOAD NEGATIVE (64)	LNGR	RRE	C N					B901	7-227
LOAD NEGATIVE (64←32)	LNGFR	RRE	C N					B911	7-227
LOAD ON CONDITION (32)	LOC	RSY-b	CL	A			B ₂	EBF2	7-228
LOAD ON CONDITION (32)	LOCR	RRF-c	CL	A				B9F2	7-228
LOAD ON CONDITION (64)	LOCG	RSY-b	CL	A			B ₂	EBE2	7-228
LOAD ON CONDITION (64)	LOCGR	RRF-c	CL	A				B9E2	7-228
LOAD PAIR DISJOINT (32)	LPD	SSF	C IA	A	SP		B ₁ B ₂	C84	7-229
LOAD PAIR DISJOINT (64)	LPDG	SSF	C IA	A	SP		B ₁ B ₂	C85	7-229
LOAD PAIR FROM QUADWORD	LPQ	RXY-a	N	A	SP		B ₂	E38F	7-230
LOAD POSITIVE (32)	LPR	RR	C			IF		10	7-230
LOAD POSITIVE (64)	LPGR	RRE	C N			IF		B900	7-230
LOAD POSITIVE (64←32)	LPGFR	RRE	C N					B910	7-230
LOAD RELATIVE LONG (32)	LRL	RIL-b	GE	A	SP			C4D	7-214
LOAD RELATIVE LONG (64)	LGRL	RIL-b	GE	A	SP			C48	7-214
LOAD RELATIVE LONG (64←32)	LGFRL	RIL-b	GE	A	SP			C4C	7-214
LOAD REVERSED (16)	LRVH	RXY-a	N3	A			B ₂	E31F	7-231
LOAD REVERSED (32)	LRV	RXY-a	N3	A			B ₂	E31E	7-231
LOAD REVERSED (32)	LRVR	RRE	N3					B91F	7-230
LOAD REVERSED (64)	LRVG	RXY-a	N	A			B ₂	E30F	7-231
LOAD REVERSED (64)	LRVGR	RRE	N					B90F	7-231
MONITOR CALL	MC	SI			SP	ME		AF	7-231
MOVE (16←16)	MVHHI	SIL	GE	A			ST B ₁	E544	7-232
MOVE (32←16)	MVHI	SIL	GE	A			ST B ₁	E54C	7-233
MOVE (64←16)	MVGHI	SIL	GE	A			ST B ₁	E548	7-233
MOVE (character)	MVC	SS-a		A			ST B ₁ B ₂	D2	7-232
MOVE (immediate)	MVI	SI		A			ST B ₁	92	7-233
MOVE (immediate)	MVIY	SIY	LD	A			ST B ₁	EB52	7-233
MOVE INVERSE	MVCIN	SS-a		A			ST B ₁ B ₂	E8	7-233
MOVE LONG	MVCL	RR	C	A	SP	II	ST R ₁ R ₂	0E	7-234
MOVE LONG EXTENDED	MVCLE	RS-a	C	A	SP	IC	ST R ₁ R ₃	A8	7-238
MOVE LONG UNICODE	MVCLU	RSY-a	C E2	A	SP	IC	ST R ₁ R ₃	EB8E	7-241
MOVE NUMERICS	MVN	SS-a		A			ST B ₁ B ₂	D1	7-244

Figure 7-1. Summary of General Instructions (Part 7 of 12)

Name	Mne- monic	Characteristics							Op- code	Page
MOVE STRING	MVST	RRE C	A SP	IC	G0	ST	R ₁ R ₂	B255	7-245	
MOVE WITH OFFSET	MVO	SS-b	A			ST	B ₁ B ₂	F1	7-246	
MOVE ZONES	MVZ	SS-a	A			ST	B ₁ B ₂	D3	7-246	
MULTIPLY (64←32)	M	RX-a	A SP				B ₂	5C	7-247	
MULTIPLY (64←32)	MFY	RXY-a GE	A SP				B ₂	E35C	7-247	
MULTIPLY (64←32)	MR	RR	SP					1C	7-247	
MULTIPLY HALFWORD (32)	MH	RX-a	A				B ₂	4C	7-248	
MULTIPLY HALFWORD (32)	MHY	RXY-a GE	A				B ₂	E37C	7-248	
MULTIPLY HALFWORD IMMEDIATE (32)	MHI	RI-a						A7C	7-248	
MULTIPLY HALFWORD IMMEDIATE (64)	MGHI	RI-a N						A7D	7-248	
MULTIPLY LOGICAL (128←64)	MLG	RXY-a N	A SP				B ₂	E386	7-249	
MULTIPLY LOGICAL (128←64)	MLGR	RRE N	SP					B986	7-249	
MULTIPLY LOGICAL (64←32)	ML	RXY-a N3	A SP				B ₂	E396	7-249	
MULTIPLY LOGICAL (64←32)	MLR	RRE N3	SP					B996	7-248	
MULTIPLY SINGLE (32)	MS	RX-a	A				B ₂	71	7-249	
MULTIPLY SINGLE (32)	MSR	RRE						B252	7-249	
MULTIPLY SINGLE (32)	MSY	RXY-a LD	A				B ₂	E351	7-249	
MULTIPLY SINGLE (64)	MSG	RXY-a N	A				B ₂	E30C	7-249	
MULTIPLY SINGLE (64)	MSGR	RRE N						B90C	7-249	
MULTIPLY SINGLE (64←32)	MSGF	RXY-a N	A				B ₂	E31C	7-249	
MULTIPLY SINGLE (64←32)	MSGFR	RRE N						B91C	7-249	
MULTIPLY SINGLE IMMEDIATE (32)	MSFI	RII-a GE						C21	7-250	
MULTIPLY SINGLE IMMEDIATE (64←32)	MSGFI	RII-a GE						C20	7-250	
OR (32)	O	RX-a C	A				B ₂	56	7-250	
OR (32)	OR	RR C						16	7-250	
OR (32)	ORK	RRF-a C DO						B9F6	7-250	
OR (32)	OY	RXY-a C LD	A				B ₂	E356	7-250	
OR (64)	OG	RXY-a C N	A				B ₂	E381	7-250	
OR (64)	OGR	RRE C N						B981	7-250	
OR (64)	OGRK	RRF-a C DO						B9E6	7-250	
OR (character)	OC	SS-a C	A			ST	B ₁ B ₂	D6	7-251	
OR (immediate)	OI	SI C	A			ST	B ₁	96	7-251	
OR (immediate)	OIY	SIY C LD	A			ST	B ₁	EB56	7-251	
OR IMMEDIATE (high high)	OIHH	RI-a C N						A58	7-251	
OR IMMEDIATE (high low)	OIHL	RI-a C N						A59	7-251	
OR IMMEDIATE (high)	OIHF	RII-a C EI						C0C	7-251	
OR IMMEDIATE (low high)	OILH	RI-a C N						A5A	7-252	
OR IMMEDIATE (low low)	OILL	RI-a C N						A5B	7-252	
OR IMMEDIATE (low)	OILF	RII-a C EI						C0D	7-251	
PACK	PACK	SS-b	A			ST	B ₁ B ₂	F2	7-252	
PACK ASCII	PKA	SS-f E2	A SP			ST	B ₁ B ₂	E9	7-253	
PACK UNICODE	PKU	SS-f E2	A SP			ST	B ₁ B ₂	E1	7-254	
PERFORM CRYPTOGRAPHIC COMPUTATION	PCC	RRE C M4	A SP	IC	GM I1	ST		B92C	7-255	
PERFORM LOCKED OPERATION	PLO	SS-e C	A SP		\$ GM	ST	FC	EE	7-270	
POPULATION COUNT	POPCNT	RRE C PK						B9E1	7-284	
PREFETCH DATA	PFD	RXY-b GE					B ₂	E336	7-285	
PREFETCH DATA RELATIVE LONG	PFDR	RII-c GE						C62	7-285	
ROTATE LEFT SINGLE LOGICAL (32)	RLL	RSY-a N3						EB1D	7-286	
ROTATE LEFT SINGLE LOGICAL (64)	RLLG	RSY-a N						EB1C	7-286	
ROTATE THEN AND SELECTED BITS	RNSBG	RIE-f C GE						EC54	7-287	
ROTATE THEN EXCLUSIVE OR SELECTED BITS	RXSBG	RIE-f C GE						EC57	7-287	
ROTATE THEN INSERT SELECTED BITS	RISBG	RIE-f C GE						EC55	7-288	
ROTATE THEN INSERT SELECTED BITS HIGH	RISBGH	RIE-f HW						EC5D	7-290	
ROTATE THEN INSERT SELECTED BITS LOW	RISBLG	RIE-f HW						EC51	7-290	
ROTATE THEN OR SELECTED BITS	ROSBG	RIE-f C GE						EC56	7-287	

Figure 7-1. Summary of General Instructions (Part 8 of 12)

Name	Mnemonic	Characteristics						Op-code	Page
SEARCH STRING	SRST	RRE C	A SP	IC	G0		R ₂	B25E	7-291
SEARCH STRING UNICODE	SRSTU	RRE C E3	A SP	IC	G0		R ₁ R ₂	B9BE	7-293
SET ACCESS	SAR	RRE					U ₁	B24E	7-295
SET ADDRESSING MODE (24)	SAM24	E N3	SP	T				010C	7-295
SET ADDRESSING MODE (31)	SAM31	E N3	SP	T				010D	7-295
SET ADDRESSING MODE (64)	SAM64	E N		T				010E	7-295
SET PROGRAM MASK	SPM	RR L						04	7-296
SHIFT LEFT DOUBLE	SLDA	RS-a C	SP	IF				8F	7-296
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS-a	SP					8D	7-297
SHIFT LEFT SINGLE (32)	SLA	RS-a C		IF				8B	7-297
SHIFT LEFT SINGLE (32)	SLAK	RSY-a C DO		IF				EBDD	7-297
SHIFT LEFT SINGLE (64)	SLAG	RSY-a C N		IF				EB0B	7-297
SHIFT LEFT SINGLE LOGICAL (32)	SLL	RS-a						89	7-298
SHIFT LEFT SINGLE LOGICAL (32)	SLLK	RSY-a DO						EBDF	7-298
SHIFT LEFT SINGLE LOGICAL (64)	SLLG	RSY-a N						EB0D	7-298
SHIFT RIGHT DOUBLE	SRDA	RS-a C	SP					8E	7-299
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS-a	SP					8C	7-299
SHIFT RIGHT SINGLE (32)	SRA	RS-a C						8A	7-300
SHIFT RIGHT SINGLE (32)	SRAK	RSY-a C DO						EBDC	7-300
SHIFT RIGHT SINGLE (64)	SRAG	RSY-a C N						EB0A	7-300
SHIFT RIGHT SINGLE LOGICAL (32)	SRL	RS-a						88	7-301
SHIFT RIGHT SINGLE LOGICAL (32)	SRLK	RSY-a DO						EBDE	7-301
SHIFT RIGHT SINGLE LOGICAL (64)	SRLG	RSY-a N						EB0C	7-301
STORE (32)	ST	RX-a	A			ST	B ₂	50	7-301
STORE (32)	STY	RXY-a LD	A			ST	B ₂	E350	7-301
STORE (64)	STG	RXY-a N	A			ST	B ₂	E324	7-301
STORE ACCESS MULTIPLE	STAM	RS-a	A SP			ST	UB	9B	7-302
STORE ACCESS MULTIPLE	STAMY	RSY-a LD	A SP			ST	UB	EB9B	7-302
STORE CHARACTER	STC	RX-a	A			ST	B ₂	42	7-302
STORE CHARACTER	STCY	RXY-a LD	A			ST	B ₂	E372	7-302
STORE CHARACTER HIGH (8)	STCH	RXY-a HW	A			ST	B ₂	E3C3	7-303
STORE CHARACTERS UNDER MASK (high)	STCMH	RSY-b N	A			ST	B ₂	EB2C	7-303
STORE CHARACTERS UNDER MASK (low)	STCM	RS-b	A			ST	B ₂	BE	7-303
STORE CHARACTERS UNDER MASK (low)	STCMY	RSY-b LD	A			ST	B ₂	EB2D	7-303
STORE CLOCK	STCK	S C	A		\$	ST	B ₂	B205	7-304
STORE CLOCK EXTENDED	STCKE	S C	A		\$	ST	B ₂	B278	7-305
STORE CLOCK FAST	STCKF	S C SC	A			ST	B ₂	B27C	7-304
STORE FACILITY LIST EXTENDED	STFLE	S C FL	A SP		G0	ST	B ₂	B2B0	7-306
STORE HALFWORD	STH	RX-a	A			ST	B ₂	40	7-307
STORE HALFWORD	STHY	RXY-a LD	A			ST	B ₂	E370	7-307
STORE HALFWORD HIGH (16)	STHH	RXY-a HW	A			ST	B ₂	E3C7	7-308
STORE HALFWORD RELATIVE LONG	STHRL	RIL-b GE	A			ST		C47	7-307
STORE HIGH (32)	STFH	RXY-a HW	A			ST	B ₂	E3CB	7-308
STORE MULTIPLE (32)	STM	RS-a	A			ST	B ₂	90	7-308
STORE MULTIPLE (32)	STMY	RSY-a LD	A			ST	B ₂	EB90	7-308
STORE MULTIPLE (64)	STMG	RSY-a N	A			ST	B ₂	EB24	7-308
STORE MULTIPLE HIGH	STMH	RSY-a N	A			ST	B ₂	EB26	7-309
STORE ON CONDITION (32)	STOC	RSY-b CL	A			ST	B ₂	EBF3	7-309
STORE ON CONDITION (64)	STOCG	RSY-b CL	A			ST	B ₂	EBE3	7-309
STORE PAIR TO QUADWORD	STPQ	RXY-a N	A SP			ST	B ₂	E38E	7-310
STORE RELATIVE LONG (32)	STRL	RIL-b GE	A SP			ST		C4F	7-301
STORE RELATIVE LONG (64)	STGRL	RIL-b GE	A SP			ST		C4B	7-301
STORE REVERSED (16)	STRVH	RXY-a N3	A			ST	B ₂	E33F	7-310
STORE REVERSED (32)	STRV	RXY-a N3	A			ST	B ₂	E33E	7-310
STORE REVERSED (64)	STRVG	RXY-a N	A			ST	B ₂	E32F	7-311

Figure 7-1. Summary of General Instructions (Part 9 of 12)

Name	Mne- monic	Characteristics						Op- code	Page
SUBTRACT (32)	S	RX-a C	A	IF		B ₂	5B	7-311	
SUBTRACT (32)	SR	RR C		IF			1B	7-311	
SUBTRACT (32)	SRK	RRF-a C DO		IF			B9F9	7-311	
SUBTRACT (32)	SY	RXY-a C LD	A	IF		B ₂	E35B	7-311	
SUBTRACT (64)	SG	RXY-a C N	A	IF		B ₂	E309	7-311	
SUBTRACT (64)	SGR	RRE C N		IF			B909	7-311	
SUBTRACT (64)	SGRK	RRF-a C DO		IF			B9E9	7-311	
SUBTRACT (64←32)	SGF	RXY-a C N	A	IF		B ₂	E319	7-311	
SUBTRACT (64←32)	SGFR	RRE C N		IF			B919	7-311	
SUBTRACT HALFWORD	SH	RX-a C	A	IF		B ₂	4B	7-312	
SUBTRACT HALFWORD	SHY	RXY-a C LD	A	IF		B ₂	E37B	7-312	
SUBTRACT HIGH (32)	SHHHR	RRF-a C HW		IF			B9C9	7-312	
SUBTRACT HIGH (32)	SHHLR	RRF-a C HW		IF			B9D9	7-312	
SUBTRACT LOGICAL (32)	SL	RX-a C	A			B ₂	5F	7-313	
SUBTRACT LOGICAL (32)	SLR	RR C					1F	7-313	
SUBTRACT LOGICAL (32)	SLRK	RRF-a C DO					B9FB	7-313	
SUBTRACT LOGICAL (32)	SLY	RXY-a C LD	A			B ₂	E35F	7-313	
SUBTRACT LOGICAL (64)	SLG	RXY-a C N	A			B ₂	E30B	7-313	
SUBTRACT LOGICAL (64)	SLGR	RRE C N					B90B	7-313	
SUBTRACT LOGICAL (64)	SLGRK	RRF-a C DO					B9EB	7-313	
SUBTRACT LOGICAL (64←32)	SLGF	RXY-a C N	A			B ₂	E31B	7-313	
SUBTRACT LOGICAL (64←32)	SLGFR	RRE C N					B91B	7-313	
SUBTRACT LOGICAL HIGH (32)	SLHHHR	RRF-a C HW					B9CB	7-314	
SUBTRACT LOGICAL HIGH (32)	SLHHLR	RRF-a C HW					B9DB	7-314	
SUBTRACT LOGICAL IMMEDIATE (32)	SLFI	RIL-a C EI					C25	7-313	
SUBTRACT LOGICAL IMMEDIATE (64←32)	SLGFI	RIL-a C EI					C24	7-313	
SUBTRACT LOGICAL WITH BORROW (32)	SLB	RXY-a C N3	A			B ₂	E399	7-315	
SUBTRACT LOGICAL WITH BORROW (32)	SLBR	RRE C N3					B999	7-314	
SUBTRACT LOGICAL WITH BORROW (64)	SLBG	RXY-a C N	A			B ₂	E389	7-315	
SUBTRACT LOGICAL WITH BORROW (64)	SLBGR	RRE C N					B989	7-314	
SUPERVISOR CALL	SVC	I		¢			0A	7-315	
TEST ADDRESSING MODE	TAM	E C N3					010B	7-315	
TEST AND SET	TS	S C	A	\$		ST B ₂	93	7-316	
TEST UNDER MASK	TM	SI C	A			B ₁	91	7-316	
TEST UNDER MASK	TMY	SIY C LD	A			B ₁	EB51	7-316	
TEST UNDER MASK (high high)	TMHH	RI-a C N					A72	7-316	
TEST UNDER MASK (high low)	TMHL	RI-a C N					A73	7-316	
TEST UNDER MASK (low high)	TMLH	RI-a C N					A70	7-316	
TEST UNDER MASK (low low)	TMLL	RI-a C N					A71	7-316	
TEST UNDER MASK HIGH	TMH	RI-a C					A70	7-316	
TEST UNDER MASK LOW	TML	RI-a C					A71	7-316	
TRANSLATE	TR	SS-a	A			ST B ₁ B ₂	DC	7-317	
TRANSLATE AND TEST	TRT	SS-a C	A	GM		B ₁ B ₂	DD	7-318	
TRANSLATE AND TEST EXTENDED	TRTE	RRF-c C PE	A SP	IC		ST RM	B9BF	7-319	
TRANSLATE AND TEST REVERSE	TRTR	SS-a C E3	A	GM		B ₁ B ₂	D0	7-323	
TRANSLATE AND TEST REVERSE EXTENDED	TRTRE	RRF-c C PE	A SP	IC		ST RM	B9BD	7-319	
TRANSLATE EXTENDED	TRE	RRE C	A SP	IC		ST R ₁ R ₂	B2A5	7-324	
TRANSLATE ONE TO ONE	TROO	RRF-c C E2	A SP	IC	GM	ST RM R ₂	B993	7-327	
TRANSLATE ONE TO TWO	TROT	RRF-c C E2	A SP	IC	GM	ST RM R ₂	B992	7-327	
TRANSLATE TWO TO ONE	TRTO	RRF-c C E2	A SP	IC	GM	ST RM R ₂	B991	7-327	
TRANSLATE TWO TO TWO	TRTT	RRF-c C E2	A SP	IC	GM	ST RM R ₂	B990	7-327	
UNPACK	UNPK	SS-b	A			ST B ₁ B ₂	F3	7-331	
UNPACK ASCII	UNPKA	SS-a C E2	A SP			ST B ₁ B ₂	EA	7-332	
UNPACK UNICODE	UNPKU	SS-a C E2	A SP			ST B ₁ B ₂	E2	7-333	
UPDATE TREE	UPT	E C	A SP	II	GM I4	ST	0102	7-334	

Figure 7-1. Summary of General Instructions (Part 10 of 12)

Name	Mne- monic	Characteristics	Op- code	Page
Explanation:				
¢		Causes serialization and checkpoint synchronization.		
¢ ¹		Causes serialization and checkpoint synchronization when the M ₁ and R ₂ fields contain 1111 binary and 0000 binary, respectively. Causes only serialization when the fast-BCR-serialization facility is installed, and the M ₁ and R ₂ fields contain 1110 binary and 0000 binary, respectively.		
\$		Causes serialization.		
A		Access exceptions for logical addresses.		
AI		Access exceptions for instruction address.		
B		PER branch event.		
B ₁		B ₁ field designates an access register in the access-register mode.		
B ₂		B ₂ field designates an access register in the access-register mode.		
BP		B ₂ field designates an access register when PSW bits 16 and 17 have the value 01 binary.		
C		Condition code is set.		
CL		Load/store-on-condition facility		
CS		Compare-and-swap-and-store facility.		
Dc		Compare-and-trap data exception		
Dd		Decimal-operand data exception.		
DO		Distinct-operands facility		
E		E instruction format.		
EI		Extended-immediate facility.		
ET		Extract-CPU-time facility.		
EX		Execute exception.		
E2		Extended-translation facility 2.		
E3		Extended-translation facility 3.		
FC		Designation of access registers depends on the function code of the instruction.		
FL		Store-facility-list-extended facility.		
G0		Instruction execution includes the implied use of general register 0.		
GE		General-instructions-extension facility.		
GM		Instruction execution includes the implied use of multiple general registers: <ul style="list-style-type: none"> General registers 1, 2, and 3 for COMPARE AND FORM CODEWORD. General registers 0 and 1 for CIPHER MESSAGE, CIPHER MESSAGE WITH CHAINING, CIPHER MESSAGE WITH CFB, CIPHER MESSAGE WITH COUNTER, CIPHER MESSAGE WITH OFB, COMPARE UNTIL SUBSTRING EQUAL, COMPRESSION CALL, COMPUTE INTERMEDIATE MESSAGE DIGEST, COMPUTE LAST MESSAGE DIGEST, COMPUTE MESSAGE AUTHENTICATION CODE, EXTRACT CPU TIME, PERFORM CRYPTOGRAPHIC COMPUTATION, PERFORM LOCKED OPERATION, TRANSLATE ONE TO ONE, TRANSLATE ONE TO TWO, TRANSLATE TWO TO ONE, and TRANSLATE TWO TO TWO General registers 1 and 2 for TRANSLATE AND TEST and TRANSLATE AND TEST REVERSE. General registers 0-5 for UPDATE TREE. 		
HW		High-word facility		
I		I instruction format.		
I1		Access register 1 is implicitly designated in the access-register mode.		
I2		Access register 2 is implicitly designated in the access-register mode.		
I4		Access register 4 is implicitly designated in the access-register mode.		
IA		Interlocked-access facility		
IC		Condition code alternative to interruptible instruction		
IF		Fixed-point-overflow exception.		
II		Interruptible instruction.		
IK		Fixed-point-divide exception.		
L		New condition code is loaded.		

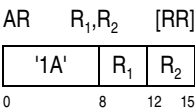
Figure 7-1. Summary of General Instructions (Part 11 of 12)

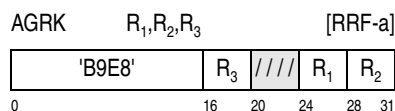
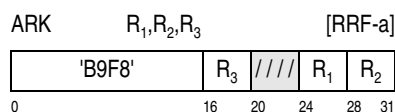
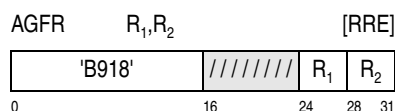
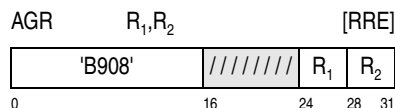
	Name	Mne- monic	Characteristics	Op- code	Page
	LD		Long-displacement facility.		
	ME		Monitor event.		
	MS		Message-security assist.		
I	M4		Message-security-assist extension 4.		
	N		Instruction is new in z/Architecture as compared to ESA/390.		
	N3		Instruction is new in z/Architecture and has been added to ESA/390. Any RSY or RXY instructions still use the RSE or RXE format and 12-bit displacements in ESA/390.		
	PE		Parsing-enhancement facility.		
I	PK		Population-count facility		
	R ₁		R ₁ field designates an access register in the access-register mode.		
	R ₂		R ₂ field designates an access register in the access-register mode.		
	R ₃		R ₃ field designates an access register in the access-register mode.		
	RI		RI instruction format.		
	RIE		RIE instruction format.		
	RIL		RIL instruction format.		
	RM		R ₁ field designates an access register in the access-register mode, and access-register 1 also is used in the access-register mode.		
	RR		RR instruction format.		
	RRE		RRE instruction format.		
	RRF		RRF instruction format.		
	RRS		RRS instruction format.		
	RS		RS instruction format.		
	RSI		RSI instruction format.		
	RSY		RSY instruction format.		
	RX		RX instruction format.		
	RXY		RXY instruction format.		
	S		S instruction format.		
	SC		Store-clock-fast facility.		
	SI		SI instruction format.		
	SIL		SIL instruction format.		
	SIY		SIY instruction format.		
	SP		Specification exception.		
	SS		SS instruction format.		
	SSF		SSF instruction format.		
	ST		PER storage-alteration event.		
	T		Trace exceptions (includes trace table, addressing, and low-address protection).		
	U ₁		R ₁ field designates an access register unconditionally.		
	U ₂		R ₂ field designates an access register unconditionally.		
	UB		R ₁ and R ₃ fields designate access registers unconditionally, and B ₂ field designates an access register in the access-register mode.		
	XX		Execute-extension facility.		

Figure 7-1. Summary of General Instructions (Part 12 of 12)

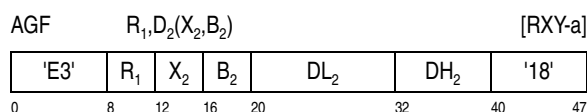
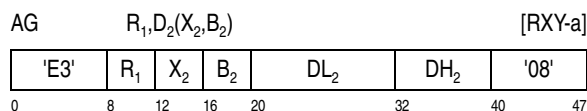
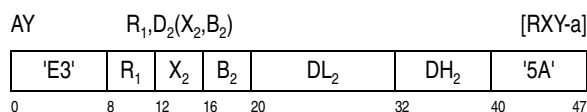
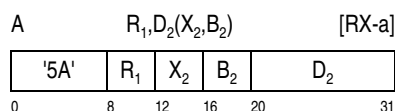
ADD

Register-and-register formats:



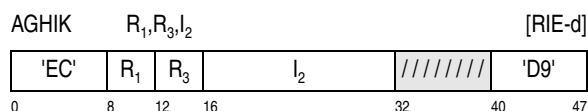
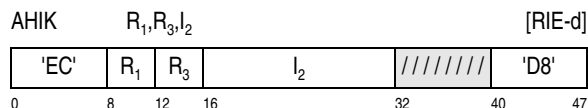
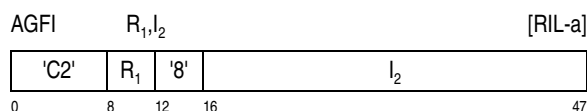
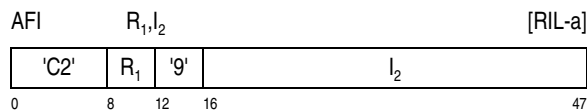


Register-and-storage formats:

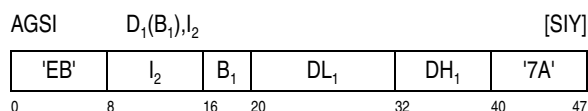
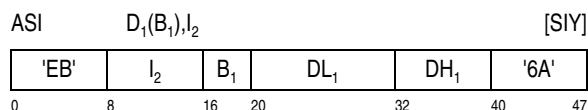


ADD IMMEDIATE

Register-and-immediate formats:



Storage-and-immediate formats:



For ADD (A, AG, AGF, AGFR, AGR, AR, and AY) and for ADD IMMEDIATE (AFI, AGFI, AGSI, and ASI), the second operand is added to the first operand, and the sum is placed at the first-operand location. For ADD (AGRK and ARK) and for ADD IMMEDIATE (AGHIK and AHIK), the second operand is added to the third operand, and the sum is placed at the first-operand location.

For ADD (A, AR, ARK, and AY) and for ADD IMMEDIATE (AFI), the operands and the sum are treated as 32-bit signed binary integers. For ADD (AG, AGR, and AGRK), they are treated as 64-bit signed binary integers. For ADD (AGFR, AGF) and for ADD IMMEDIATE (AGFI), the second operand is treated as a 32-bit signed binary integer, and the first operand and the sum are treated as 64-bit signed binary integers. For ADD IMMEDIATE (ASI), the second operand is treated as an 8-bit signed binary integer, and the first operand and the sum are treated as 32-bit signed binary integers. For ADD IMMEDIATE (AGSI), the second operand is treated as an 8-bit signed binary integer, and the first operand and the sum are treated as 64-bit signed binary integers. For ADD IMMEDIATE (AHIK), the first and third operands are treated as 32-bit signed binary integers, and the second operand is treated as a 16-bit signed binary integer. For ADD IMMEDIATE (AGHIK), the first and third operands are treated as 64-bit signed binary integers, and the second operand is treated as a 16-bit signed binary integer.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

When the interlocked-access facility is installed and the first operand of ADD IMMEDIATE (ASI, AGSI) is aligned on an integral boundary corresponding to its size, then the fetch and store of the first operand are performed as an interlocked update as observed by other CPUs, and a specific-operand-serialization operation is performed. When the interlocked-access facility is not installed, or when the first operand of ADD IMMEDIATE (ASI, AGSI) is not aligned on an integral boundary corresponding to its size, then the fetch and store of the operand are not performed as an interlocked update.

The displacement for A is treated as a 12-bit unsigned binary integer. The displacement for AY, AG, AGF, AGSI and ASI, is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

- Access (fetch and store, operand 1 of AGSI and ASI only; fetch, operand 2 of A, AY, AG, and AGF only)
- Fixed-point overflow
- Operation (AY, if the long-displacement facility is not installed; AFI and AGFI, if the extended-immediate facility is not installed; AGSI and ASI, if the general-instructions-extension facility is not installed; ARK, AGRK, AHIK, and AGHIK, if the distinct-operands facility is not installed)

Programming Notes:

1. Accesses to the first operand of ADD IMMEDIATE (AGSI and ASI) consist in fetching a first-operand from storage and subsequently storing the updated value.

2. When the interlocked-access facility is not installed, or when the first operand is not aligned on an integral boundary corresponding to its size, the fetch and store accesses to the first operand do not necessarily occur one immediately after the other. Under such conditions, ADD IMMEDIATE (AGSI and ASI) cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown in "Multiprogramming and Multiprocessing Examples" in Appendix A, "Number Representation and Instruction-Use Examples."

When the interlocked-access facility is installed and the first operand is aligned on an integral boundary corresponding to its size, the operand is accessed using a block-concurrent interlocked update.

3. For certain programming languages which ignore overflow conditions on arithmetic operations, the setting of condition code 3 obscures the sign of the result. However, for ADD IMMEDIATE, the sign of the I_2 field (which is known at the time of code generation) may be used in setting a branch mask which will accurately determine the resulting sign, as shown below.

I_2 Value	Result to be Tested		
	Positive	Negative	Zero
Positive	0010	0101 ^a	1000
Negative	0011 ^b	0100	1000 ^c
Zero	0010	0100	1000

Explanation:

^a If the first operand was positive, then a mask of 0001 covers that case (overflow). If the first operand was negative, then a mask of 0100 is appropriate. Combining the two gives a mask of 0101.

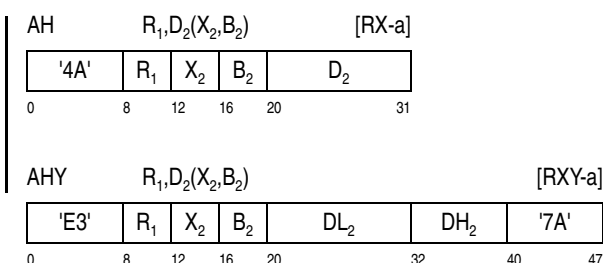
^b Similarly, a mask of 0010 (first operand was positive) and mask of 0001 (first operand was negative) yields a mask of 0011 which covers both cases.

^c A separate test for zero may be required when the I_2 field contains 80000000 hex.

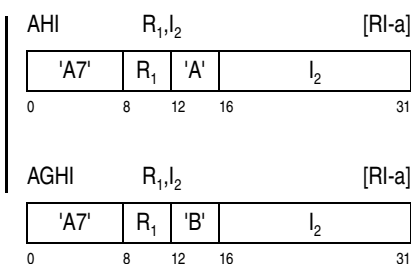
Figure 7-2. Branch Masks to Determine Resulting Sign for ADD IMMEDIATE

The technique described above is also applicable to ADD HALFWORD IMMEDIATE.

ADD HALFWORD



ADD HALFWORD IMMEDIATE



The second operand is added to the first operand, and the sum is placed at the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. For ADD HALFWORD (AH, AHY) and ADD HALFWORD IMMEDIATE (AHI), the first operand and the sum are treated as 32-bit signed binary integers. For ADD HALFWORD IMMEDIATE (AGHI), they are treated as 64-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

The displacement for AH is treated as a 12-bit unsigned binary integer. The displacement for AHY is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

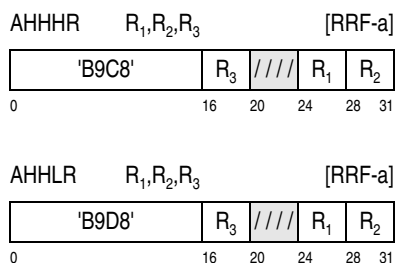
Program Exceptions:

- Access (fetch, operand 2 of AH, AHY)
- Fixed-point overflow
- Operation (AHY, if the long-displacement facility is not installed)

Programming Notes:

1. An example of the use of the ADD HALFWORD instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. See programming note 3 on page 7-26 for ADD IMMEDIATE regarding the implications of ignoring overflow.

ADD HIGH



The second operand is added to the third operand, and the sum is placed at the first-operand location. The operands and the sum are treated as 32-bit signed binary integers. The first and second operands are in bits 0-31 of general registers R₁ and R₂, respectively; bits 32-63 of general register R₁ are unchanged, and bits 32-63 of general register R₂ are ignored. For AHHHR, the third operand is in bits 0-31 of general register R₃; bits 32-63 of the register are ignored. For AHHLR, the third operand is in bits 32-63 of general register R₃; bits 0-31 of the register are ignored.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

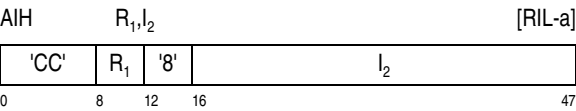
Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

- Fixed-point overflow
- Operation (if the high-word facility is not installed)

ADD IMMEDIATE HIGH



The second operand is added to the first operand, and the sum is placed at the first-operand location. The operands and the sum are treated as 32-bit signed binary integers. The first operand is in bits 0-31 of general register R_1 ; bits 32-63 of the register are unchanged.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

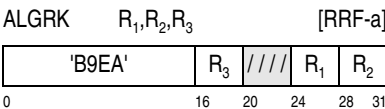
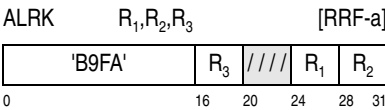
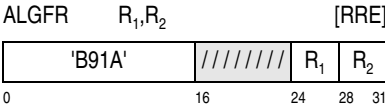
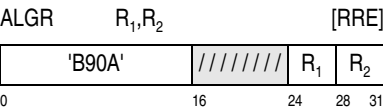
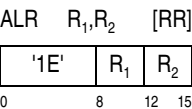
- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

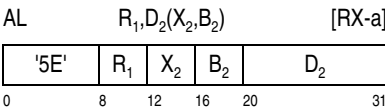
- Fixed-point overflow
- Operation (if the high-word facility is not installed)

ADD LOGICAL

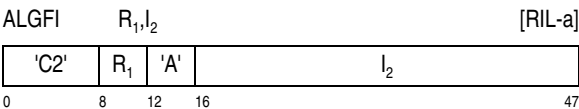
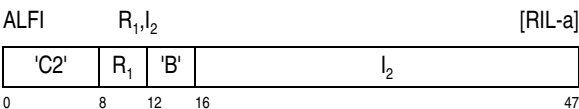
Register-and-register formats:



Register-and-storage formats:



ADD LOGICAL IMMEDIATE



For ADD LOGICAL (AL, ALG, ALGF, ALGFR, ALGR, ALR, and ALY) and for ADD LOGICAL IMMEDIATE (ALGFI and ALFI), the second operand is added to the first operand, and the sum is placed at the first-operand location. For ADD LOGICAL (ALGRK and ALRK), the second operand is added to the third

operand, and the sum is placed at the first-operand location.

For ADD LOGICAL (AL, ALR, ALRK, and ALY) and for ADD LOGICAL IMMEDIATE (ALFI), the operands and the sum are treated as 32-bit unsigned binary integers. For ADD LOGICAL (ALG, ALGR, and ALGRK), they are treated as 64-bit unsigned binary integers. For ADD LOGICAL (ALGFR, ALGF) and for ADD LOGICAL IMMEDIATE (ALGFI), the second operand is treated as a 32-bit unsigned binary integer, and the first operand and the sum are treated as 64-bit unsigned binary integers.

The displacement for AL is treated as a 12-bit unsigned binary integer. The displacement for ALY, ALG, and ALGF is treated as a 20-bit signed binary integer.

Resulting Condition Code:

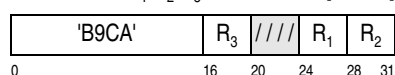
- 0 Result zero; no carry
- 1 Result not zero; no carry
- 2 Result zero; carry
- 3 Result not zero; carry

Program Exceptions:

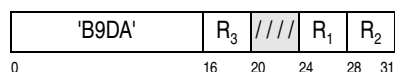
- Access (fetch, operand 2 of AL, ALY, ALG, and ALGF only)
- Operation (ALY, if the long-displacement facility is not installed; ALFI and ALGFI, if the extended-immediate facility is not installed; ALRK and ALGRK, if the distinct-operands facility is not installed)

ADD LOGICAL HIGH

ALHHHR R_1, R_2, R_3 [RRF-a]



ALHHLR R_1, R_2, R_3 [RRF-a]



The second operand is added to the third operand, and the sum is placed at the first-operand location. The operands and the sum are treated as 32-bit unsigned binary integers. The first and second oper-

ands are in bits 0-31 of general registers R_1 and R_2 , respectively; bits 32-63 of general register R_1 are unchanged, and bits 32-63 of general register R_2 are ignored. For ALHHHR, the third operand is in bits 0-31 of general register R_3 ; bits 32-63 of the register are ignored. For ALHHLR, the third operand is in bits 32-63 of general register R_3 ; bits 0-31 of the register are ignored.

Resulting Condition Code:

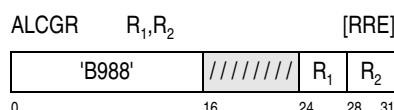
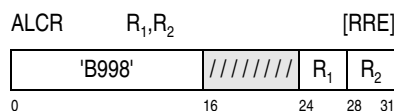
- 0 Result zero; no carry
- 1 Result not zero; no carry
- 2 Result zero; carry
- 3 Result not zero; carry

Program Exceptions:

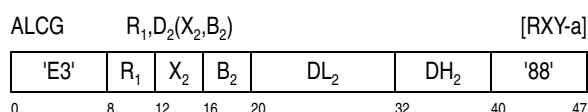
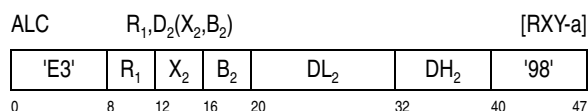
- Operation (if the high-word facility is not installed)

ADD LOGICAL WITH CARRY

Register-and-register formats:



Register-and-storage formats:



The second operand and the carry are added to the first operand, and the sum is placed at the first-operand location. For ADD LOGICAL WITH CARRY (ALCR, ALC), the operands, the carry, and the sum are treated as 32-bit unsigned binary integers. For ADD LOGICAL WITH CARRY (ALCGR, ALCG), they are treated as 64-bit unsigned binary integers.

Resulting Condition Code:

- 0 Result zero; no carry
- 1 Result not zero; no carry
- 2 Result zero; carry
- 3 Result not zero; carry

Program Exceptions:

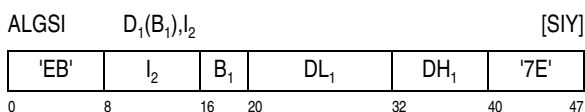
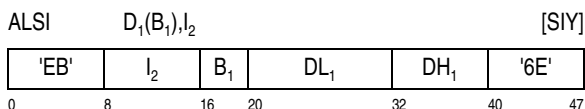
- Access (fetch, operand 2 of ALC and ALCG only)

Programming Notes:

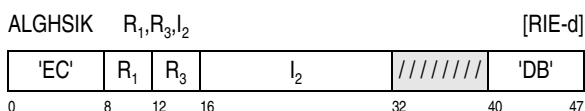
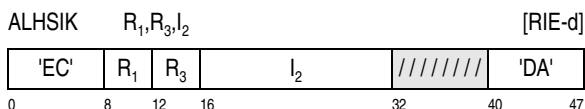
1. A carry is represented by a one value of bit 18 of the current PSW. Bit 18 is the leftmost bit of the two-bit condition code in the PSW. Bit 18 is set to one by an execution of an ADD LOGICAL or ADD LOGICAL WITH CARRY instruction that produces a carry out of bit position 0 of the result.
2. ADD and ADD LOGICAL may provide better performance than ADD LOGICAL WITH CARRY, depending on the model.

ADD LOGICAL WITH SIGNED IMMEDIATE

Storage-and-immediate formats:



Register-and-immediate formats:



For ALGSI and ALSI, the second operand is added to the first operand, and the sum is placed at the first-

operand location. For ALGHSIK and ALHSIK, the second operand is added to the third operand, and the sum is placed at the first-operand location.

For ALSI, the first operand and the sum are treated as 32-bit unsigned binary integers. For ALGSI, the first operand and the sum are treated as 64-bit unsigned binary integers. For both ALSI and ALGSI, the second operand is treated as an 8-bit signed binary integer.

For ALHSIK, the first and third operands are treated as 32-bit unsigned binary integers. For ALGHSIK, the first and third operands are treated as 64-bit unsigned binary integers. For both ALGHSIK and ALHSIK, the second operand is treated as a 16-bit signed binary integer.

When the interlocked-access facility is installed and the first operand is aligned on an integral boundary corresponding to its size, then the fetch and store of the first operand is performed as an interlocked update as observed by other CPUs, and a specific-operand-serialization operation is performed. When the interlocked-access facility is not installed, or when the first operand of ADD LOGICAL WITH SIGNED IMMEDIATE (ALSI, ALGSI) is not aligned on an integral boundary corresponding to its size, then the fetch and store of the operand are not performed as an interlocked update.

When the second operand contains a negative value, the condition code is set as though a SUBTRACT LOGICAL operation was performed. Condition code 0 is never set when the second operand is negative.

The displacement is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 Result zero; no carry
- 1 Result not zero; no carry
- 2 Result zero; carry
- 3 Result not zero; carry

Program Exceptions:

- Access (fetch and store, operand 1)
- Operation (ALGSI and ALSI, if the general-instructions-extension facility is not installed; ALGHSIK, and ALHSIK, if the distinct-operands facility is not installed)

Programming Notes:

1. When the second operand contains a negative value, the condition-code setting can also be interpreted as indicating the presence or absence of a borrow, as follows:
 - 1 Result not zero; borrow
 - 2 Result zero; no borrow
 - 3 Result not zero; no borrow
2. Accesses to the first operand of consist in fetching a first-operand from storage and subsequently storing the updated value.

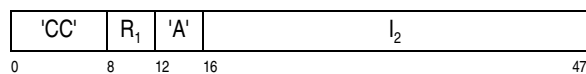
When the interlocked-access facility is not installed, or when the first operand is not aligned on an integral boundary corresponding to its size, the fetch and store accesses to the first operand do not necessarily occur one immediately after the other. Under such conditions, ADD LOGICAL WITH SIGNED IMMEDIATE cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown in "Multiprogramming and Multiprocessing Examples" in Appendix A, "Number Representation and Instruction-Use Examples."

When the interlocked-access facility is installed and the first operand is aligned on an integral boundary corresponding to its size, the operand is accessed using a block-concurrent interlocked update.

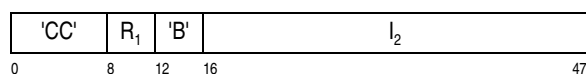
3. When the second operand is negative, the instruction effectively performs a subtraction operation.

ADD LOGICAL WITH SIGNED IMMEDIATE HIGH

ALSIH R_1, I_2 [RIL-a]



ALSIHN R_1, I_2 [RIL-a]



The second operand is added to the first operand, and the sum is placed at the first-operand location.

The first operand and the sum are treated as 32-bit unsigned binary integers. The second operand is treated as a 32-bit signed binary integer. The first operand is in bits 0-31 of general register R_1 ; bits 32-63 of the register are unchanged.

Resulting Condition Code:

For ALSIH, the code is set as follows:

- 0 Result zero; no carry
- 1 Result not zero; no carry
- 2 Result zero; carry
- 3 Result not zero; carry

For ALSIHN, the code remains unchanged.

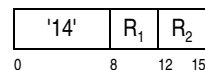
Program Exceptions:

- Operation (if the high-word facility is not installed)

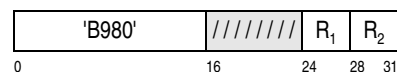
Programming Note: See programming notes 1 and 3 for ADD LOGICAL WITH SIGNED IMMEDIATE.

AND**Register-and-register formats:**

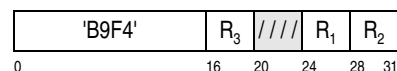
NR R_1, R_2 [RR]



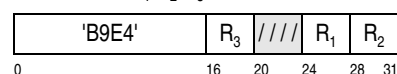
NGR R_1, R_2 [RRE]



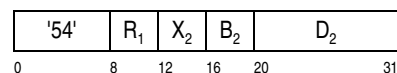
NRK R_1, R_2, R_3 [RRF-a]

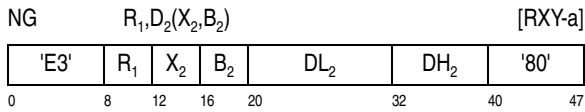
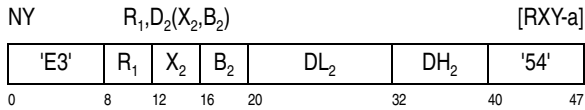


NGRK R_1, R_2, R_3 [RRF-a]

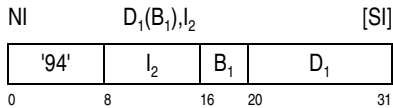
**Register-and-storage formats:**

N $R_1, D_2(X_2, B_2)$ [RX-a]

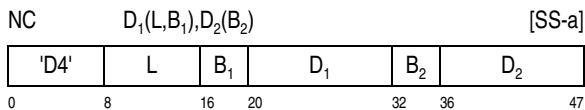




Storage-and-immediate formats:



Storage-and-storage format:



For N, NC, NG, NGR, NI, NIY, NR, and NY, the AND of the first and second operands is placed at the first-operand location. For NGRK and NRK, the AND of the second and third operands is placed at the first-operand location.

The connective AND is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the corresponding bit positions in both operands contain ones; otherwise, the result bit is set to zero.

For AND (NC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

For AND (NI, NIY), the first operand is one byte in length, and only one byte is stored.

For AND (N, NR, NRK, and NY), the operands are 32 bits, and for AND (NG, NGR, and NGRK), they are 64 bits.

The displacements for N, NI, and both operands of NC are treated as 12-bit unsigned binary integers. The displacement for NY, NIY, and NG is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 Result zero
- 1 Result not zero
- 2 --
- 3 --

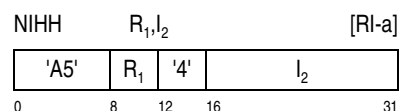
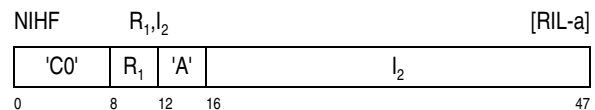
Program Exceptions:

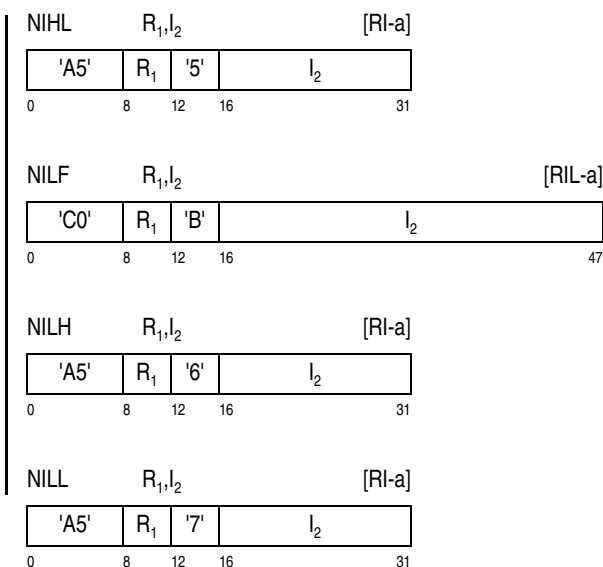
- Access (fetch, operand 2, N, NY, NG, and NC; fetch and store, operand 1, NI, NIY, and NC)
- Operation (NY and NIY, if the long-displacement facility is not installed; NGRK and NRK, if the distinct-operands facility is not installed)

Programming Notes:

1. An example of the use of the AND instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The AND instruction may be used to set a bit to zero.
3. Accesses to the first operand of AND (NI) and AND (NC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, the instruction AND cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in "Multiprogramming and Multiprocessing Examples" on page A-45.

AND IMMEDIATE





The second operand is ANDed with bits of the first operand, and the result replaces those bits of the first operand. The remainder of the first operand remains unchanged.

For each instruction, the bits of the first operand that are ANDed with the second operand and then replaced are as follows:

Instruction	Bits ANDed and Replaced
NIHF	0-31
NIHH	0-15
NIHL	16-31
NILF	32-63
NILH	32-47
NILL	48-63

The connective AND is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the corresponding bit positions in both operands contain ones; otherwise, the result bit is set to zero.

Resulting Condition Code:

0	Result is zero
1	Result is not zero
2	--
3	--

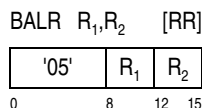
Program Exceptions:

- Operation (NIHF, NILF, if the extended-immediate facility is not installed)

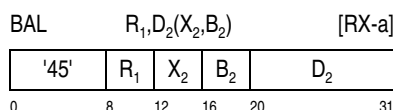
Programming Note: The setting of the condition code is based only on the bits that are ANDed and replaced.

BRANCH AND LINK

Register-and-register format:

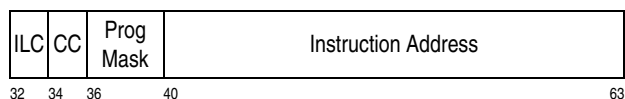


Register-and-storage format:



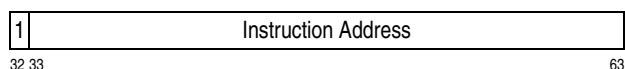
Information from the current PSW, including the updated instruction address, is saved as link information at the first-operand location. Subsequently, the instruction address in the PSW is replaced by the branch address.

The link information in the 24-bit addressing mode consists of the instruction-length code (ILC), the condition code (CC), the program-mask bits, and the rightmost 24 bits of the updated instruction address, arranged in bit positions 32-63 of the first-operand location in the following format:



The instruction-length code is 1 or 2.

The link information in the 31-bit addressing mode consists of bit 32 of the PSW, the basic-addressing-mode bit (always a one) and the rightmost 31 bits of the updated instruction address, arranged in bit positions 32-63 of the first-operand location in the following format:



In the 24-bit or 31-bit addressing mode, bits 0-31 of the first-operand location remain unchanged.

The link information in the 64-bit addressing mode consists of the updated instruction address, placed in bit positions 0-63 of the first-operand location.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of general register R_2 are used to generate the branch address; however, when the R_2 field is zero, the operation is performed without branching. The branch address is computed before general register R_1 is changed.

Condition Code: The code remains unchanged.

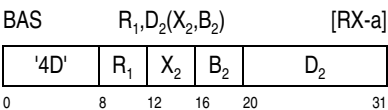
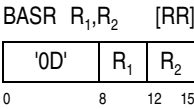
Program Exceptions:

- Trace (R_2 field nonzero, BALR only)

Programming Notes:

1. An example of the use of the BRANCH AND LINK instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. When the R_2 field in the RR format is zero, the link information is loaded without branching.
3. The BRANCH AND LINK instruction (BAL and BALR) is provided for compatibility purposes. It is recommended that, where possible, the BRANCH AND SAVE instruction (BAS and BASR), BRANCH RELATIVE AND SAVE, or BRANCH RELATIVE AND SAVE LONG be used and BRANCH AND LINK avoided, since the latter places nonzero information in bit positions 32-39 of the link register in the 24-bit addressing mode, which may lead to problems. Additionally, in the 24-bit addressing mode, BRANCH AND LINK may be slower than the other instructions because BRANCH AND LINK must construct the ILC, condition code, and program mask to be placed in bit positions 32-39 of the link register.
4. The condition-code and program-mask information, which is provided in the leftmost byte of the link information only in the 24-bit addressing mode, can be obtained in any addressing mode by means of the INSERT PROGRAM MASK instruction.

BRANCH AND SAVE



Information from the current PSW, including the updated instruction address, is saved as link information at the first-operand location. Subsequently, the instruction address in the PSW is replaced by the branch address.

In the 24-bit or 31-bit addressing mode, the link information is bits 32 and 97-127 of the PSW, consisting of the basic-addressing-mode bit and the rightmost 31 bits of the updated instruction address. The link information is placed in bit positions 32 and 33-63, respectively, of the first-operand location, and bits 0-31 of the location remain unchanged.

In the 64-bit addressing mode, the link information consists of the updated instruction address, placed in bit positions 0-63 of the first-operand location.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of general register R_2 are used to generate the branch address; however, when the R_2 field is zero, the operation is performed without branching. The branch address is computed before general register R_1 is changed.

Condition Code: The code remains unchanged.

Program Exceptions:

- Trace (R_2 field nonzero, BASR only)

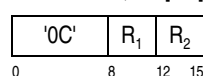
Programming Notes:

1. An example of the use of the BRANCH AND SAVE instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. The BRANCH AND SAVE instruction (BAS and BASR) is intended to be used for linkage to programs known to be in the same addressing mode

as the caller. This instruction should be used in place of the BRANCH AND LINK instruction (BAL and BALR). See the programming notes on pages 5-12 and 5-15 in the section “Subroutine Linkage without the Linkage Stack” for a detailed discussion of this and other linkage instructions. See also the programming note under BRANCH AND LINK for a discussion of the advantages of the BRANCH AND SAVE instruction.

BRANCH AND SAVE AND SET MODE

BASSM R1,R2 [RR]



Information from the current PSW, including the updated instruction address, is saved as link information at the first-operand location. Subsequently, if the R₂ field is nonzero, the addressing-mode bits and instruction address in the PSW are replaced as specified by the second operand.

In the 24-bit or 31-bit addressing mode, the link information is bits 32 and 97-127 of the PSW, consisting of the basic-addressing-mode bit and the rightmost 31 bits of the updated instruction address. The link information is placed in bit positions 32 and 33-63, respectively, of the first-operand location, and bits 0-31 of the location remain unchanged. In the 64-bit addressing mode, the link information is bits 64-126 of the PSW with a one appended on the right, placed in bit positions 0-63 of the first-operand location.

The contents of general register R₂ specify the new addressing mode and designate the branch address; however, when the R₂ field is zero, the operation is performed without branching and without setting either addressing-mode bit.

When the contents of general register R₂ are used and bit 63 of the register is zero, bit 31 of the current PSW, the extended-addressing-mode bit, is set to zero, bit 32 of the register specifies the new basic addressing mode and replaces bit 32 of the PSW, and the branch address is generated from the contents of the register under the control of the new addressing mode. The branch address replaces the instruction address in the PSW.

When the contents of general register R₂ are used and bit 63 of the register is one, the following occurs. Bits 31 and 32 of the current PSW are set to one, the branch address is generated from the contents of the register, except with bit 63 of the register treated as a zero, under the control of the new extended addressing mode, and the branch address replaces the instruction address in the PSW. Bit 63 of the register remains one. However, if R₂ is the same as R₁, the results in the designated general register are as specified for the R₁ register.

The new value for the PSW is computed before general register R₁ is changed.

Condition Code: The code remains unchanged.

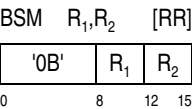
Program Exceptions:

- Trace (R₂ field nonzero)

Programming Notes:

1. An example of the use of the BRANCH AND SAVE AND SET MODE instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. BRANCH AND SAVE AND SET MODE is intended to be the principal calling instruction to subroutines which may operate in a different addressing mode from that of the caller. See the programming notes on pages 5-12 and 5-15 in the section “Subroutine Linkage without the Linkage Stack” for a detailed discussion of this and other linkage instructions.
3. An old 24-bit or 31-bit program can use BRANCH AND SAVE AND SET MODE to call a new 64-bit program without any change, provided that bits 0-31 of general register R₂ are all zeros. The old program can load into bit positions 32-63 of general register R₂ a four-byte address constant, which is provided from outside the program, in which bit 63 in the register (bit 31 of the constant in storage) either is or is not one. If the addressing mode is not changed to the 64-bit mode by the execution of the BRANCH AND SAVE AND SET MODE instruction, or even if it is, the called program can set the 64-bit mode by issuing a SET ADDRESSING MODE (SAM64) instruction.
4. See the programming notes on page 5-12 (under “Simple Branch Instructions”).

BRANCH AND SET MODE



In the 24-bit or 31-bit addressing mode, bit 32 of the current PSW, the basic-addressing-mode bit, is inserted into bit position 32 of the first operand, and bits 0-31 and 33-63 of the operand remain unchanged. In the 64-bit addressing mode, a one is inserted into bit position 63 of the first operand, and bits 0-62 of the operand remain unchanged. Subsequently, the addressing-mode bits and instruction address in the PSW are replaced as specified by the second operand. The action associated with an operand is not performed if the associated R field is zero.

The contents of general register R₂ specify the new addressing mode and designate the branch address; however, when the R₂ field is zero, the operation is performed without branching and without setting either addressing-mode bit.

When the contents of general register R₂ are used and bit 63 of the register is zero, bit 31 of the current PSW, the extended-addressing-mode bit, is set to zero, bit 32 of the register specifies the new basic addressing mode and replaces bit 32 of the PSW, and the branch address is generated from the contents of the register under the control of the new addressing mode. The branch address replaces the instruction address in the PSW.

When the contents of general register R₂ are used and bit 63 of the register is one, the following occurs. Bits 31 and 32 of the current PSW are set to one, the branch address is generated from the contents of the register, except with bit 63 of the register treated as a zero, under the control of the new extended addressing mode, and the branch address replaces the instruction address in the PSW. Bit 63 of the register remains one. However, if R₂ is the same as R₁, the results in the designated general register are as specified for the R₁ register.

The new value for the PSW is computed before general register R₁ is changed.

Condition Code: The code remains unchanged.

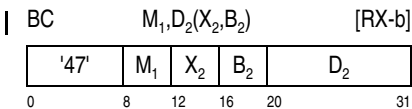
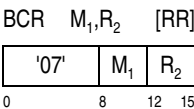
Program Exceptions:

- Trace

Programming Notes:

1. An example of the use of the BRANCH AND SET MODE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. BRANCH AND SET MODE with an R₁ field of zero is intended to be the standard return instruction in a program entered by means of BRANCH AND SAVE AND SET MODE. It can also be the return instruction in a program entered in the 24-bit or 31-bit addressing mode by means of BRANCH AND SAVE, BRANCH RELATIVE AND SAVE, or BRANCH RELATIVE AND SAVE LONG. BRANCH AND SET MODE with a nonzero R₁ field is intended to be used in a "glue module" to connect either old 24-bit programs and newer programs that are executed in the 31-bit addressing mode or old 24-bit or 31-bit programs and new programs that are executed in the 64-bit addressing mode. See the programming notes on pages 5-12 and 5-15 in the section "Subroutine Linkage without the Linkage Stack" for a detailed discussion of this and other linkage instructions.

BRANCH ON CONDITION



The instruction address in the current PSW is replaced by the branch address if the condition code has one of the values specified by M₁; otherwise, normal instruction sequencing proceeds with the updated instruction address.

In the RX format, the second-operand address is used as the branch address. In the RR format, the contents of general register R₂ are used to generate the branch address; however, when the R₂ field is zero, the operation is performed without branching.

The M_1 field is used as a four-bit mask. The four condition codes (0, 1, 2, and 3) correspond, left to right, with the four bits of the mask, as follows:

Condition Code	Instruction Bit Number of Mask	Mask Position Value
0	8	8
1	9	4
2	10	2
3	11	1

The current condition code is used to select the corresponding mask bit. If the mask bit selected by the condition code is one, the branch is successful. If the mask bit selected is zero, normal instruction sequencing proceeds with the next sequential instruction.

When the M_1 and R_2 fields of BRANCH ON CONDITION (BCR) are 1111 binary and 0000 binary, respectively, a serialization and checkpoint-synchronization function is performed. When the fast-BCR-serialization facility is installed and the M_1 and R_2 fields of BRANCH ON CONDITION (BCR) are 1110 binary and 0000 binary, respectively, a serialization function is performed.

Condition Code: The code remains unchanged.

Program Exceptions: None.

Programming Notes:

1. An example of the use of the BRANCH ON CONDITION instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When a branch is to depend on more than one condition, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made when the condition code is 0 or 1.
3. When all four mask bits are zeros or when the R_2 field in the RR format contains zero, the branch instruction is equivalent to a no-operation. When all four mask bits are ones, that is, the mask value is 15, the branch is unconditional unless the R_2 field in the RR format is zero.

4. Execution of BCR 15,0 (that is, an instruction with a value of 07F0 hex) may result in significant performance degradation. To ensure optimum performance, the program should avoid use of BCR 15,0 except in cases when the serialization or checkpoint-synchronization function is actually required.

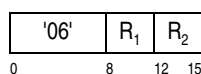
When the fast-BCR-serialization facility is installed, and the program only needs the serialization function (without the checkpoint synchronization function), then BCR 14,0 should be used. Depending on the model, this may be faster than BCR 15,0.

5. Note that the relation between the RR and RX formats in branch-address specification is not the same as in operand-address specification. For branch instructions in the RX format, the branch address is the address specified by X_2 , B_2 , and D_2 ; in the RR format, the branch address is contained in the register designated by R_2 . For operands, the address specified by X_2 , B_2 , and D_2 is the operand address, but the register designated by R_2 contains the operand, not the operand address.

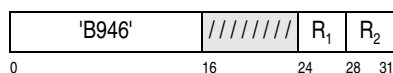
BRANCH ON COUNT

Register-and-register formats:

BCTR R_1, R_2 [RR]

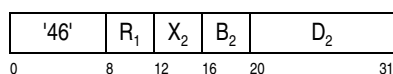


BCTGR R_1, R_2 [RRE]

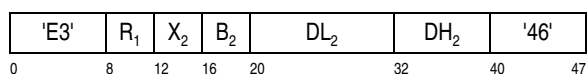


Register-and-storage formats:

BCT $R_1, D_2(X_2, B_2)$ [RX-a]



BCTG $R_1, D_2(X_2, B_2)$ [RXY-a]



A one is subtracted from the first operand, and the result is placed at the first-operand location. For

BRANCH ON COUNT (BCT, BCTR), the first operand and result are treated as 32-bit binary integers, with overflow ignored. For BRANCH ON COUNT (BCTG, BCTGR), the first operand and result are treated as 64-bit binary integers, with overflow ignored. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address in the current PSW is replaced by the branch address.

In the RX or RXY format, the second-operand address is used as the branch address. In the RR or RRE format, the contents of general register R_2 are used to generate the branch address; however, when the R_2 field is zero, the operation is performed without branching. The branch address is generated before general register R_1 is changed.

Condition Code: The code remains unchanged.

Program Exceptions: None.

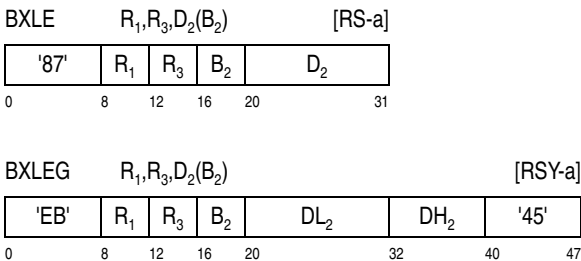
Programming Notes:

- 1. An example of the use of the BRANCH ON COUNT instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
- 2. The first operand and result can be considered as either signed or unsigned binary integers since the result of a binary subtraction is the same in both cases.
- 3. An initial count of one results in zero, and no branching takes place; an initial count of zero results in -1 and causes branching to be performed; an initial count of -1 results in -2 and causes branching to be performed; and so on. In a loop, branching takes place each time the instruction is executed until the result is again zero. Note that for BCT or BCTR, because of the number range, an initial count of -2^{31} results in a positive value of $2^{31} - 1$, or, for BCTG or BCTGR, an initial count of -2^{63} results in a positive value of $2^{63} - 1$.
- 4. Counting is performed without branching when the R_2 field in the RR or RRE format contains zero.

BRANCH ON INDEX HIGH



BRANCH ON INDEX LOW OR EQUAL



An increment is added to the first operand, and the sum is compared with a compare value. The result of the comparison determines whether branching occurs. Subsequently, the sum is placed at the first-operand location. The second-operand address is used as a branch address. The R_3 field designates registers containing the increment and the compare value.

For BRANCH ON INDEX HIGH, when the sum is high, the instruction address in the current PSW is replaced by the branch address. When the sum is low or equal, normal instruction sequencing proceeds with the updated instruction address.

For BRANCH ON INDEX LOW OR EQUAL, when the sum is low or equal, the instruction address in the current PSW is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address.

When the R_3 field is even, it designates a pair of registers; the contents of the even and odd registers of the pair are used as the increment and the compare value, respectively. When the R_3 field is odd, it designates a single register, the contents of which are used as both the increment and the compare value.

For purposes of the addition and comparison, all operands and results are treated as 32-bit signed binary integers for BXH and BXLE or as 64-bit signed binary integers for BXHG and BXLEG. Overflow caused by the addition is ignored.

The original contents of the compare-value register are used as the compare value even when that register is also specified to be the first-operand location. The branch address is generated before general register R₁ is changed.

The sum is placed at the first-operand location, regardless of whether the branch is taken.

Condition Code: The code remains unchanged.

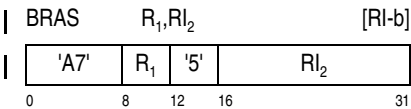
Program Exceptions: None.

Programming Notes:

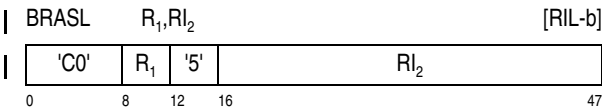
- Several examples of the use of the BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL instructions are given in Appendix A, “Number Representation and Instruction-Use Examples.”
- The word “index” in the names of these instructions indicates that one of the major purposes is the incrementing and testing of an index value. The increment, being a signed binary integer, may be used to increase or decrease the value in general register R₁ by an arbitrary amount, subject to the limit of the integer size.
- Care must be taken in the 31-bit addressing mode when a data area in storage is at the rightmost end of a 31-bit address space and a BRANCH ON INDEX HIGH (BXH) or BRANCH ON INDEX LOW OR EQUAL (BXLE) instruction is used to step upward through the data. Since the addition and comparison operations performed during the execution of these instructions treat the operands as 32-bit signed binary integers, the value following 2³¹ - 1 is not 2³¹, which cannot be represented in that format, but -2³¹. The instruction does not provide an indication of such overflow. Consequently, some common looping techniques based on the use of these instructions do not work when a data area ends at address 2³¹ - 1. This problem is illustrated in a BRANCH ON INDEX LOW OR EQUAL example in Appendix A, “Number Representation and Instruction-Use Examples.” A similar caution

applies in the 64-bit addressing mode when data is at the end of a 64-bit address space and BRANCH ON INDEX HIGH (BXHG) or BRANCH ON INDEX LOW OR EQUAL (BXLEG) is used.

BRANCH RELATIVE AND SAVE



BRANCH RELATIVE AND SAVE LONG



Information from the current PSW, including the updated instruction address, is saved as link information at the first-operand location. Subsequently, the instruction address in the PSW is replaced by the branch address.

In the 24-bit or 31-bit addressing mode, the link information is bits 32 and 97-127 of the PSW, consisting of the basic-addressing-mode bit and the rightmost 31 bits of the updated instruction address. The link information is placed in bit positions 32 and 33-63, respectively, of the first-operand location, and bits 0-31 of the location remain unchanged.

In the 64-bit addressing mode, the link information consists of the updated instruction address, placed in bit positions 0-63 of the first-operand location.

The contents of the I₂ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

Condition Code: The code remains unchanged.

Program Exceptions: None.

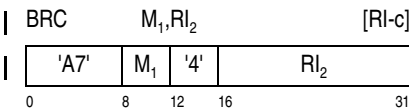
Programming Notes:

- The operation is the same as that of the BRANCH AND SAVE (BAS) instruction except for the means of specifying the branch address.

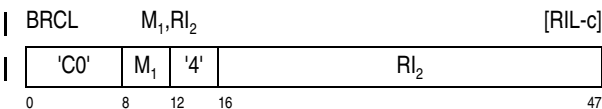
An example of the use of BRANCH AND SAVE is given in Appendix A, “Number Representation and Instruction-Use Examples.”

- 2. The BRANCH RELATIVE AND SAVE and BRANCH RELATIVE AND SAVE LONG instructions, like the BRANCH AND SAVE instruction, are intended to be used for linkage to programs known to be in the same addressing mode as the caller. These instructions should be used in place of the BRANCH AND LINK instruction (BAL and BALR). See the programming notes on pages 5-12 and 5-15 in the section “Subroutine Linkage without the Linkage Stack” for a detailed discussion of these and other linkage instructions. See also the programming note under BRANCH AND LINK for a discussion of the advantages of the BRANCH RELATIVE AND SAVE, BRANCH RELATIVE AND SAVE LONG, and BRANCH AND SAVE instructions.
- 3. When the instruction is the target of an execute-type instruction, the branch is relative to the target address; see “Branch-Address Generation” on page 5-10.

BRANCH RELATIVE ON CONDITION



BRANCH RELATIVE ON CONDITION LONG



The instruction address in the current PSW is replaced by the branch address if the condition code has one of the values specified by M₁; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The contents of the I₂ field are a signed binary integer specifying the number of halfwords that is added

to the address of the instruction to generate the branch address.

The M₁ field is used as a four-bit mask. The four condition codes (0, 1, 2, and 3) correspond, left to right, with the four bits of the mask, as follows:

Condition Code	Instruction Bit Number of Mask	Mask Position Value
0	8	8
1	9	4
2	10	2
3	11	1

The current condition code is used to select the corresponding mask bit. If the mask bit selected by the condition code is one, the branch is successful. If the mask bit selected is zero, normal instruction sequencing proceeds with the next sequential instruction.

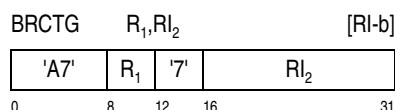
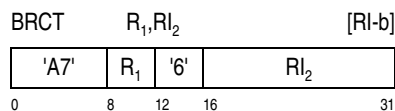
Condition Code: The code remains unchanged.

Program Exceptions: None.

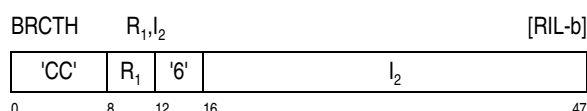
Programming Notes:

- 1. The operation is the same as that of the BRANCH ON CONDITION instruction except for the means of specifying the branch address. An example of the use of BRANCH ON CONDITION is given in Appendix A, “Number Representation and Instruction-Use Examples.”
- 2. When a branch is to depend on more than one condition, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made when the condition code is 0 or 1.
- 3. When all four mask bits are zeros, the branch instruction is equivalent to a no-operation. When all four mask bits are ones, that is, the mask value is 15, the branch is unconditional.
- 4. When the instruction is the target of an execute-type instruction, the branch is relative to the target address; see “Branch-Address Generation” on page 5-10.

BRANCH RELATIVE ON COUNT



BRANCH RELATIVE ON COUNT HIGH



A one is subtracted from the first operand, and the result is placed at the first-operand location. For BRANCH RELATIVE ON COUNT (BRCT), the first operand and result are treated as 32-bit binary integers in bits 32-63 of general register R_1 , with overflow ignored; bits 0-31 of the register are unchanged. For BRANCH RELATIVE ON COUNT HIGH (BRCTH), the first operand and result are treated as 32-bit binary integers in bits 0-31 of general register R_1 , with overflow ignored; bits 32-63 of the register are unchanged. For BRANCH RELATIVE ON COUNT (BRCTG), the first operand and result are treated as 64-bit binary integers, with overflow ignored. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address in the current PSW is replaced by the branch address.

The contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

Condition Code: The code remains unchanged.

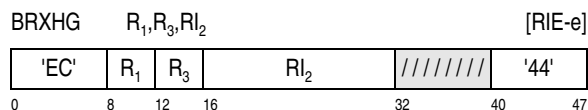
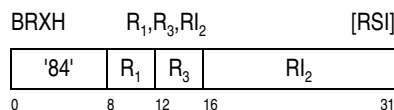
Program Exceptions:

- Operation (BRCTH, if the high-word facility is not installed)

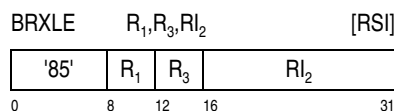
Programming Notes:

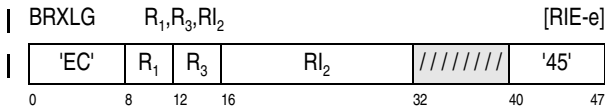
- The operation is the same as that of the BRANCH ON COUNT instruction except for the means of specifying the branch address. An example of the use of BRANCH ON COUNT is given in Appendix A, "Number Representation and Instruction-Use Examples."
- The first operand and result can be considered as either signed or unsigned binary integers since the result of a binary subtraction is the same in both cases.
- An initial count of one results in zero, and no branching takes place; an initial count of zero results in -1 and causes branching to be executed; an initial count of -1 results in -2 and causes branching to be executed; and so on. In a loop, branching takes place each time the instruction is executed until the result is again zero. Note that for BRCT, because of the number range, an initial count of -2^{31} results in a positive value of $2^{31} - 1$, or, for BRCTG, an initial count of -2^{63} results in a positive value of $2^{63} - 1$.
- When the instruction is the target of an execute-type instruction, the branch is relative to the target address; see "Branch-Address Generation" on page 5-10.

BRANCH RELATIVE ON INDEX HIGH



BRANCH RELATIVE ON INDEX LOW OR EQUAL





An increment is added to the first operand, and the sum is compared with a compare value. The result of the comparison determines whether branching occurs. Subsequently, the sum is placed at the first-operand location. The R₃ field designates registers containing the increment and the compare value.

The contents of the I₂ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

For BRANCH RELATIVE ON INDEX HIGH, when the sum is high, the instruction address in the current PSW is replaced by the branch address. When the sum is low or equal, normal instruction sequencing proceeds with the updated instruction address.

For BRANCH RELATIVE ON INDEX LOW OR EQUAL, when the sum is low or equal, the instruction address in the current PSW is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address.

When the R₃ field is even, it designates a pair of registers; the contents of the even and odd registers of the pair are used as the increment and the compare value, respectively. When the R₃ field is odd, it designates a single register, the contents of which are used as both the increment and the compare value.

For purposes of the addition and comparison, all operands and results are treated as 32-bit signed binary integers for BRXH and BRXLE or as 64-bit signed binary integers for BRXHG and BRXLG. Overflow caused by the addition is ignored.

The original contents of the compare-value register are used as the compare value even when that register is also specified to be the first-operand location.

The sum is placed at the first-operand location, regardless of whether the branch is taken.

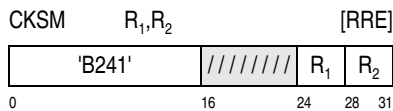
Condition Code: The code remains unchanged.

Program Exceptions: None.

Programming Notes:

1. The operations are the same as those of the BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL instructions except for the means of specifying the branch address. Several examples of the use of BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The word "index" in the names of these instructions indicates that one of the major purposes is the incrementing and testing of an index value. The increment, being a signed binary integer, may be used to increase or decrease the value in general register R₁ by an arbitrary amount.
3. Care must be taken in the 31-bit addressing mode when a data area in storage is at the right-most end of an address space and a BRANCH RELATIVE ON INDEX HIGH (BRXH) or BRANCH RELATIVE ON INDEX LOW OR EQUAL (BRXLE) instruction is used to step upward through the data. Since the addition and comparison operations performed during the execution of these instructions treat the operands as 32-bit signed binary integers, the value following $2^{31} - 1$ is not 2^{31} , which cannot be represented in that format, but -2^{31} . The instruction does not provide an indication of such overflow. Consequently, some common looping techniques based on the use of these instructions do not work when a data area ends at address $2^{31} - 1$. This problem is illustrated in a BRANCH ON INDEX LOW OR EQUAL example in Appendix A, "Number Representation and Instruction-Use Examples." A similar caution applies in the 64-bit addressing mode when data is at the end of a 64-bit address space and BRANCH RELATIVE ON INDEX HIGH (BRXHG) or BRANCH RELATIVE ON INDEX LOW OR EQUAL (BRXLG) is used.
4. When the instruction is the target of an execute-type instruction, the branch is relative to the target address; see "Branch-Address Generation" on page 5-10.

CHECKSUM



Successive four-byte elements of the second operand are added to the first operand in bit positions 32-63 of general register R_1 to form a 32-bit checksum in those bit positions. The first operand and the four-byte elements are treated as 32-bit unsigned binary integers. After each addition of an element, a carry out of bit position 32 of the first operand is added to bit position 63 of the first operand. Bits 0-31 of general register R_1 always remain unchanged. If the second operand is not a multiple of four bytes, its last one, two, or three bytes are treated as appended on the right with the number of all-zeros bytes needed to form a four-byte element. The four-byte elements are added to the first operand until either the entire second operand or a CPU-determined amount of the second operand has been processed. The result is indicated in the condition code.

The R_2 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the second operand is specified by the contents of the R_2 general register. The number of bytes in the second-operand location is specified by the 32-bit or 64-bit unsigned binary integer in the $R_2 + 1$ general register.

The handling of the address in general register R_2 and the length in general register $R_2 + 1$ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the register constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of the register constitute the address. In the 24-bit or 31-bit addressing mode, the length is a 32-bit unsigned binary integer in bit positions 32-63 of general register $R_2 + 1$, and the contents of bit positions 0-31 are ignored. In the 64-bit addressing mode, the length is a 64-bit unsigned binary integer in the register.

The addition of second-operand four-byte elements to the first operand proceeds left to right, four-byte element by four-byte element, and ends as soon as (1) the entire second operand has been processed or (2) a lesser CPU-determined amount of the second operand has been processed. In either case, the result in bit positions 32-63 of general register R_1 is a 32-bit checksum for the part of the second operand that has been processed. When the second operand is not a multiple of four bytes, the final second-operand bytes in excess of a multiple of four are conceptually appended on the right with an appropriate number of all-zeros bytes to form the final four-byte element.

If the operation ends because the entire second operand has been processed, the condition code is set to 0. If the operation ends because a lesser CPU-determined amount of the second operand has been processed, the condition code is set to 3. When the operation is to end with a setting of condition code 3, any carry out of bit position 32 of the first operand is added to bit position 63 of the first operand before the operation ends.

At the completion of the operation, the 32-bit or 64-bit operand-length field in the $R_2 + 1$ register is decremented by the number of actual second-operand bytes added to the first operand (not including any conceptually appended all-zeros bytes), and the address in the R_2 register is incremented by the same number. Thus, the 32-bit or 64-bit operand-length field contains a zero value if the condition code is set to 0, or it contains a nonzero value if the condition code is set to 3. In the 24-bit or 31-bit addressing mode, bits 0-31 of the $R_2 + 1$ register always remain unchanged.

When condition code 3 is set, the general registers used by the instruction have been set so that the remainder of the second operand can be processed by simply branching back to reexecute the instruction.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed. The minimum amount is four bytes or the number of bytes specified in the $R_2 + 1$ general register, whichever is smaller.

At the completion of the operation in the 24-bit or 31-bit addressing mode, the leftmost bits which are

not part of the address in bit positions 32-63 of general register R_2 may be set to zeros or may remain unchanged, even when the initial length in register $R_2 + 1$ is zero. Bits 0-31 of general register R_2 remain unchanged.

When the R_1 register is the same register as the R_2 or $R_2 + 1$ register, the results are unpredictable.

Access exceptions for the portion of the second operand to the right of the last byte processed may or may not be recognized. For a second operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

Access exceptions are not recognized if the R_2 field is odd. When the length of the second operand is zero, no access exceptions are recognized.

Resulting Condition Code:

- 0 Entire second operand processed
- 1 --
- 2 --
- 3 CPU-determined amount of second operand processed

Program Exceptions:

- Access (fetch, operand 2)
- Specification

Programming Notes:

1. The initial contents of bit positions 32-63 of the R_1 general register contribute to the 32-bit checksum. The program normally should set those contents to all zeros before issuing the CHECKSUM instruction.
2. A 16-bit checksum is used in, for example, the TCP/IP application. The following program can be executed after the CHECKSUM instruction to produce in bit positions 32-63 of general register R_2 a 16-bit checksum from the 32-bit checksum in bit positions 32-63 of general register R_1 . The program is annotated to show the contents of bit positions 32-63 of the R_2 and $R_2 + 1$ registers after the execution of each instruction. The con-

tents of bit positions 32-63 of the R_1 register are represented as A,B, meaning the value A in bit positions 32-47 and the value B in bit positions 48-63. The value C is a carry from $A + B$. Note that bit positions 32-63 of register $R_2 + 1$ are known to contain all zeros when CHECKSUM has set condition code 0.

Program		R_2 Bits 32-63	$R_2 + 1$ Bits 32-63
LR	R_2, R_1	A,B	0,0
SRDL	$R_2, 16$	0,A	B,0
ALR	$R_2, R_2 + 1$	B,A	B,0
ALR	R_2, R_1	$A + B + C, A + B$	B,0
SRL	$R_2, 16$	0, $A + B + C$	B,0

3. The CHECKSUM instruction may be used in computing hash values as illustrated in the following programming example. The variable KEY contains a string to be mapped into a slot in a hash table. The variable SIZE is a prime number designating the size of the hash table. The value of SIZE is determined by (a) the number of strings to be hashed into the table divided by the acceptable number of hash collisions, and (b) a value that is not too close to a power of two. Following the DIVIDE (D) instruction, the remainder in register 0 represents the resulting hash value.

LOOP	SR	1,1	Zero accumulator
	LA	2,KEY	Point to string
	LA	3,L'KEY	Load string length
	CKSM	1,2	Compute checksum
	BNZ	LOOP	Repeat if not done
	SR	0,0	Zero for divide
	D	0,SIZE	Compute hash value
	...		
KEY	DS	CL64	String to be hashed
SIZE	DS	F	Size of hash table

4. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
5. The storage-operand references of CHECKSUM may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)
6. Figure 7-3 on page 7-45 contains a summary of the operation.

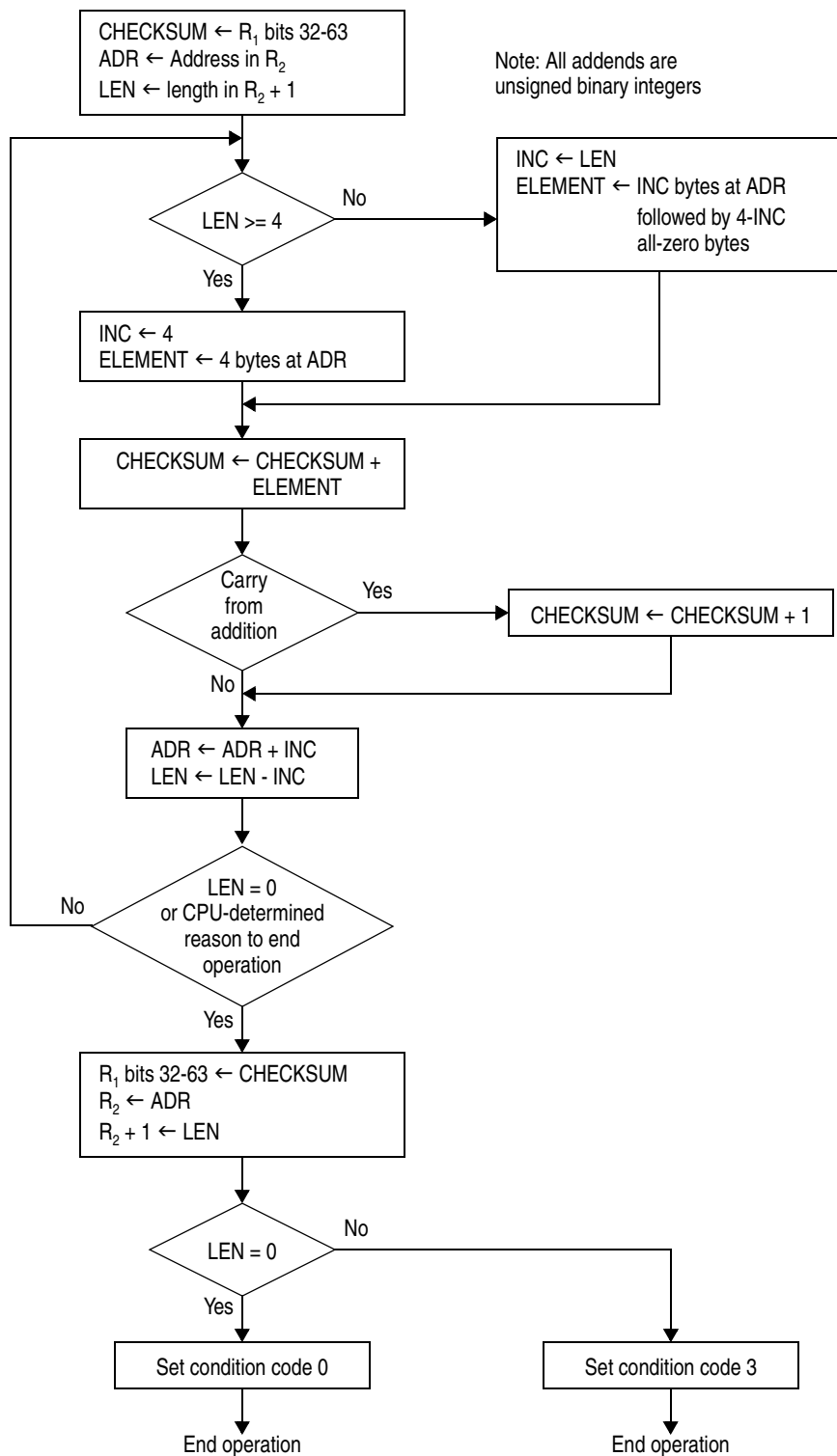
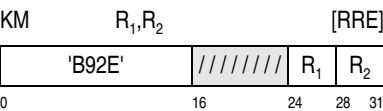
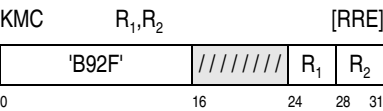


Figure 7-3. Execution of CHECKSUM

CIPHER MESSAGE



CIPHER MESSAGE WITH CHAINING



A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction are ignored.

Bit positions 57-63 of general register 0 contain the function code. Figure 7-4 and 7-5 show the assigned function codes for CIPHER MESSAGE and CIPHER MESSAGE WITH CHAINING, respectively. All other function codes are unassigned. For cipher functions, bit 56 is the modifier bit which specifies whether an encryption or a decryption operation is to be performed. The modifier bit is ignored for all other functions. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

The function codes for CIPHER MESSAGE are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	KM-Query	16	—
1	KM-DEA	8	8
2	KM-TDEA-128	16	8
3	KM-TDEA-192	24	8
9	KM-Encrypted-DEA	32	8
10	KM-Encrypted-TDEA-128	40	8
11	KM-Encrypted-TDEA-192	48	8
18	KM-AES-128	16	16
19	KM-AES-192	24	16
20	KM-AES-256	32	16
26	KM-Encrypted-AES-128	48	16
27	KM-Encrypted-AES-192	56	16
28	KM-Encrypted-AES-256	64	16
50	KM-XTS-AES-128	32	16
52	KM-XTS-AES-256	48	16
58	KM-XTS-Encrypted-AES-128	64	16
60	KM-XTS-Encrypted-AES-256	80	16
Explanation:			
—	Not applicable		

Figure 7-4. Function Codes for CIPHER MESSAGE

The function codes for CIPHER MESSAGE WITH CHAINING are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	KMC-Query	16	—
1	KMC-DEA	16	8
2	KMC-TDEA-128	24	8
3	KMC-TDEA-192	32	8
9	KMC-Encrypted-DEA	40	8
10	KMC-Encrypted-TDEA-128	48	8
11	KMC-Encrypted-TDEA-192	56	8
18	KMC-AES-128	32	16
19	KMC-AES-192	40	16

Figure 7-5. Function Codes for CIPHER MESSAGE WITH CHAINING

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
20	KMC-AES-256	48	16
67	KMC-PRNG	32	8
26	KMC-Encrypted-AES-128	64	16
27	KMC-Encrypted-AES-192	72	16
28	KMC-Encrypted-AES-256	80	16
Explanation:			
—	Not applicable		

Figure 7-5. Function Codes for CIPHER MESSAGE WITH CHAINING (Continued)

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers R_1 , R_2 , and $R_2 + 1$ are ignored for the query function.

For all other functions, the second operand is ciphered as specified by the function code using a cryptographic key in the parameter block, and the result is placed in the first-operand location. For CIPHER MESSAGE WITH CHAINING, ciphering also uses an initial chaining value in the parameter block, and the chaining value is updated as part of the operation. For XTS functions, ciphering also uses an XTS parameter in the parameter block, and the XTS parameter is updated as part of the operation.

The R_1 field designates a general register and must designate an even-numbered register; otherwise, a specification exception is recognized.

The R_2 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first and second operands is specified by the contents of the R_1 and R_2 general registers, respectively. The number of bytes in the second-operand location is specified in general register $R_2 + 1$. The first operand is the same length as the second operand.

As part of the operation, the addresses in general registers R_1 and R_2 are incremented by the number of bytes processed, and the length in general register

$R_2 + 1$ is decremented by the same number. The formation and updating of the addresses and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively, and the contents of bit positions 0-39 are ignored; bits 40-63 of the updated addresses replace the corresponding bits in general registers R_1 and R_2 , carries out of bit position 40 of the updated address are ignored, and the contents of bit positions 32-39 of general registers R_1 and R_2 are set to zeros. In the 31-bit addressing mode, the contents of bit positions 33-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively, and the contents of bit positions 0-32 are ignored; bits 33-63 of the updated addresses replace the corresponding bits in general registers R_1 and R_2 , carries out of bit position 33 of the updated address are ignored, and the content of bit position 32 of general registers R_1 and R_2 is set to zero. In the 64-bit addressing mode, the contents of bit positions 0-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively; bits 0-63 of the updated addresses replace the contents of general registers R_1 and R_2 , and carries out of bit position 0 are ignored.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 32-63 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which specifies the number of bytes in the first and second operands, and the contents of bit positions 0-31 are ignored; bits 32-63 of the updated value replace the corresponding bits in general register $R_2 + 1$. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register $R_2 + 1$ form a 64-bit unsigned binary integer which specifies the number of bytes in the first and second operands; and the updated value replaces the contents of general register $R_2 + 1$.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , R_2 , and $R_2 + 1$, always remain unchanged.

Figure 7-6 on page 7-48 shows the contents of the general registers just described.

In the access-register mode, access registers 1, R_1 , and R_2 specify the address spaces containing the parameter block, first, and second operands, respectively.

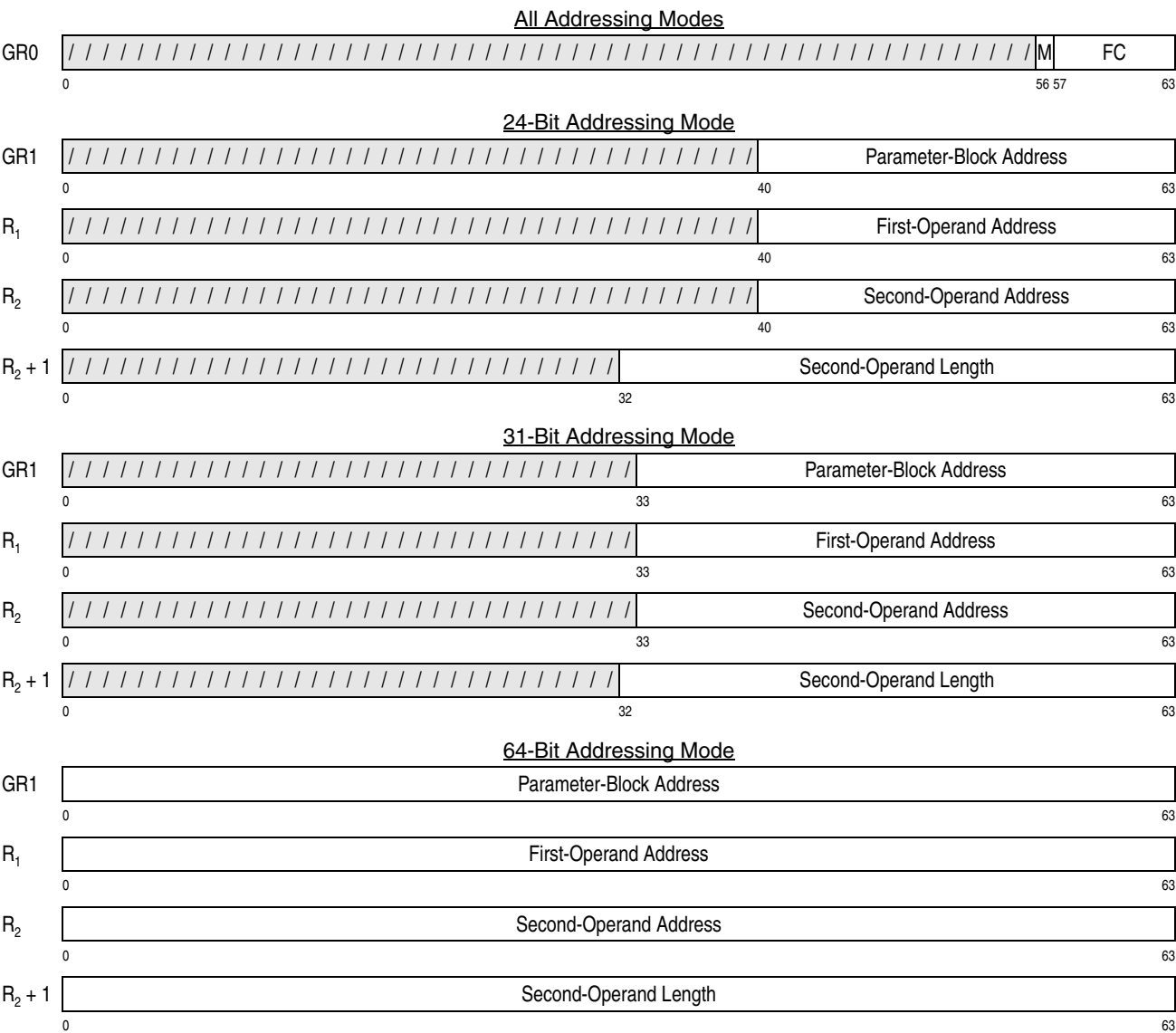


Figure 7-6. General Register Assignment for KM and KMC

The result is obtained as if processing starts at the left end of both the first and second operands and proceeds to the right, block by block. The operation is ended when the number of bytes in the second operand as specified in general register R₂ + 1 have been processed and placed at the first-operand location (called normal completion) or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually non-zero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no

progress. However, the CPU protects against endless reoccurrence of this no-progress case.

The results in the first-operand location, the chaining-value field, or the XTS parameter field are unpredictable if any of the following situations occur:

1. The cryptographic-key field or the encrypted cryptographic-key field overlaps any portion of the first operand.
2. The XTS parameter field overlaps any portion of the first operand.
3. The chaining-value field overlaps any portion of the first operand or the second operand.

4. The first and second operands overlap destructively. Operands are said to overlap destructively when the first-operand location would be used as a source after data would have been moved into it, assuming processing to be performed from left to right and one byte at a time.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in $R_2 + 1$ is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in $R_2 + 1$ is nonzero.

When a storage-alteration PER event is recognized, fewer than 4K additional bytes are stored into the first-operand locations before the event is reported.

When the second-operand length is initially zero, the parameter block, first, and second operands are not accessed, general registers R_1 , R_2 , and $R_2 + 1$ are not changed, and condition code 0 is set.

When the contents of the R_1 and R_2 fields are the same, the contents of the designated registers are incremented only by the number of bytes processed, not by twice the number of bytes processed.

As observed by other CPUs and channel programs, references to the parameter block and storage operands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

In certain unusual situations, instruction execution may complete by setting condition code 3 without updating the registers, the chaining value, and the XTS parameter to reflect the last unit of the first and second operands processed. The size of the unit processed in this case depends on the situation and the model, but is limited such that the portion of the first and second operands which have been processed and not reported do not overlap in storage. In all cases, change bits are set and PER storage-alteration events are reported, when applicable, for all first-operand locations processed.

Access exceptions may be reported for a larger portion of an operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of an operand nor for locations more than 4K bytes beyond the current location being processed.

Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the CIPHER MESSAGE and CIPHER MESSAGE WITH CHAINING functions. For data-encryption-algorithm (DEA) functions, the DEA-key-parity bit in each byte of the DEA key is ignored, and the operation proceeds normally, regardless of the DEA-key parity of the key.

Further description of the data-encryption algorithm may be found in *Data Encryption Algorithm*, ANSI-X3.92.1981, American National Standard for Information Systems. Further description of the AES standard may be found in *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, dated November 26, 2001. The XTS multiplication operation is the same as the GCM (Galois/counter mode) multiplication operation. Further description of the GCM multiplication over $GF(2^{128})$ may be found in the draft of NIST Special Publication 800-38D, dated June 2007.

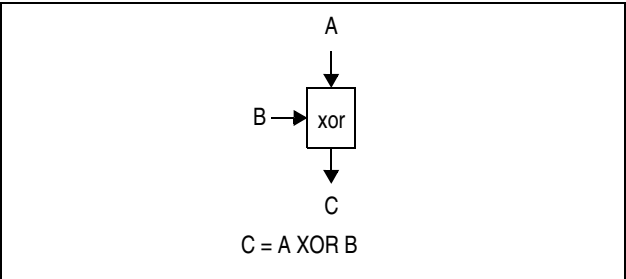


Figure 7-7. Symbol For Bit-Wise Exclusive OR

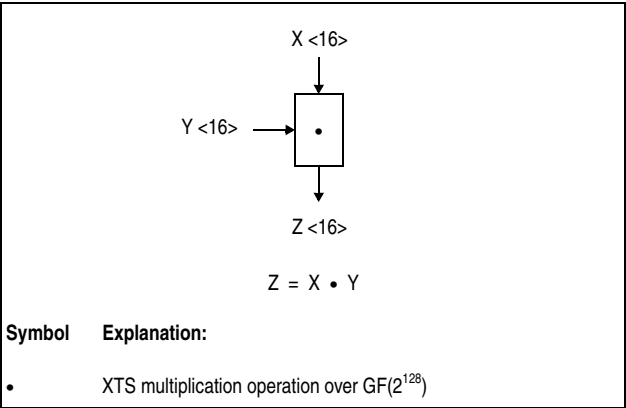


Figure 7-8. Symbol For XTS Multiplication Operation Over $GF(2^{128})$

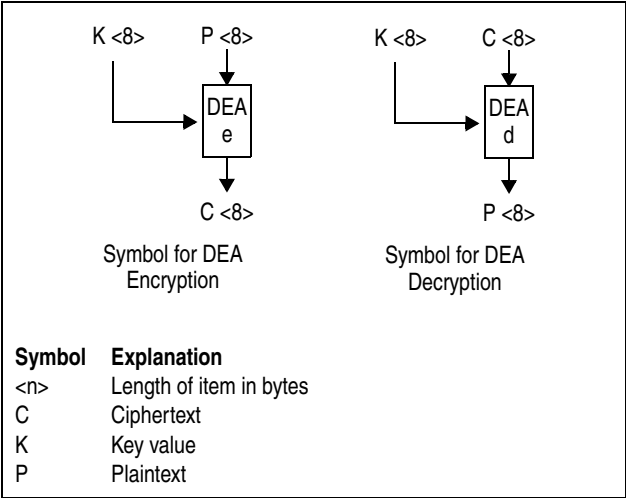


Figure 7-9. Symbols for DEA Encryption and Decryption

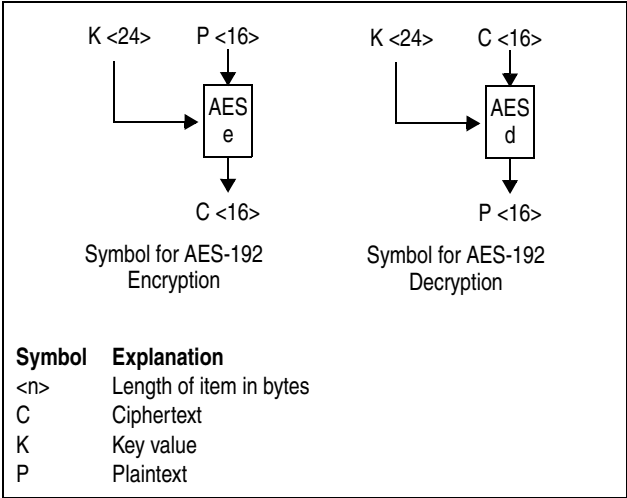


Figure 7-11. Symbols for AES-192 Encryption and Decryption

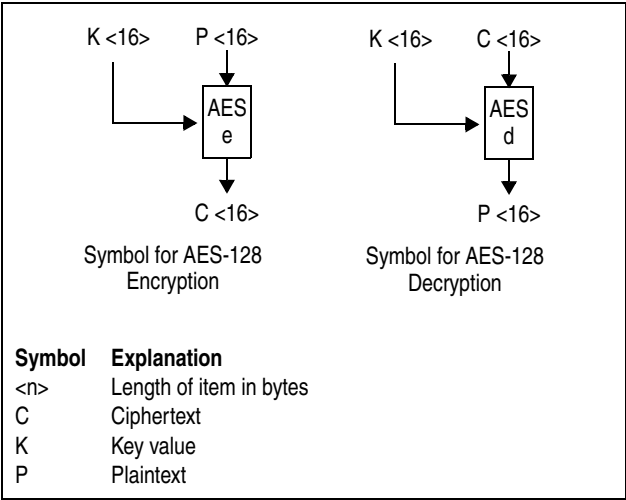


Figure 7-10. Symbols for AES-128 Encryption and Decryption

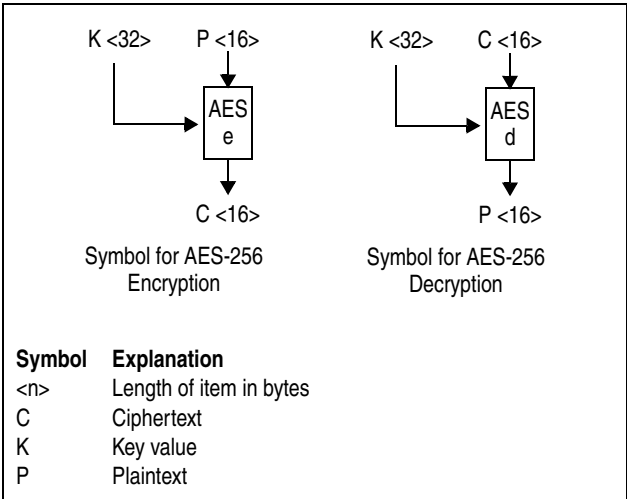


Figure 7-12. Symbols for AES-256 Encryption and Decryption

KM-Query (KM Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the function has the following format:

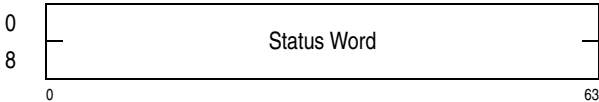


Figure 7-13. Parameter Block for KM-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function

codes 0-127, respectively, of the CIPHER MESSAGE instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KM-Query function completes; condition code 3 is not applicable to this function.

KM-DEA (KM Function Code 1)

KM-Encrypted-DEA (KM Function Code 9)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KM-DEA function has the following format:

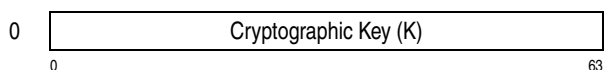


Figure 7-14. Parameter Block for KM-DEA

For the KM-DEA function, the cryptographic key is in byte offsets 0-7 of the parameter block.

The parameter block used for the KM-Encrypted-DEA function has the following format:

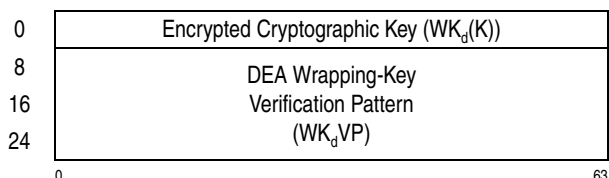


Figure 7-15. Parameter Block for KM-Encrypted-DEA

For the KM-Encrypted-DEA function, the contents of byte offsets 8-31 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-7 of the parameter block are deciphered using the DEA wrapping key to obtain the 64-bit cryptographic key. (See the section, "Protection of Cryptographic Key" on page 7-339, for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P_1, P_2, \dots, P_n) in operand 2 are enciphered using the DEA algorithm with the 64-bit cryptographic key. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C_1, C_2, \dots, C_n) are stored in operand 1. The operation is shown in Figure 7-16.

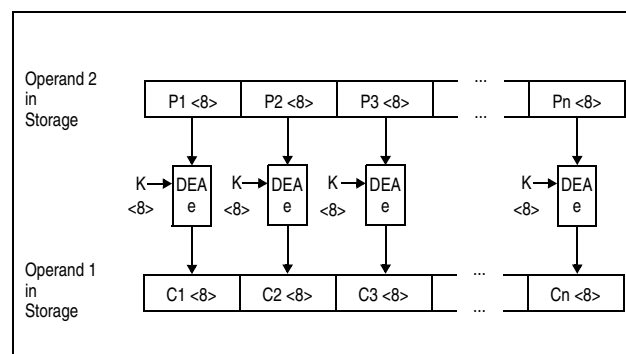


Figure 7-16. KM-DEA Encipher Operation Using 64-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C_1, C_2, \dots, C_n) in operand 2 are deciphered using the DEA algorithm with the 64-bit cryptographic key. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P_1, P_2, \dots, P_n) are stored in operand 1. The operation is shown in Figure 7-17.

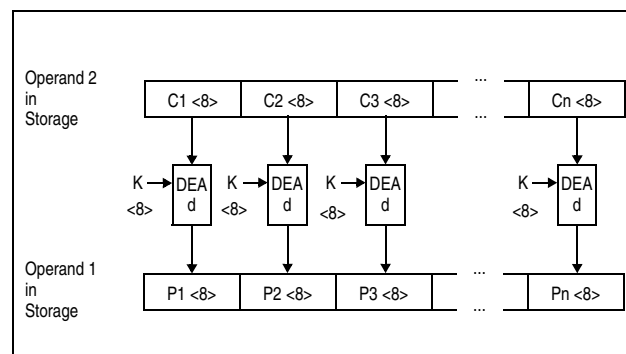


Figure 7-17. KM-DEA Decipher Operation Using 64-Bit Key

KM-TDEA-128 (KM Function Code 2)

KM-Encrypted-TDEA-128 (KM Function Code 10)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KM-TDEA-128 function has the following format:

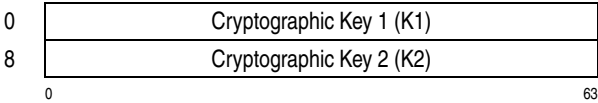


Figure 7-18. Parameter Block for KM-TDEA-128

For the KM-TDEA-128 function, the cryptographic key is in byte offsets 0-15 of the parameter block.

The parameter block used for the KM-Encrypted-TDEA-128 function has the following format:

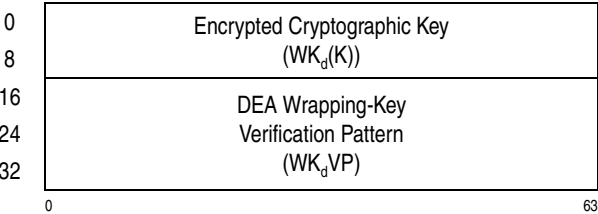


Figure 7-19. Parameter Block for KM-Encrypted-TDEA-128

For the KM-Encrypted-TDEA-128 function, the contents of byte offsets 16-39 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-15 of the parameter block are deciphered using the DEA wrapping key to obtain the 128-bit cryptographic key, $K = K1 \parallel K2$, where $K1$ is the leftmost 64 bits of K and $K2$ is the rightmost 64 bits of K . (See the section, “Protection of Cryptographic Key” on page 7-339, for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks ($P1, P2, \dots, Pn$) in operand 2 are enciphered using the TDEA (triple DEA) algorithm with the two 64-bit cryptographic keys. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks ($C1, C2, \dots, Cn$) are stored in operand 1. The operation is shown in Figure 7-20.

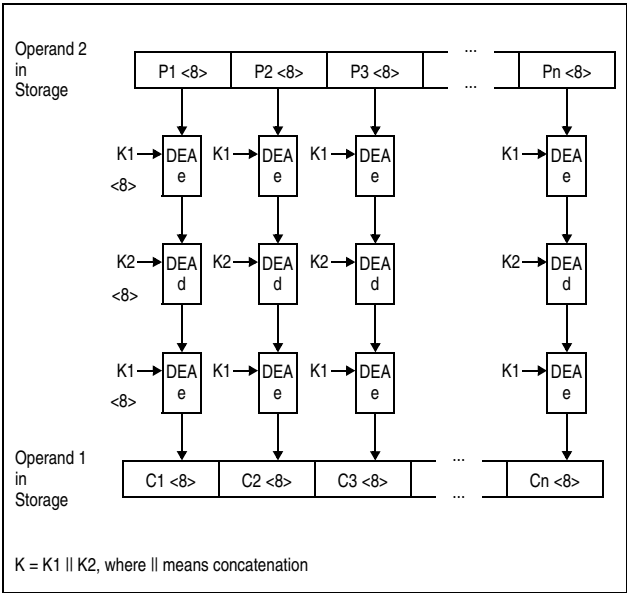


Figure 7-20. KM-TDEA Encipher Operation Using 128-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks ($C1, C2, \dots, Cn$) in operand 2 are deciphered using the TDEA algorithm with the two 64-bit cryptographic keys. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks ($P1,$

P2, ..., Pn) are stored in operand 1. The operation is shown in Figure 7-21.

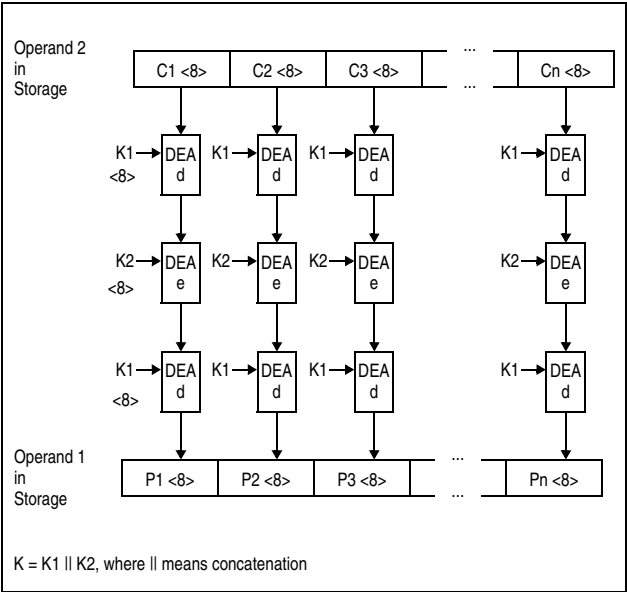


Figure 7-21. KM TDEA Decipher Operation Using 128-Bit Key

KM-TDEA-192 (KM Function Code 3)

KM-Encrypted-TDEA-192 (KM Function Code 11)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KM-TEDA-192 function has the following format:

0	Cryptographic Key 1 (K1)
8	Cryptographic Key 2 (K2)
16	Cryptographic Key 3 (K3)
0	63

Figure 7-22. Parameter Block for KM-TDEA-192

For the KM-TDEA-192 function, the cryptographic key is in byte offsets 0-23 of the parameter block.

The parameter block used for the KM-Encrypted-TDEA-192 function has the following format:

0	Encrypted Cryptographic Key (WK _d (K))
8	
16	
24	DEA Wrapping-Key Verification Pattern (WK _d VP)
32	
40	
0	63

Figure 7-23. Parameter Block for KM-Encrypted-TDEA-192

For the KM-Encrypted-TDEA-192 function, the contents of byte offsets 24-47 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-23 of the parameter block are deciphered using the DEA wrapping key to obtain the 192-bit cryptographic key, K = K1 || K2 || K3, where K1 is the leftmost 64 bits of K, K2 is the middle 64 bits of K, and K3 is the rightmost 64 bits of K. (See the section, "Protection of Cryptographic Key" on page 7-339, for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA algorithm with the three 64-bit cryptographic keys. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1,

C2, ..., Cn) are stored in operand 1. The operation is shown in Figure 7-24.

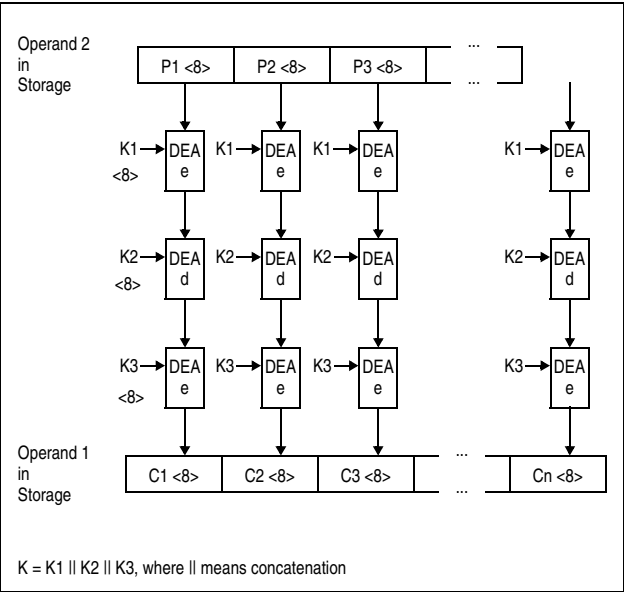


Figure 7-24. KM TDEA Encipher Operation Using 192-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA algorithm with the three 64-bit cryptographic keys. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext

blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in Figure 7-25.

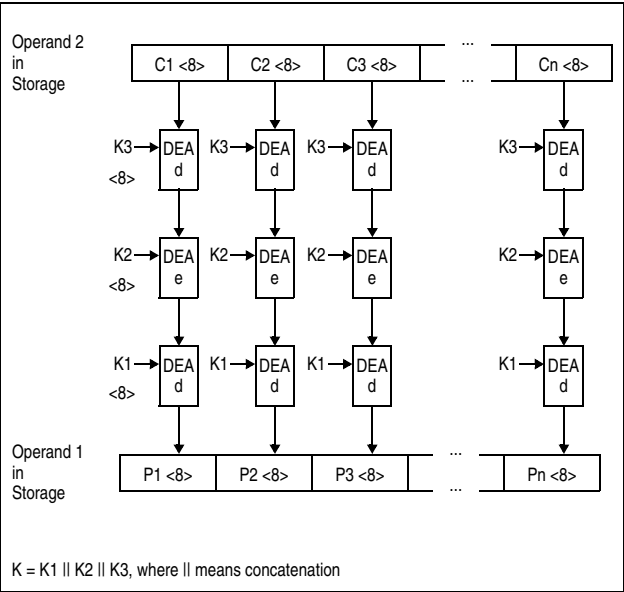


Figure 7-25. KM TDEA Decipher Operation Using 192-Bit Key

KM-AES-128 (KM Function Code 18)

KM-Encrypted-AES-128 (KM Function Code 26)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KM-AES-128 function has the following format:

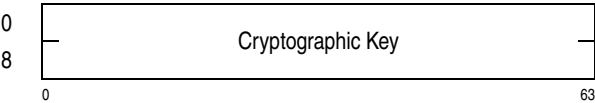


Figure 7-26. Parameter Block for KM-AES-128

For the KM-AES-128 function, the cryptographic key is in byte offsets 0-15 of the parameter block.

The parameter block used for the KM-encrypted-AES-128 function has the following format:

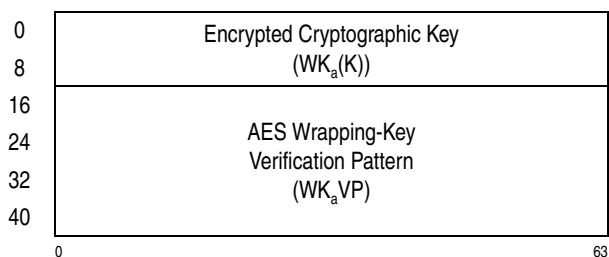


Figure 7-27. Parameter Block for KM-Encrypted-AES-128

For the KM-Encrypted-AES-128 function, the contents of byte offsets 16-47 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-15 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key, K. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES algorithm with the 128-bit cryptographic key. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in Figure 7-28.

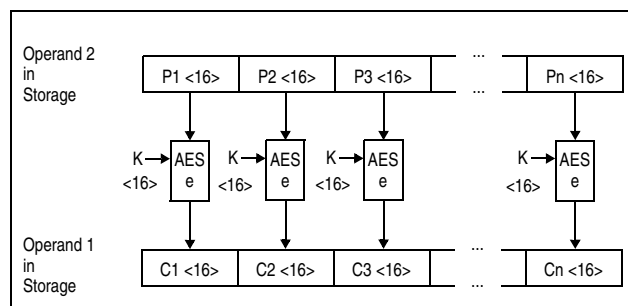


Figure 7-28. KM-AES Encipher Operation Using 128-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES algorithm with the 128-bit cryptographic key. Each ciphertext block is indepen-

dently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in Figure 7-29.

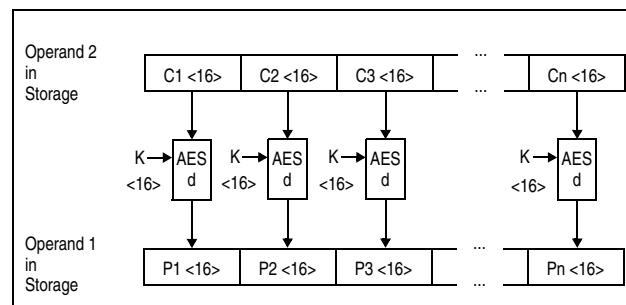


Figure 7-29. KM-AES Decipher Operation Using 128-Bit Key

KM-AES-192 (KM Function Code 19)

KM-Encrypted-AES-192 (KM Function Code 27)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KM-AES-192 function has the following format:

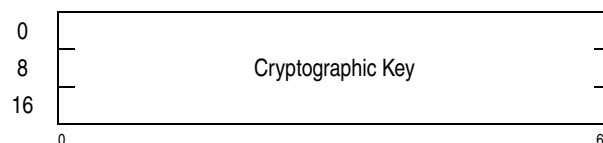


Figure 7-30. Parameter Block for KM-AES-192

For the KM-AES-192 function, the cryptographic key is in byte offsets 0-23 of the parameter block.

The parameter block used for the KM-Encrypted-AES-192 function has the following format:

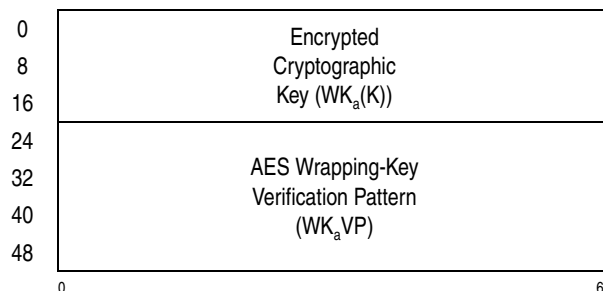


Figure 7-31. Parameter Block for KM-Encrypted-AES-192

For the KM-Encrypted-AES-192 function, the contents of byte offsets 24-55 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-23 of the parameter block are deciphered using the AES wrapping key to obtain the 192-bit cryptographic key, K. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES algorithm with the 192-bit cryptographic key. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in Figure 7-32.

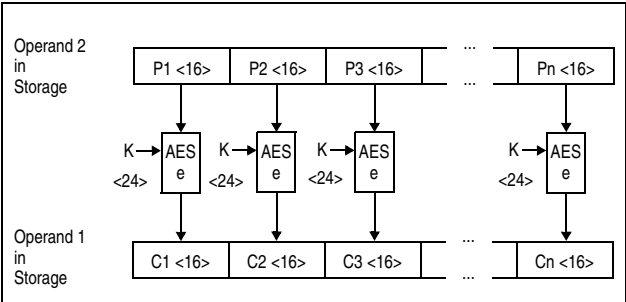


Figure 7-32. KM AES Encipher Operation Using 192-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES algorithm with the 192-bit cryptographic key. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P1,

P2, ..., Pn) are stored in operand 1. The operation is shown Figure 7-33.

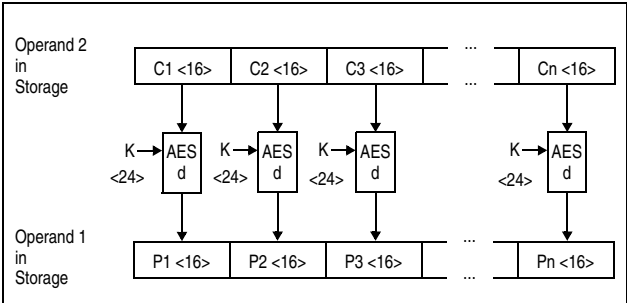


Figure 7-33. KM AES Decipher Operation Using 192-Bit Key

KM-AES-256 (KM Function Code 20)

KM-Encrypted-AES-256 (KM Function Code 28)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KM-AES-256 function has the following format:

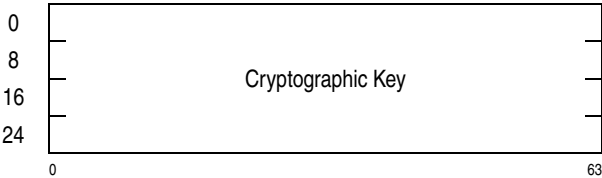


Figure 7-34. Parameter Block for KM-AES-256

For the KM-AES-256 function, the cryptographic key is in byte offsets 0-31 of the parameter block.

The parameter block used for the KM-Encrypted-AES-256 function has the following format:

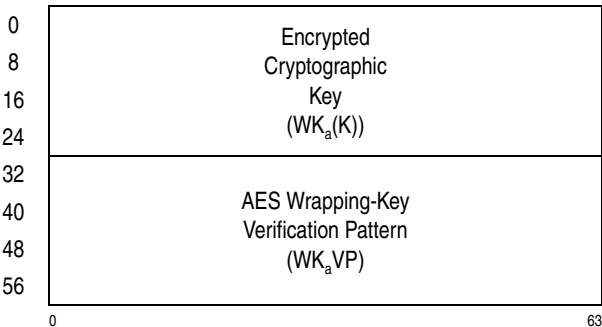


Figure 7-35. Parameter Block for KM-Encrypted-AES-256

For the KM-Encrypted-AES-256 function, the contents of byte offsets 32-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-31 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key, K. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES algorithm with the 256-bit cryptographic key. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in Figure 7-36.

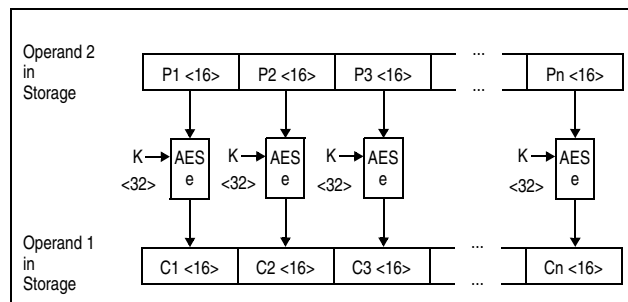


Figure 7-36. KM AES Encipher Operation Using 256-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES algorithm with the 256-bit cryptographic key. Each ciphertext block is independently deciphered; that is, the decipher operation is

performed without chaining. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in Figure 7-37.

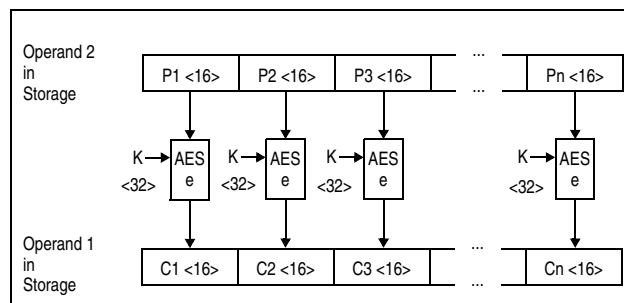


Figure 7-37. KM AES Decipher Operation Using 256-Bit Key

KM-XTS-AES-128 (KM Function Code 50)

KM-XTS-Encrypted-AES-128 (KM Function Code 58)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KM-XTS-AES-128 function has the following format:

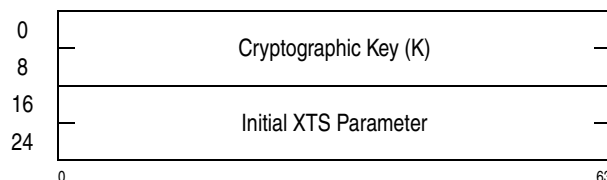


Figure 7-38. Parameter Block for KM-XTS-AES-128

For the KM-XTS-AES-128 function, the cryptographic key is in byte offsets 0-15 of the parameter block, and the initial XTS parameter is in bytes offsets 16-31 of the parameter block.

The parameter block used for the KM-XTS-encrypted-AES-128 function has the following format:

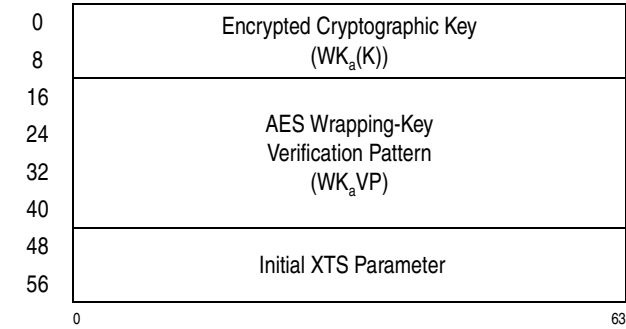


Figure 7-39. Parameter Block for KM-XTS-Encrypted-AES-128

For the KM-XTS-Encrypted-AES-128 function, the contents of byte offsets 16-47 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 48-63 of the parameter block contain the initial XTS parameter, and the contents of byte offsets 0-15 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES-encryption algorithm with the 128-bit cryptographic key and the 128-bit initial XTS parameter.

Except for the first block, the XTS parameter used is the XTS parameter for the previous block multiplied by the value of 2 in $GF(2^{128})$. To encrypt the first block of plaintext, the initial XTS parameter is used.

The XTS parameter for each block is exclusive-ORed with the corresponding ciphertext block. The result of the exclusive-OR operation is then encrypted using the AES-encryption algorithm with the 128-bit cryptographic key. The result of the encryption is then exclusive-ORed with the XTS parameter to produce the ciphertext block. The XTS parameter for this

block is multiplied by the value of 2 in $GF(2^{128})$ to obtain the XTS parameter for the next block. The operation is shown in Figure 7-40.

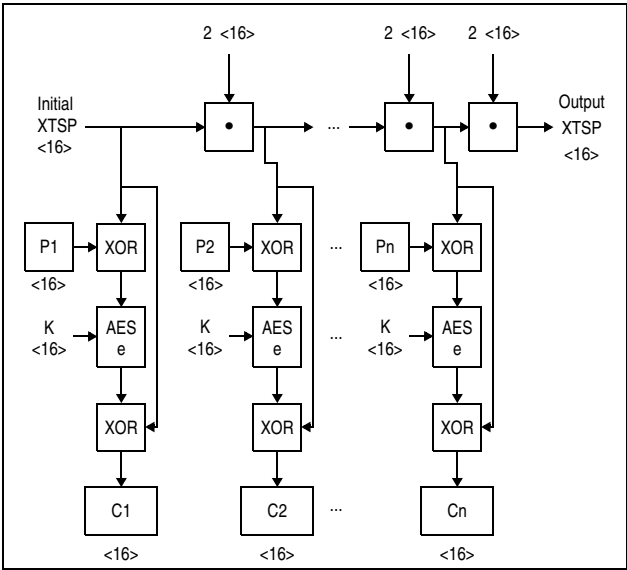


Figure 7-40. KM-XTS-AES-128 Encipher Operation

The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The XTS parameter for the next block, called output XTS parameter (output XTSP) is stored into the initial XTS parameter field of the parameter block.

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES-decryption algorithm with the 128-bit cryptographic key and the 128-bit initial XTS parameter.

Except for the first block, the XTS parameter used is the XTS parameter for the previous block multiplied by the value of 2 in $GF(2^{128})$. To decrypt the first block of ciphertext, the initial XTS parameter is used.

The XTS parameter for each block is exclusive-ORed with the corresponding ciphertext block. The result of the exclusive-OR operation is then decrypted using the AES-decryption algorithm with the 128-bit cryptographic key. The result of the encryption is then exclusive-ORed with the XTS parameter to produce the plaintext block. The XTS parameter for this block is multiplied by the value of 2 in $GF(2^{128})$ to obtain the

XTS parameter for the next block. The operation is shown in Figure 7-41.

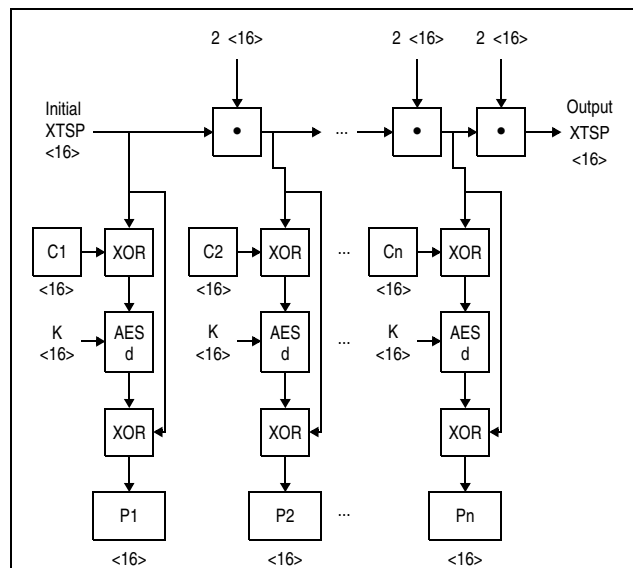


Figure 7-41. XTS-AES-128 Decipher Operation

The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The XTS parameter for the next block, called output XTS parameter (output XTSP) is stored into the initial XTS parameter field of the parameter block.

KM-XTS-AES-256 (KM Function Code 52)

KM-XTS-Encrypted-AES-256 (KM Function Code 60)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KM-XTS-AES-256 function has the following format:

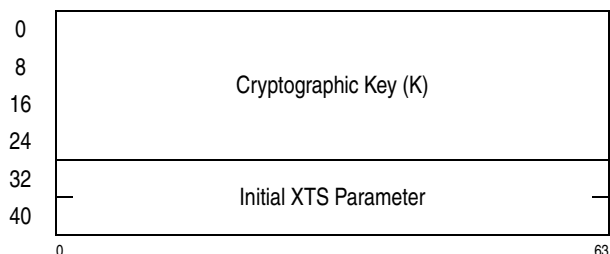


Figure 7-42. Parameter Block for KM-XTS-AES-256

For the KM-XTS-AES-256 function, the cryptographic key is in byte offsets 0-31 of the parameter block, and

the initial XTS parameter is in bytes offsets 32-47 of the parameter block.

The parameter block used for the KM-XTS-encrypted-AES-256 function has the following format:

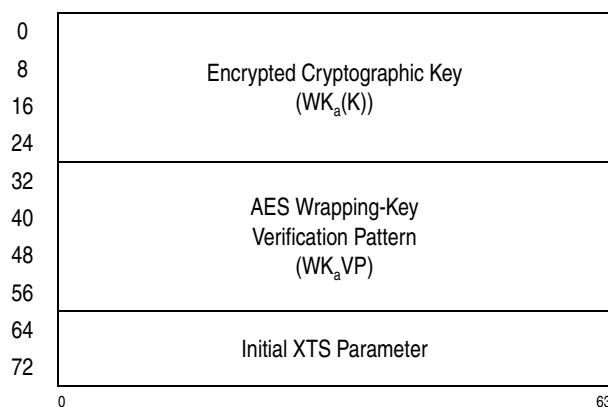


Figure 7-43. Parameter Block for KM-XTS-Encrypted-AES-256

For the KM-XTS-Encrypted-AES-256 function, the contents of byte offsets 32-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 64-79 of the parameter block contain the initial XTS parameter, and the contents of byte offsets 0-31 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES-encryption algorithm with the 256-bit cryptographic key and the 128-bit initial XTS parameter.

Except for the first block, the XTS parameter used is the XTS parameter for the previous block multiplied by the value of 2 in $GF(2^{128})$. To encrypt the first block of plaintext, the initial XTS parameter is used.

The XTS parameter for each block is exclusive-ORed with the corresponding plaintext block. The result of the exclusive-OR operation is then encrypted using the AES-encryption algorithm with the 256-bit crypto-

graphic key. The result of the encryption is then exclusive-ORed with the XTS parameter to produce the ciphertext block. The XTS parameter for this block is multiplied by the value of 2 in $GF(2^{128})$ to obtain the XTS parameter for the next block. The operation is shown in Figure 7-44.

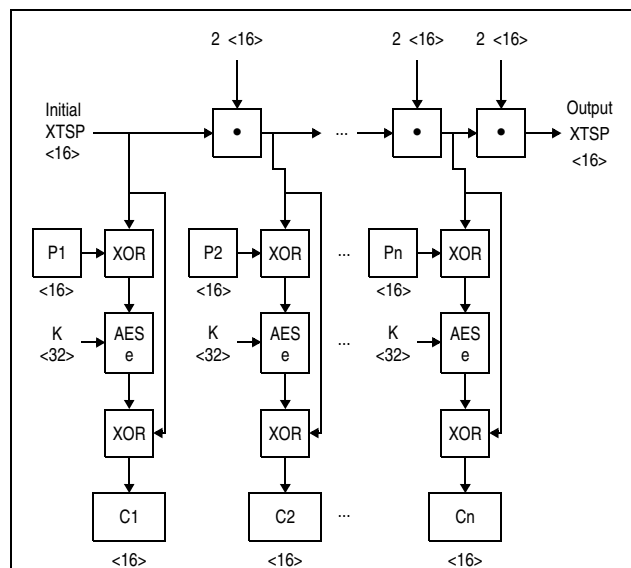


Figure 7-44. KM-XTS-AES-256 Encipher Operation

The ciphertext blocks (C_1, C_2, \dots, C_n) are stored in operand 1. The XTS parameter for the next block, called output XTS parameter (output XTSP) is stored into the initial XTS parameter field of the parameter block.

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C_1, C_2, \dots, C_n) in operand 2 are deciphered using the AES-decryption algorithm with the 256-bit cryptographic key and the 128-bit initial XTS parameter.

Except for the first block, the XTS parameter used is the XTS parameter for the previous block multiplied by the value of 2 in $GF(2^{128})$. To decrypt the first block of ciphertext, the initial XTS parameter is used.

The XTS parameter for each block is exclusive-ORed with the corresponding ciphertext block. The result of the exclusive-OR operation is then decrypted using the AES-decryption algorithm with the 256-bit cryptographic key. The result of the encryption is then exclusive-ORed with the XTS parameter to produce the plaintext block. The XTS parameter for this block is multiplied by the value of 2 in $GF(2^{128})$ to obtain the

XTS parameter for the next block. The operation is shown in Figure 7-45.

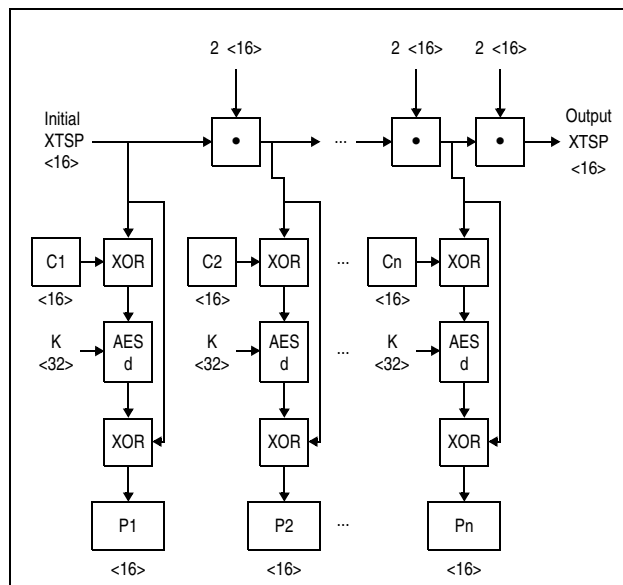


Figure 7-45. XTS-AES-128 Decipher Operation

The plaintext blocks (P_1, P_2, \dots, P_n) are stored in operand 1. The XTS parameter for the next block, called output XTS parameter (output XTSP) is stored into the initial XTS parameter field of the parameter block.

KMC-Query (KMC Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the function has the following format:

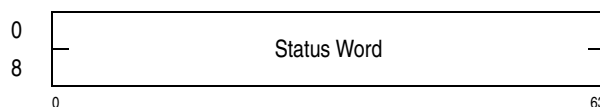


Figure 7-46. Parameter Block for KMC-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the CIPHER MESSAGE WITH CHAINING instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KMC-Query function completes; condition code 3 is not applicable to this function.

KMC-DEA (KMC Function Code 1)

KMC-Encrypted-DEA (KMC Function Code 9)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMC-DEA function has the following format:

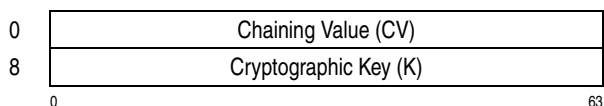


Figure 7-47. Parameter Block for KMC-DEA

For the KMC-DEA function, the chaining value is in byte offsets 0-7 of the parameter block and the cryptographic key is in byte offsets 8-15 of the parameter block.

The parameter block used for the KMC-Encrypted-DEA function has the following format:

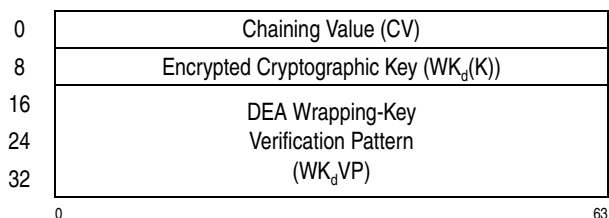


Figure 7-48. Parameter Block for KMC-Encrypted-DEA

For the KMC-Encrypted-DEA function, the contents of byte offsets 16-39 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 8-15 of the parameter block are deciphered using the DEA wrapping key to obtain the 64-bit cryptographic key. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.) The chaining value is in byte offsets 0-7 of the parameter block.

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P_1, P_2, \dots, P_n) in operand 2 are enciphered using the DEA algorithm with the 64-bit cryptographic key and the 64-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous ciphertext block. The ciphertext blocks (C_1, C_2, \dots, C_n) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-49.

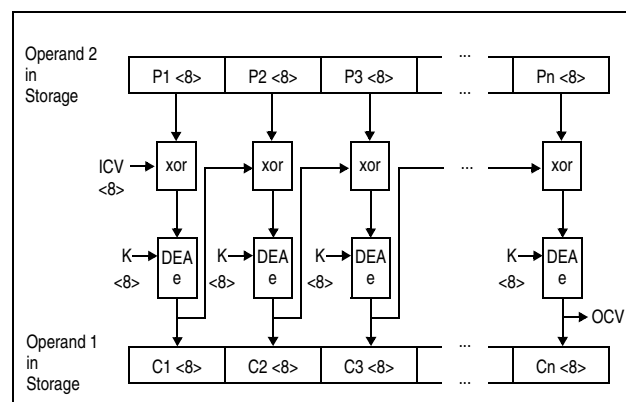


Figure 7-49. KMC DEA Encipher Operation Using 64-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C_1, C_2, \dots, C_n) in operand 2 are deciphered using the DEA algorithm with the 64-bit cryptographic key and the 64-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P_1, P_2, \dots, P_n) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is

stored into the chaining-value field in the parameter block. The operation is shown in Figure 7-50.

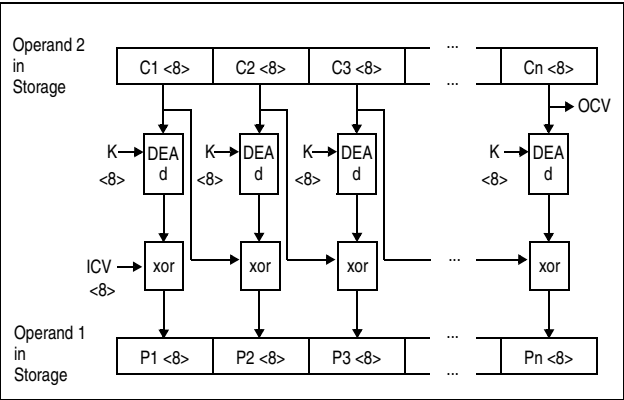


Figure 7-50. KMC DEA Decipher Operation Using 64-Bit Key

KMC-TDEA-128 (KMC Function Code 2)

KMC-Encrypted-TDEA-128 (KMC Function Code 10)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMC-TDEA-128 function has the following format:

0	Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)

Figure 7-51. Parameter Block for KMC-TDEA-128

For the KMC-TDEA-128 function, the chaining value is in byte offsets 0-7 of the parameter block and the cryptographic key is in byte offsets 8-23 of the parameter block.

The parameter block used for the KMC-Encrypted-TDEA-128 function has the following format:

0	Chaining Value (CV)
8	Encrypted Cryptographic Key (WK _d (K))
16	
24	DEA Wrapping-Key Verification Pattern (WK _d VP)
32	
40	

Figure 7-52. Parameter Block for KMC-Encrypted-TDEA-128

For the KMC-Encrypted-TDEA-128 function, the contents of byte offsets 24-47 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 8-23 of the parameter block are deciphered using the DEA wrapping key to obtain the 128-bit cryptographic key. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.) The chaining value is in byte offsets 0-7 of the parameter block.

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA algorithm with the two 64-bit cryptographic keys and the 64-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous ciphertext block. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value

field of the parameter block. The operation is shown in Figure 7-53.

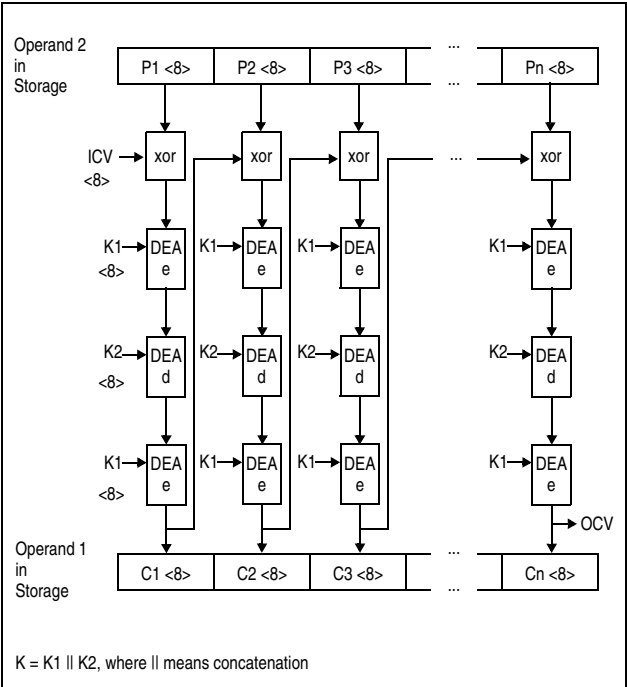


Figure 7-53. KMC TDEA Encipher Operation Using 128-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA algorithm with the two 64-bit cryptographic keys and the 64-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is

stored into the chaining-value field in the parameter block. The operation is shown in Figure 7-54.

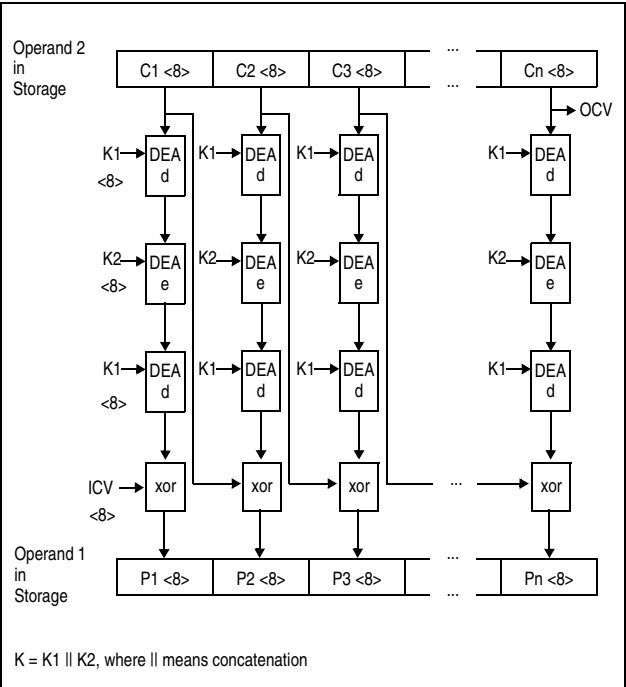


Figure 7-54. KMC TDEA Decipher Operation Using 128-Bit Key

KMC-TDEA-192 (KMC Function Code 3)

KMC-Encrypted-TDEA-192 (KMC Function Code 11)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMC-TDEA-192 function has the following format:

0	Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)
24	Cryptographic Key 3 (K3)

Figure 7-55. Parameter Block for KMC-TDEA-192

For the KMC-TDEA-192 function, the chaining value is in byte offsets 0-7 of the parameter block and the cryptographic key is in byte offsets 8-31 of the parameter block.

The parameter block used for the KMC-Encrypted-TDEA-192 function has the following format:

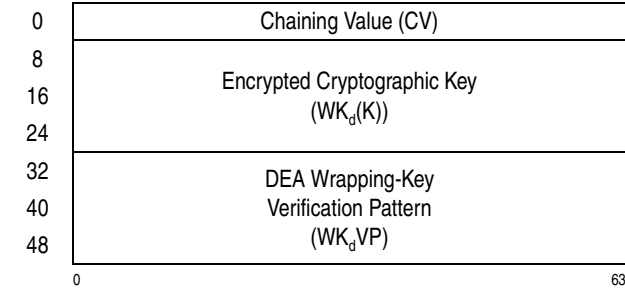


Figure 7-56. Parameter Block for KMC-Encrypted-TDEA-192

For the KMC-Encrypted-TDEA-192 function, the contents of byte offsets 32-55 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 8-31 of the parameter block are deciphered using the DEA wrapping key to obtain the 192-bit cryptographic key. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.) The chaining value is in byte offsets 0-7 of the parameter block.

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA algorithm with the three 64-bit cryptographic keys and the 64-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous ciphertext block. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The last ciphertext block is the output chaining

value (OCV) and is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-57.

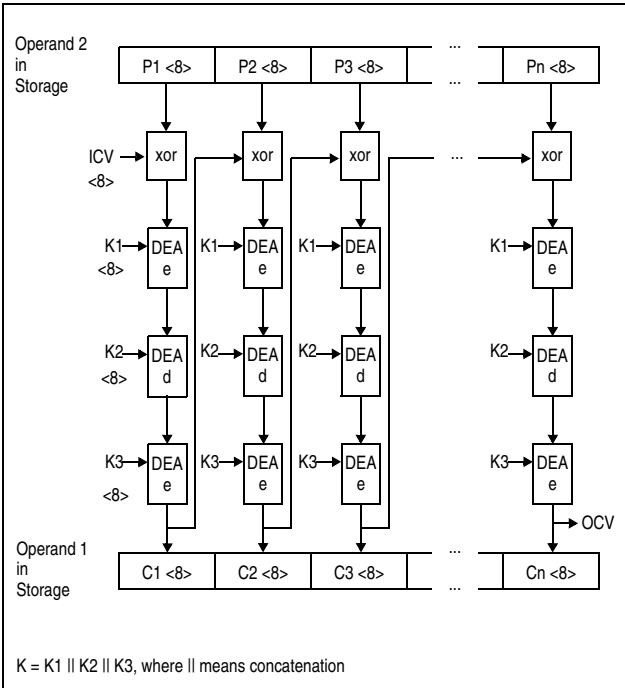


Figure 7-57. KMC TDEA Encipher Operation Using 192-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA algorithm with the three 64-bit cryptographic keys and the 64-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is

stored into the chaining-value field in the parameter block. The operation is shown in Figure 7-58.

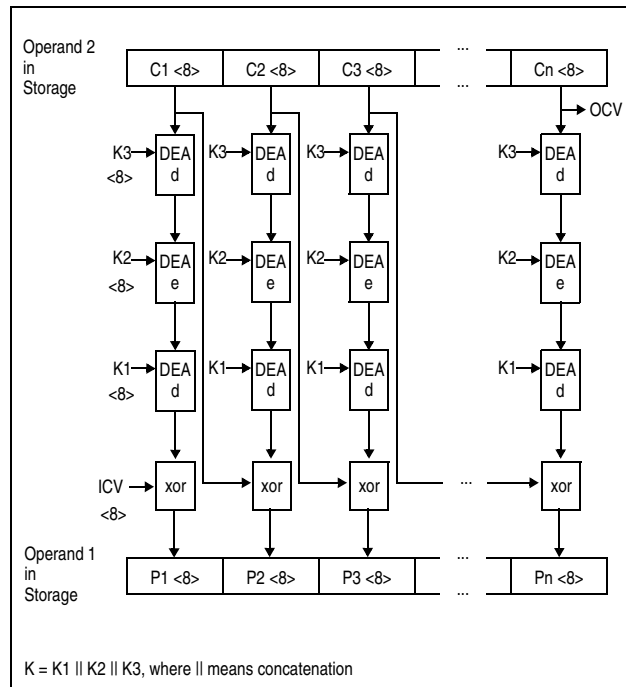


Figure 7-58. KMC TDEA Decipher Operation Using 192-Bit Key

KMC-AES-128 (KMC Function Code 18)

KMC-Encrypted-AES-128 (KMC Function Code 26)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMC-AES-128 function has the following format:

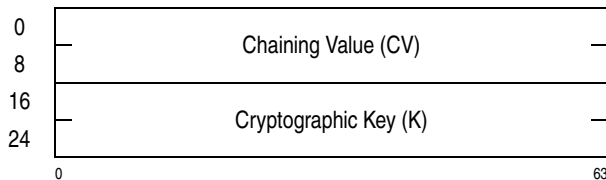


Figure 7-59. Parameter Block for KMC-AES-128

For the KMC-AES-128 function, the chaining value is in byte offsets 0-15 of the parameter block and the cryptographic key is in byte offsets 16-31 of the parameter block.

The parameter block used for the KMC-Encrypted-AES-128 function has the following format:

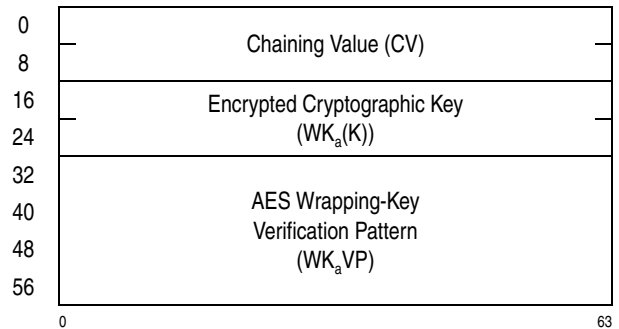


Figure 7-60. Parameter Block for KMC-Encrypted-AES-128

For the KMC-Encrypted-AES-128 function, the contents of byte offsets 32-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 16-31 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key, K. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.) The chaining value is in byte offsets 0-15 of the parameter block.

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES algorithm with the 128-bit cryptographic key and the 128-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous ciphertext block. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value

field of the parameter block. The operation is shown in Figure 7-61.

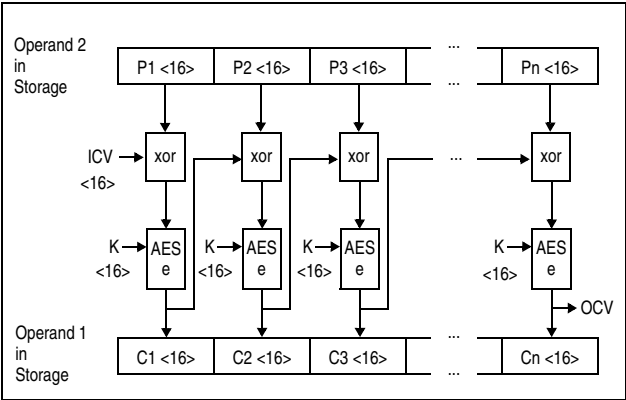


Figure 7-61. KMC AES Encipher Operation Using 128-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES algorithm with the 128-bit cryptographic key and the 128-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field in the parameter block. The operation is shown in Figure 7-62.

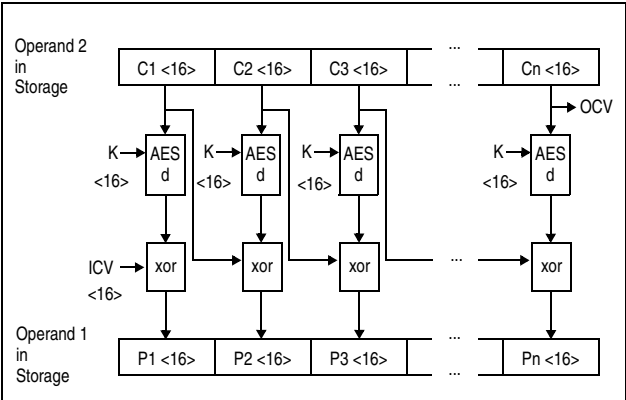


Figure 7-62. KMC AES Decipher Operation Using 128-Bit Key

KMC-AES-192 (KMC Function Code 19)

KMC-Encrypted-AES-192 (KMC Function Code 27)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMC-AES-192 function has the following format:

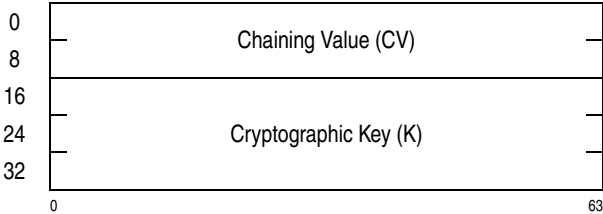


Figure 7-63. Parameter Block for KMC-AES-192

For the KMC-AES-192 function, the chaining value is in byte offsets 0-15 of the parameter block and the cryptographic key is in byte offsets 16-39 of the parameter block.

The parameter block used for the KMC-Encrypted-AES-192 function has the following format:

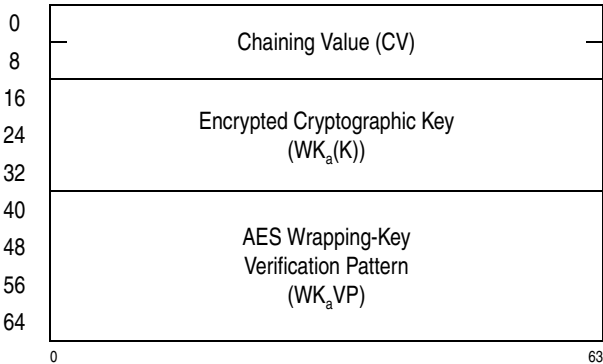


Figure 7-64. Parameter Block for KMC-Encrypted-AES-192

For the KMC-Encrypted-AES-192 function, the contents of byte offsets 40-71 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 16-39 of the parameter block are deciphered using the AES wrapping key to obtain the 192-bit cryptographic key, K. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.) The chaining value is in byte offsets 0-15 of the parameter block.

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES algorithm with the 192-bit cryptographic key and the 128-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous ciphertext block. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-65.

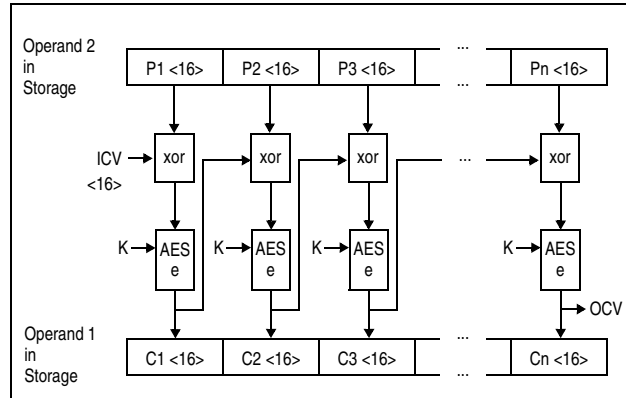


Figure 7-65. KMC AES Encipher Operation Using 192-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES algorithm with the 192-bit cryptographic key and the 128-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The last ciphertext

block is the output chaining value (OCV) and is stored into the chaining-value field in the parameter block. The operation is shown in Figure 7-66.

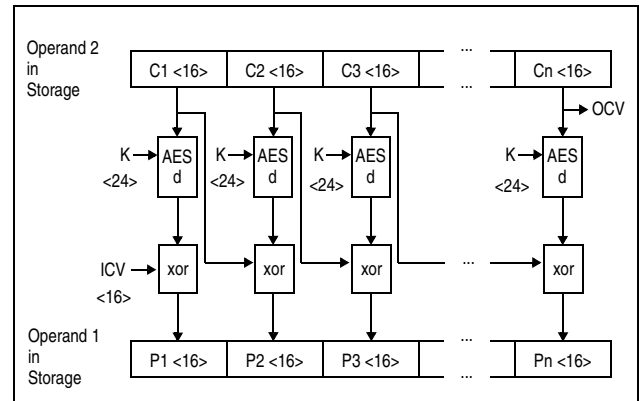


Figure 7-66. KMC AES Decipher Operation Using 192-Bit Key

KMC-AES-256 (KMC Function Code 20)

KMC-Encrypted-AES-256 (KMC Function Code 28)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMC-AES-256 function has the following format:

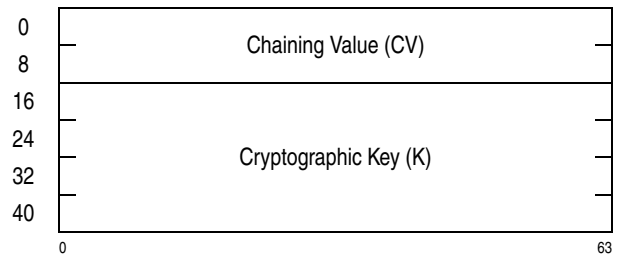


Figure 7-67. Parameter Block for KMC-AES-256

For the KMC-AES-256 function, the chaining value is in byte offsets 0-15 of the parameter block and the cryptographic key is in byte offsets 16-47 of the parameter block.

The parameter block used for the KMC-Encrypted-AES-256 function has the following format:

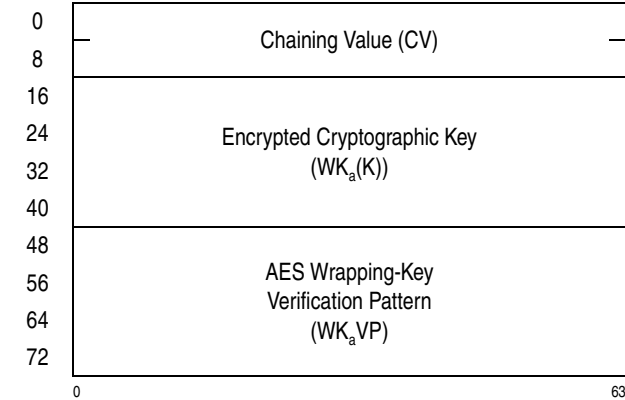


Figure 7-68. Parameter Block for KMC-Encrypted-AES-256

For the KMC-Encrypted-AES-256 function, the contents of byte offsets 48-79 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 16-47 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key, K. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.) The chaining value is in byte offsets 0-15 of the parameter block.

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES algorithm with the 256-bit cryptographic key and the 128-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first ciphertext block is the chaining value in the parameter block; the chaining value for deriving each subsequent ciphertext block is the corresponding previous ciphertext block. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value

field of the parameter block. The operation is shown in Figure 7-69.

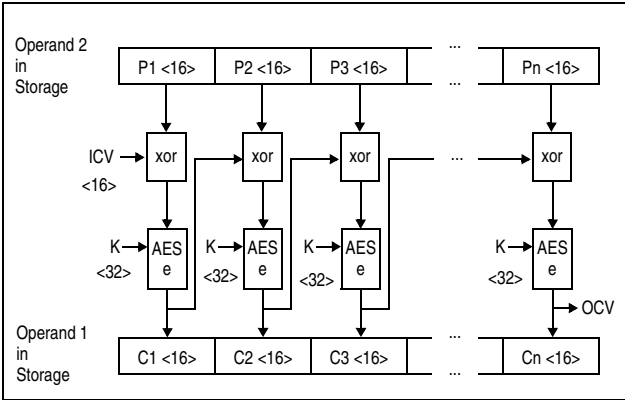


Figure 7-69. KMC AES Encipher Operation Using 256-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES algorithm with the 256-bit cryptographic key and the 128-bit chaining value.

The chaining value, called the initial chaining value (ICV), for deriving the first plaintext block is in the parameter block; the chaining value for deriving each subsequent plaintext block is the corresponding previous ciphertext block. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The last ciphertext block is the output chaining value (OCV) and is stored into the chaining-value field in the parameter block. The operation is shown in Figure 7-70.

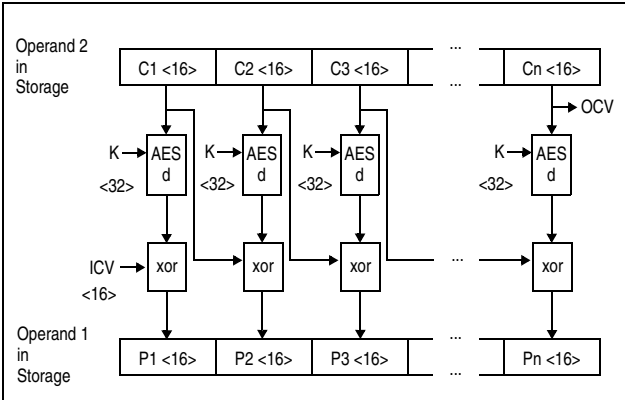


Figure 7-70. KMC AES Decipher Operation Using 256-Bit Key

KMC-PRNG (KMC Function Code 67)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the function has the following format:

0	Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)
24	Cryptographic Key 3 (K3)
0	63

Figure 7-71. Parameter Block for KMC-PRNG

Regardless of the value of the modifier bit in general register 0, the 8-byte plaintext blocks in operand 2 are used to generate the 8-byte ciphertext blocks in operand 1 based on a pseudo-random-number-generation algorithm. The algorithm is based on TDEA using three 64-bit cryptographic keys and a 64-bit chaining value. The output chaining value (OCV) is

stored in the chaining-value field of the parameter block. The operation is shown in Figure 7-72.

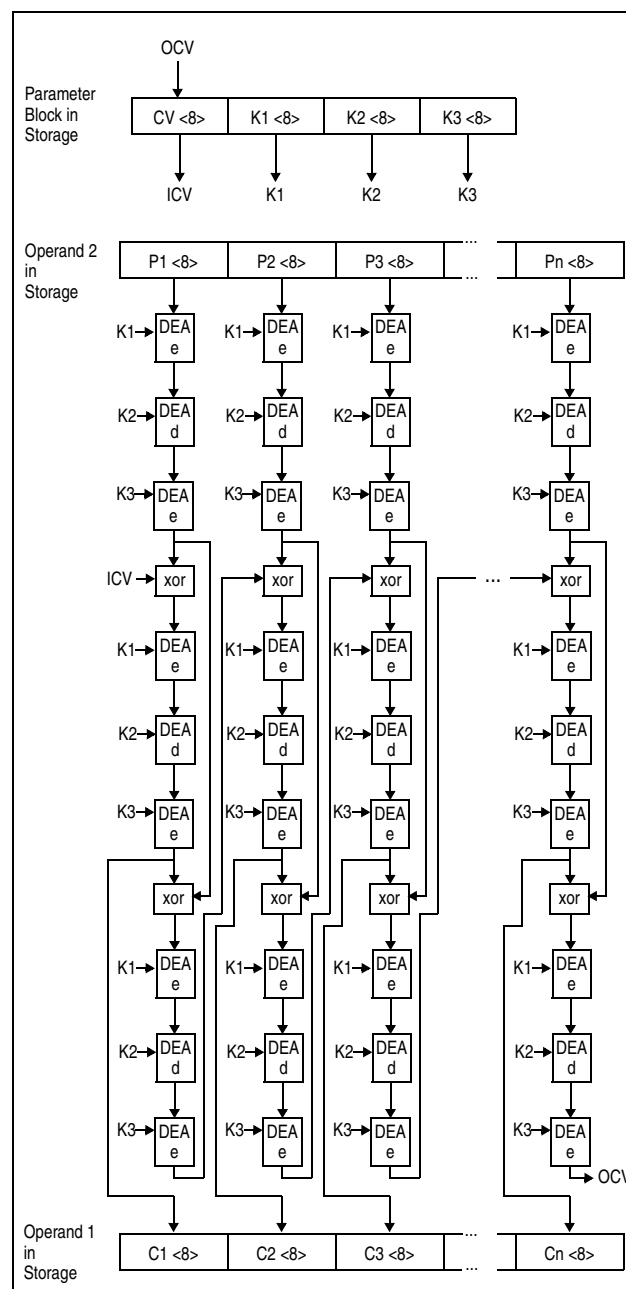


Figure 7-72. KMC-PRNG

Special Conditions for KM and KMC

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bits 57-63 of general register 0 specify an unassigned or uninstalled function code.

- 2. The R₁ or R₂ field designates an odd-numbered register or general register 0.
- 3. The second operand length is not a multiple of the data block size of the designated function (see Figure 7-4 on page 7-46 to determine the data block sizes for CIPHER MESSAGE functions; see Figure 7-5 on page 7-46 to determine the data block sizes for CIPHER MESSAGE WITH CHAINING functions). This specification-exception condition does not apply to the query functions.

Resulting Condition Code:

- 0 Normal completion
- 1 Verification pattern mismatch
- 2 --
- 3 Partial completion

Program Exceptions:

- Access (fetch, operand 2 and cryptographic key; store, operand 1; fetch and store, chaining value, XTS parameter)
- Operation (if the message-security assist is not installed)

• Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Operation exception.
8.	Specification exception due to invalid function code or invalid register number.
9.	Specification exception due to invalid operand length.
10.	Condition code 0 due to second-operand length originally zero.
11.	Access exceptions for an access to the parameter block, first, or second operand.
12.	Condition code 1 due to verification pattern mismatch.
13.	Condition code 0 due to normal completion (second-operand length originally nonzero, but stepped to zero).
14.	Condition code 3 due to partial completion (second-operand length still nonzero).

Figure 7-73. Priority of Execution: KM and KMC

Programming Notes:

- 1. When condition code 3 is set, the general registers containing the operand addresses and length, and the chaining value or the XTS parameter in the parameter block, are usually updated such that the program can simply branch back to the instruction to continue the operation.

For unusual situations, the CPU protects against endless reoccurrence of the no-progress case and also protects against setting condition code 3 when the portion of the first and second operands to be reprocessed overlap in storage. Thus, the program can safely branch back to the instruction whenever condition code 3 is set with no exposure to an endless loop and no exposure to incorrectly retrying the instruction.

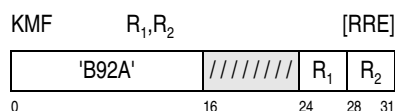
- 2. If the length of the second operand is nonzero initially and condition code 0 is set, the registers are updated in the same manner as for condition code 3. For CIPHER MESSAGE WITH CHAINING, the chaining value in this case is such that

additional operands can be processed as if they were part of the same chain.

3. To save storage, the first and second operands may overlap exactly or the starting point of the first operand may be to the left of the starting point of the second operand. In either case, the overlap is not destructive.
4. For XTS functions, the program must compute the initial XTS parameter by using the appropriate Compute-XTS-Parameter PCC functions before processing the first part of a message.
5. The initial XTS parameter used in the KM-XTS-AES-128 (or the KM-XTS-Encrypted-AES-128) function is computed by using the AES encryption algorithm with a 128-bit cryptographic key, which shall be different from the 128-bit cryptographic key used in the KM-XTS-AES-128 (or the KM-XTS-Encrypted-AES-128) function.

Similarly, the initial XTS parameter used in the KM-XTS-AES-256 (or the KM-XTS-Encrypted-AES-256) function is computed by using the AES-encryption algorithm with a 256-bit cryptographic key, which shall be different from the 256-bit cryptographic key used in the KM-XTS-AES-256 (or the KM-XTS-Encrypted-AES-256) function.

CIPHER MESSAGE WITH CFB



A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction are ignored.

Bit positions 32-39 of general register 0 contain the length of cipher feedback (LCFB) in bytes, and bit positions 57-63 of general register 0 contain the function code. Figure 7-5 shows the assigned function codes for CIPHER MESSAGE WITH CFB (cipher feedback). All other function codes are unassigned. For cipher functions, bit 56 is the modifier bit which specifies whether an encryption or a decryption operation is to be performed. The modifier bit is ignored for all other functions. Bits 0-31 and 40-55 of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

The function codes for CIPHER MESSAGE WITH CFB are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	KMF-Query	16	—
1	KMF-DEA	16	8
2	KMF-TDEA-128	24	8
3	KMF-TDEA-192	32	8
9	KMF-Encrypted-DEA	40	8
10	KMF-Encrypted-TDEA-128	48	8
11	KMF-Encrypted-TDEA-192	56	8
18	KMF-AES-128	32	16
19	KMF-AES-192	40	16
20	KMF-AES-256	48	16
26	KMF-Encrypted-AES-128	64	16
27	KMF-Encrypted-AES-192	72	16
28	KMF-Encrypted-AES-256	80	16
Explanation:			
—	Not applicable		

Figure 7-74. Function Codes for CIPHER MESSAGE WITH CFB

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers R₁, R₂, and R₂ + 1 are ignored for the query function.

For all other functions, the second operand is ciphered as specified by the function code using a cryptographic key and an initial chaining value in the parameter block, and the result is placed in the first-

operand location. The chaining value is updated as part of the operation.

The R_1 field designates a general register and must designate an even-numbered register; otherwise, a specification exception is recognized.

The R_2 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first and second operands is specified by the contents of the R_1 and R_2 general registers, respectively. The number of bytes in the second-operand location is specified in general register $R_2 + 1$. The first operand is the same length as the second operand.

As part of the operation, the addresses in general registers R_1 and R_2 are incremented by the number of bytes processed, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the addresses and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively, and the contents of bit positions 0-39 are ignored; bits 40-63 of the updated addresses replace the corresponding bits in general registers R_1 and R_2 , carries out of bit position 40 of the updated address are ignored, and the contents of bit positions 32-39 of general registers R_1 and R_2 are set to zeros. In the 31-bit addressing mode, the contents of bit positions 33-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively, and the contents of bit positions 0-32 are ignored; bits 33-63 of the updated addresses replace the corresponding bits in general registers R_1 and R_2 , carries out of bit position 33 of the updated address are ignored, and the content of bit position 32 of general registers R_1 and R_2 is set to zero. In the 64-bit addressing mode, the contents of bit positions 0-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively; bits 0-63 of the updated addresses replace the contents of general registers R_1 and R_2 , and carries out of bit position 0 are ignored.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 32-63 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which

specifies the number of bytes in the first and second operands, and the contents of bit positions 0-31 are ignored; bits 32-63 of the updated value replace the corresponding bits in general register $R_2 + 1$. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register $R_2 + 1$ form a 64-bit unsigned binary integer which specifies the number of bytes in the first and second operands; and the updated value replaces the contents of general register $R_2 + 1$.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , R_2 , and $R_2 + 1$, always remain unchanged.

Figure 7-6 on page 7-48 shows the contents of the general registers just described.

In the access-register mode, access registers 1, R_1 , and R_2 specify the address spaces containing the parameter block, first, and second operands, respectively.

The result is obtained as if processing starts at the left end of both the first and second operands and proceeds to the right, block by block. The operation is ended when the number of bytes in the second operand as specified in general register $R_2 + 1$ have been processed and placed at the first-operand location (called normal completion) or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually non-zero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against end-less reoccurrence of this no-progress case.

The results in the first-operand location and the chaining-value field are unpredictable if any of the following situations occur:

1. The cryptographic-key field or the encrypted cryptographic-key field overlaps any portion of the first operand.
2. The chaining-value field overlaps any portion of the first operand or the second operand.
3. The first and second operands overlap destructively. Operands are said to overlap destructively when the first-operand location would be used as

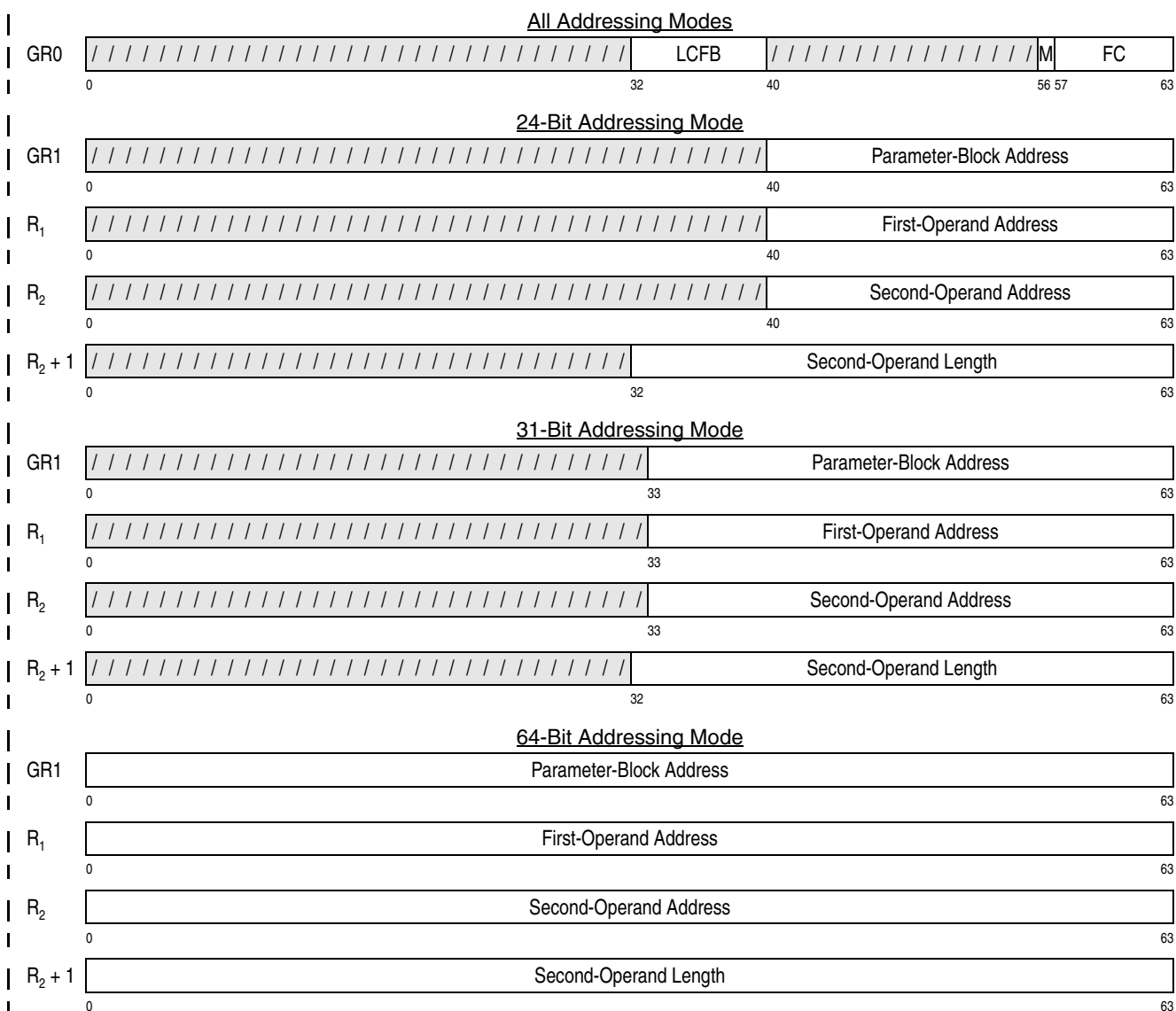


Figure 7-75. General Register Assignment for KMF

a source after data would have been moved into it, assuming processing to be performed from left to right and one byte at a time.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in R₂ + 1 is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in R₂ + 1 is nonzero.

When a storage-alteration PER event is recognized, fewer than 4K additional bytes are stored into the first-operand locations before the event is reported.

When the second-operand length is initially zero, the parameter block, first, and second operands are not

accessed, general registers R₁, R₂, and R₂ + 1 are not changed, and condition code 0 is set.

When the contents of the R₁ and R₂ fields are the same, the contents of the designated registers are incremented only by the number of bytes processed, not by twice the number of bytes processed.

As observed by other CPUs and channel programs, references to the parameter block and storage operands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

In certain unusual situations, instruction execution may complete by setting condition code 3 without updating the registers and chaining value to reflect the last unit of the first and second operands processed. The size of the unit processed in this case depends on the situation and the model, but is limited such that the portion of the first and second operands which have been processed and not reported do not overlap in storage. In all cases, change bits are set and PER storage-alteration events are reported, when applicable, for all first-operand locations processed.

Access exceptions may be reported for a larger portion of an operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of an operand nor for locations more than 4K bytes beyond the current location being processed.

Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the CIPHER MESSAGE WITH CFB functions. For data-encryption-algorithm (DEA) functions, the DEA-key-parity bit in each byte of the DEA key is ignored, and the operation proceeds normally, regardless of the DEA-key parity of the key.

Further description of the data-encryption algorithm may be found in *Data Encryption Algorithm*, ANSI-X3.92.1981, American National Standard for Information Systems. Further description of the AES standard may be found in *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, dated November 26, 2001.

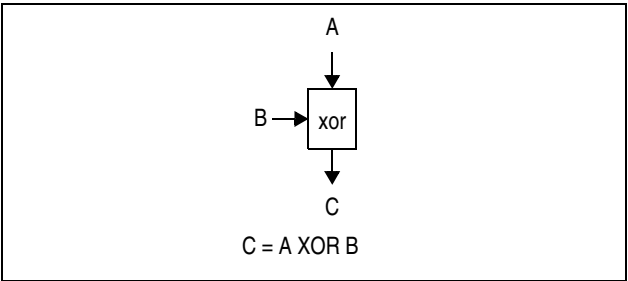


Figure 7-76. Symbol For Bit-Wise Exclusive OR

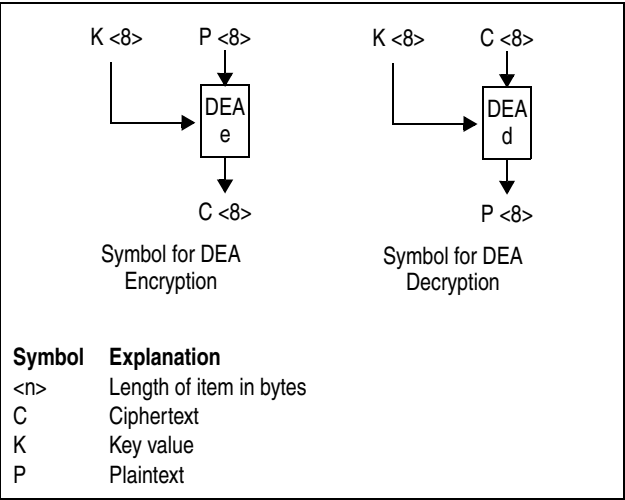


Figure 7-77. Symbols for DEA Encryption and Decryption

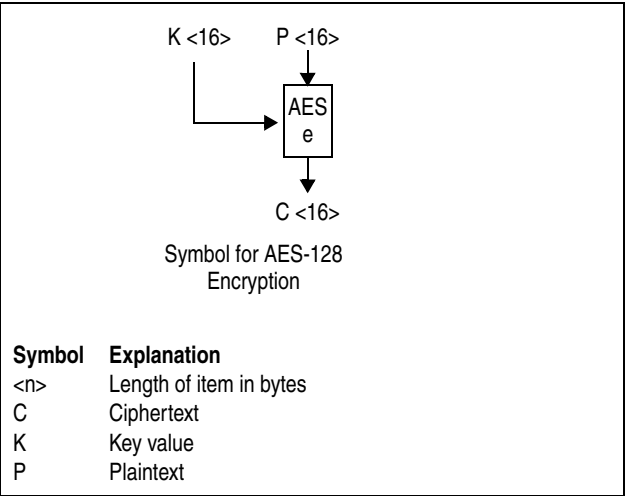


Figure 7-78. Symbols for AES-128 Encryption

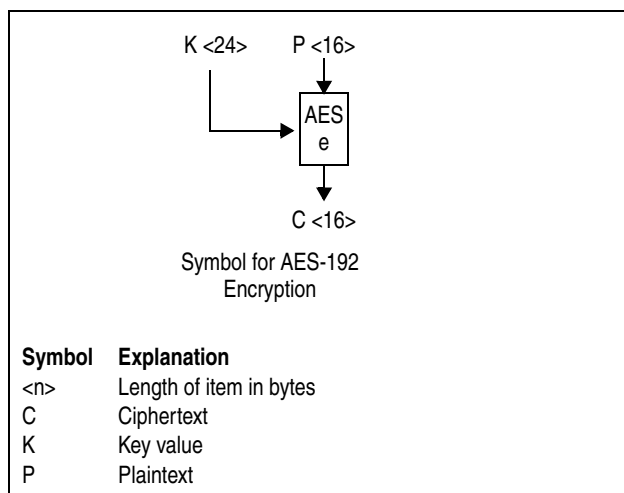


Figure 7-79. Symbols for AES-192 Encryption

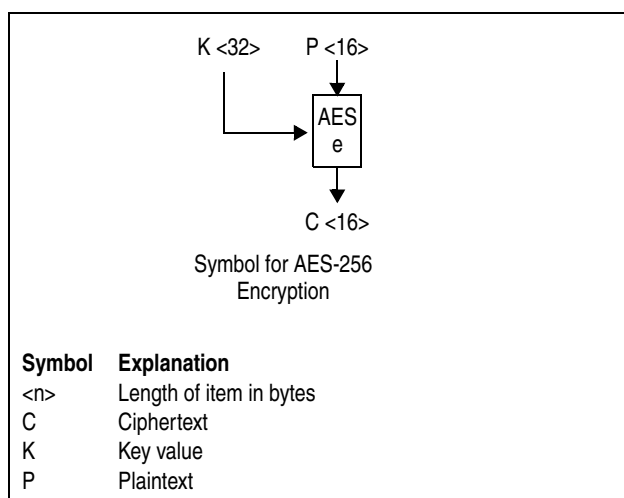


Figure 7-80. Symbols for AES-256 Encryption

KMF-Query (Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the function has the following format:

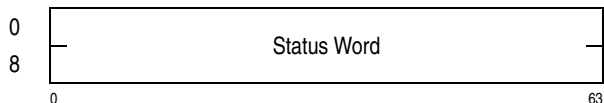


Figure 7-81. Parameter Block for KMF-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the CIPHER MESSAGE WITH CFB instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KMF-Query function completes; condition code 3 is not applicable to this function.

KMF-DEA (Function Code 1)

KMF-Encrypted-DEA (Function Code 9)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMF-DEA function has the following format:

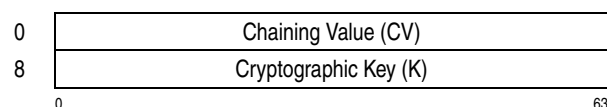


Figure 7-82. Parameter Block for KMF-DEA

For the KMF-DEA function, the chaining value is in byte offset 0-7 of the parameter block and the cryptographic key is in byte offsets 8-15 of the parameter block.

The parameter block used for the KMF-Encrypted-DEA function has the following format:

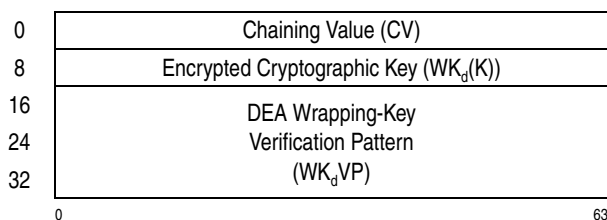


Figure 7-83. Parameter Block for KMF-Encrypted-DEA

For the KMF-Encrypted-DEA function, the contents of byte offsets 16-39 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the chaining value, and the contents of byte offsets 8-15 of the parameter block are deciphered using the DEA wrapping key to obtain the 64-bit cryp-

tographic key. (See the section, “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The s-byte plaintext segments (P1, P2, ..., Pn) in operand 2 are enciphered using the DEA-encryption algorithm with the 64-bit cryptographic key and the 64-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the DEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s-byte plaintext segment to form a s-byte ciphertext segment. The remaining (8 - s) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (8 - s) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a ciphertext segment is produced for every plaintext segment, or until a CPU-determined number of ciphertext segments have been produced.

The ciphertext segments (C1, C2, ..., Cn) are stored in operand 1. The next input block is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-49.

Symbol	Explanation
	Concatenation
b	Block size; 8 bytes for DEA
lj	Input block j to DEA
RB(lj)	Rightmost (b-s) bytes of input block j

Figure 7-84. KMF-DEA Encipher Operation (Continued)

When the modifier bit in general register 0 is one, a decipher operation is performed. The s-byte ciphertext segments (C1, C2, ..., Cn) in operand 2 are deciphered using the DEA-encryption algorithm with the 64-bit cryptographic key and the 64-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the DEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s-byte ciphertext segment to form a s-byte plaintext segment. The remaining (8 - s) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (8 - s) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a plaintext segment is produced for every ciphertext segment, or until a CPU-determined number of plaintext segments have been produced.

The plaintext segments (P1, P2, ..., Pn) are stored in operand 1. The next input block (OCV) is stored into

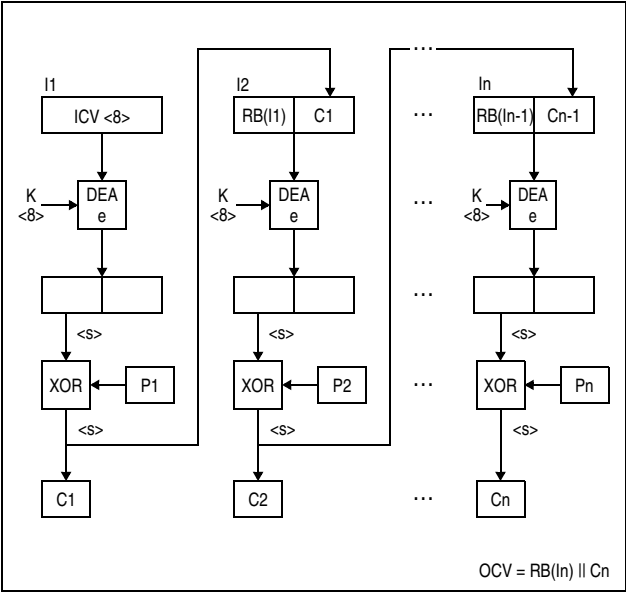


Figure 7-84. KMF-DEA Encipher Operation

the chaining-value field of the parameter block. The operation is shown in Figure 7-85.

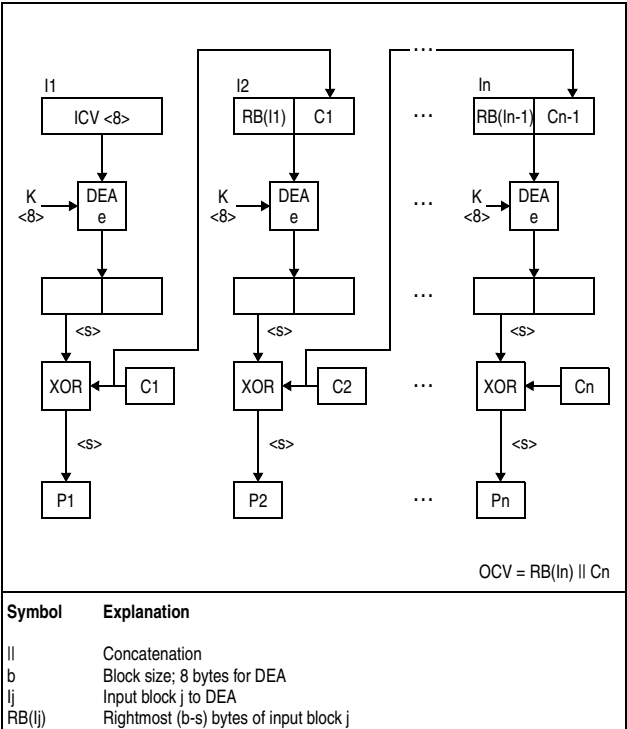


Figure 7-85. KMF-DEA Decipher Operation

KMF-TDEA-128 (Function Code 2)

KMF-Encrypted-TDEA-128 (Function Code 10)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMF-TDEA-128 function has the following format:

0	Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)
0	63

Figure 7-86. Parameter Block for KMF-TDEA-128

For the KMF-TDEA-128 function, the chaining value is in byte offset 0-7 of the parameter block and the cryptographic key is in byte offsets 8-23 of the parameter block.

The parameter block used for the KMF-Encrypted-TDEA-128 function has the following format:

0	Chaining Value (CV)
8	Encrypted Cryptographic Key (WK _e (K))
16	DEA Wrapping-Key Verification Pattern (WK _d VP)
24	
32	
40	
0	63

Figure 7-87. Parameter Block for KMF-TDEA-128

For the KMF-Encrypted-TDEA-128 function, the contents of byte offsets 24-47 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the chaining value, and the contents of byte offsets 8-23 of the parameter block are deciphered using the DEA wrapping key to obtain the 128-bit cryptographic key. (See the section, "Protection of Cryptographic Key" on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The s-byte plaintext segments (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA-encryption algorithm with the 128-bit cryptographic key and the 64-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the TDEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s-byte plaintext segment to form a s-byte ciphertext segment. The remaining (8 - s) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (8 - s) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a ciphertext segment is produced for every plaintext segment, or until a CPU-determined number of ciphertext segments have been produced.

The ciphertext segments (C1, C2, ..., Cn) are stored in operand 1. The next input block is stored into the

chaining-value field of the parameter block. The operation is shown in Figure 7-88.

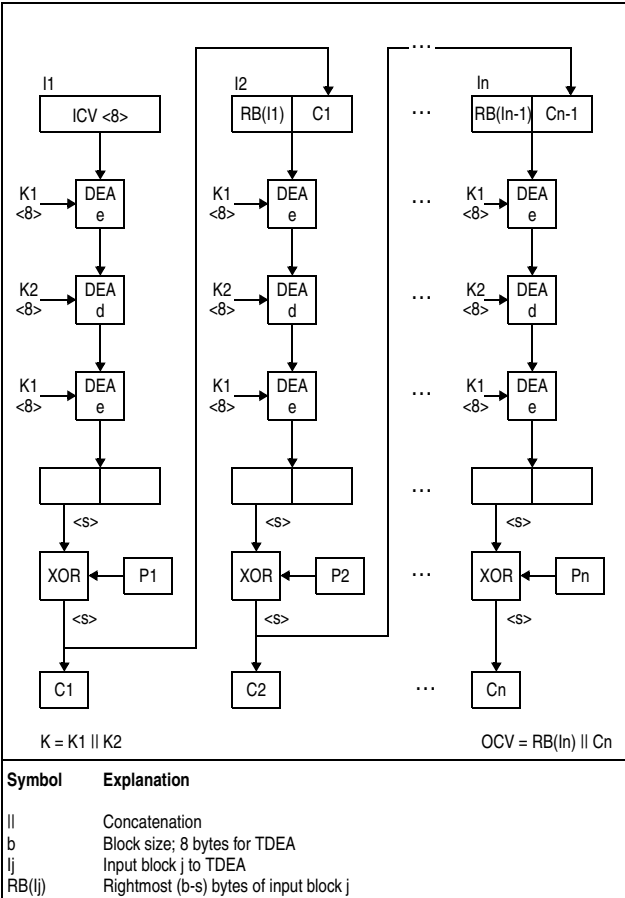


Figure 7-88. KMF-TDEA-128 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The s-byte ciphertext segments (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA-encryption algorithm with the 128-bit cryptographic key and the 64-bit chaining value, where s is the length of cipher feedback bytes.

The first input block to the TDEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s-byte ciphertext segment to form a s-byte plaintext segment. The remaining (8 - s) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (8 - s) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a plaintext segment is produced for every ciphertext segment, or until a CPU-determined number of plaintext segments have been produced.

The plaintext segments (P1, P2, ..., Pn) are stored in operand 1. The next input block (OCV) is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-89.

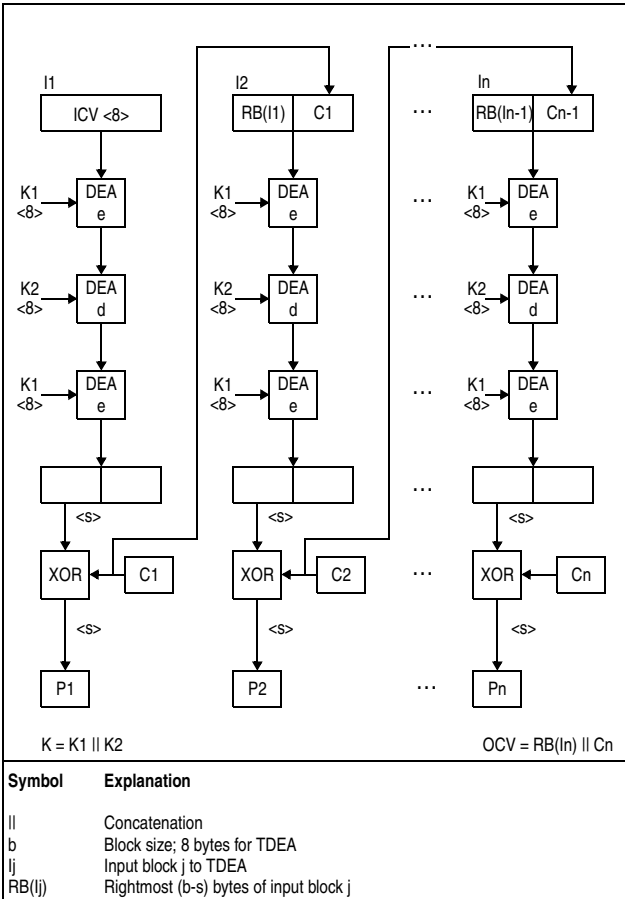


Figure 7-89. KMF-TDEA-128 Decipher Operation

KMF-TDEA-192 (Function Code 3)

KMF-Encrypted-TDEA-192 (Function Code 11)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMF-TDEA-192 function has the following format:

0	Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)
24	Cryptographic Key 3 (K3)
0	63

Figure 7-90. Parameter Block for KMF-TDEA-192

For the KMF-TDEA-192 function, the chaining value is in byte offset 0-7 of the parameter block and the cryptographic key is in byte offsets 8-31 of the parameter block.

The parameter block used for the KMF-Encrypted-TDEA-192 function has the following format:

0	Chaining Value (CV)
8	Encrypted Cryptographic Key ($WK_d(K)$)
16	
24	
32	DEA Wrapping-Key Verification Pattern (WK_dVP)
40	
48	
0	63

Figure 7-91. Parameter Block for KMF-Encrypted-TDEA-192

For the KMF-Encrypted-TDEA-192 function, the contents of byte offsets 32-55 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the chaining value, and the contents of byte offsets 8-31 of the parameter block are deciphered using the DEA wrapping key to obtain the 192-bit cryptographic key. (See the section, "Protection of Cryptographic Key" on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The s -byte plaintext segments (P_1, P_2, \dots, P_n) in operand 2 are enciphered using the TDEA-encryption algorithm with the 192-bit cryptographic key and the 64-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the TDEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s -byte plaintext segment to form a s -byte ciphertext segment. The remaining $(8 - s)$ bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the $(8 - s)$ rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a ciphertext segment is produced for every plaintext segment, or until a CPU-determined number of ciphertext segments have been produced.

The ciphertext segments (C_1, C_2, \dots, C_n) are stored in operand 1. The next input block (OCV) is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-92.

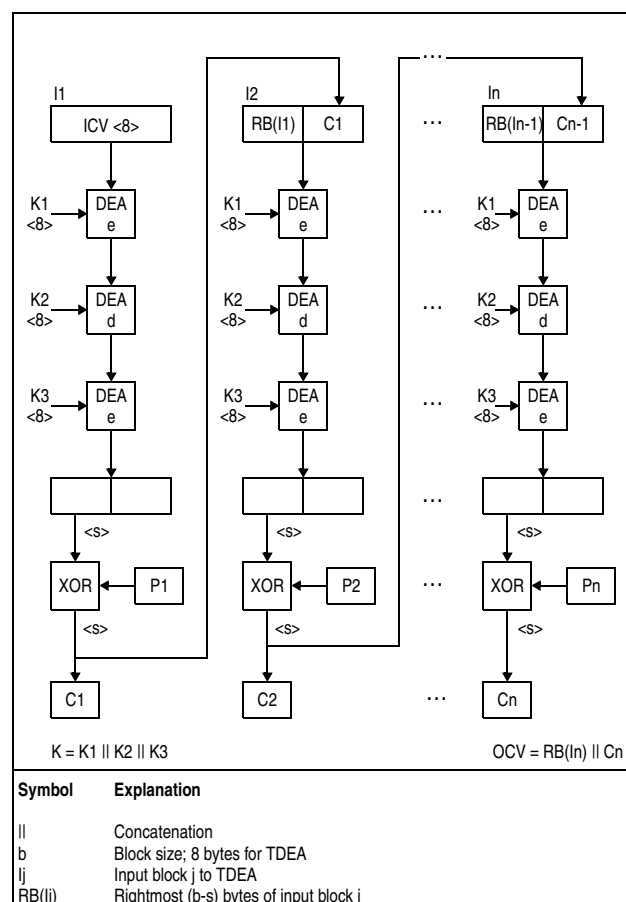


Figure 7-92. KMF-TDEA-192 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The s -byte cipher-

text segments (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA-encryption algorithm with the 192-bit cryptographic key and the 64-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the TDEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s-byte ciphertext segment to form a s-byte plaintext segment. The remaining (8 - s) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (8 - s) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a plaintext segment is produced for every ciphertext segment, or until a CPU-determined number of plaintext segments have been produced.

The plaintext segments (P1, P2, ..., Pn) are stored in operand 1. The next input block is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-93.

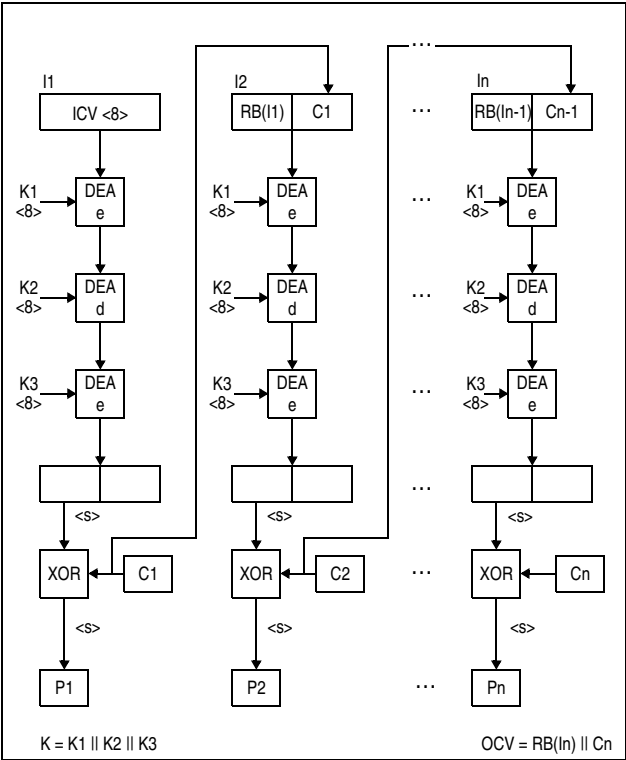


Figure 7-93. KMF-TDEA-192 Decipher Operation

Symbol	Explanation
	Concatenation
b	Block size; 8 bytes for TDEA
Ij	Input block j to TDEA
RB(Ij)	Rightmost (b-s) bytes of input block j

Figure 7-93. KMF-TDEA-192 Decipher Operation

KMF-AES-128 (Function Code 18)

KMF-Encrypted-AES-128 (Function Code 26)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMF-AES-128 function has the following format:

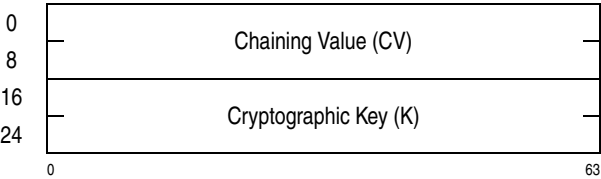


Figure 7-94. Parameter Block for KMF-AES-128

For the KMF-AES-128 function, the chaining value is in byte offset 0-15 of the parameter block and the cryptographic key is in byte offsets 16-31 of the parameter block.

The parameter block used for the KMF-Encrypted-AES-128 function has the following format:

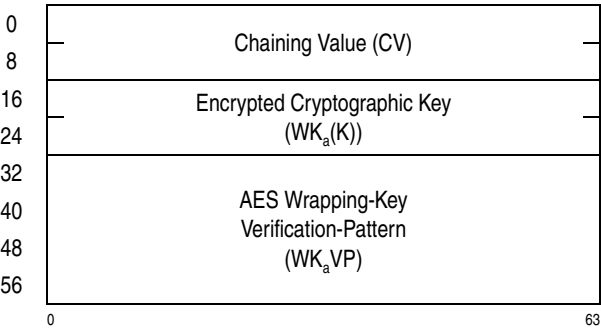


Figure 7-95. Parameter Block for KMF-Encrypted-AES-128

For the KMF-Encrypted-AES-128 function, the contents of byte offsets 32-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block

contain the chaining value, and the contents of byte offsets 16-31 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The *s*-byte plaintext segments (*P*1, *P*2, ..., *P**n*) in operand 2 are enciphered using the AES-encryption algorithm with the 128-bit cryptographic key and the 128-bit chaining value, where *s* is the length of cipher feedback in bytes.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The *s* leftmost bytes of each output block are exclusive-ORed with the corresponding *s*-byte plaintext segment to form a *s*-byte ciphertext segment. The remaining (16 - *s*) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (16 - *s*) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a ciphertext segment is produced for every plaintext segment, or until a CPU-determined number of ciphertext segments have been produced.

The ciphertext segments (*C*1, *C*2, ..., *C**n*) are stored in operand 1. The next input block is stored into the

chaining-value field of the parameter block. The operation is shown in Figure 7-96.

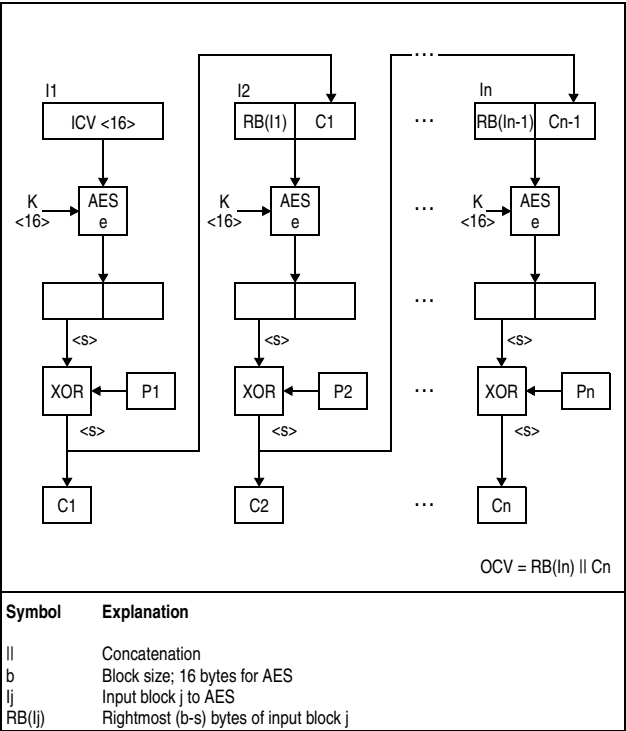


Figure 7-96. KMF-AES-128 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The *s*-byte ciphertext segments (*C*1, *C*2, ..., *C**n*) in operand 2 are deciphered using the AES-encryption algorithm with the 128-bit cryptographic key and the 128-bit chaining value, where *s* is the length of cipher feedback in bytes.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The *s* leftmost bytes of each output block are exclusive-ORed with the corresponding *s*-byte ciphertext segment to form a *s*-byte plaintext segment. The remaining (16 - *s*) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (16 - *s*) rightmost bytes

of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a plaintext segment is produced for every ciphertext segment, or until a CPU-determined number of plaintext segments have been produced.

The plaintext segments (P1, P2, ..., Pn) are stored in operand 1. The next input block (OCV) is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-97.

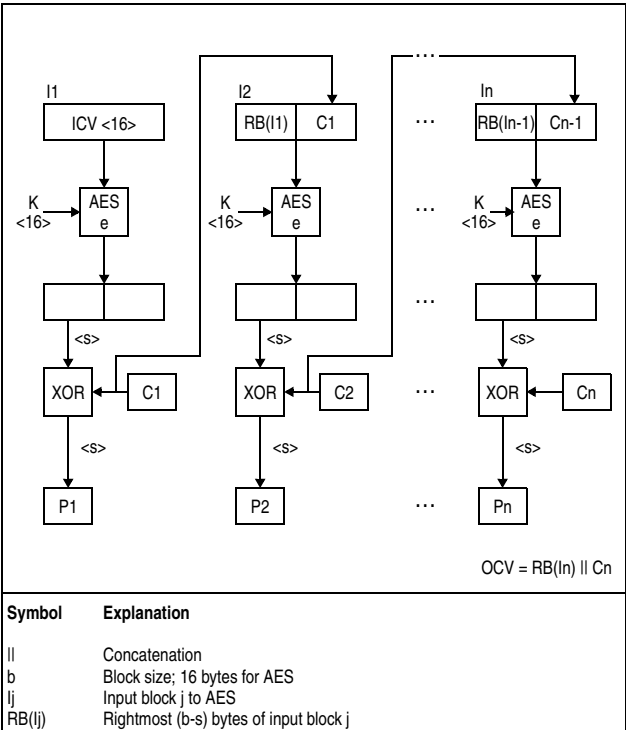


Figure 7-97. KMF-AES-128 Decipher Operation

KMF-AES-192 (Function Code 19)

KMF-Encrypted-AES-192 (Function Code 27)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMF-AES-192 function has the following format:

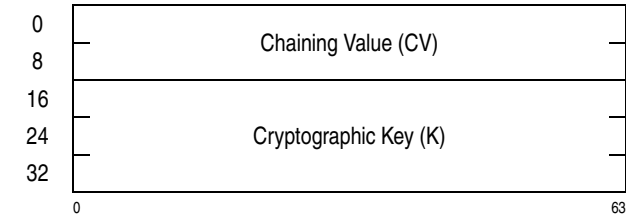


Figure 7-98. Parameter Block for KMF-AES-192

For the KMF-AES-192 function, the chaining value is in byte offset 0-15 of the parameter block and the

cryptographic key is in byte offsets 16-39 of the parameter block.

The parameter block used for the KMF-Encrypted-AES-192 function has the following format:

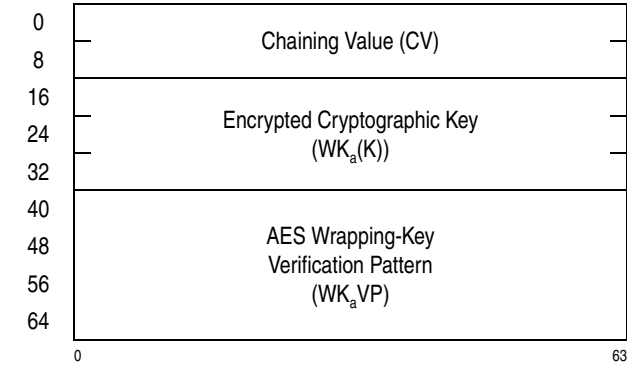


Figure 7-99. Parameter Block for KMF-Encrypted-AES-192

For the KMF-Encrypted-AES-192 function, the contents of byte offsets 40-71 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block contain the chaining value, and the contents of byte offsets 16-39 of the parameter block are deciphered using the AES wrapping key to obtain the 192-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The s-byte plaintext segments (P1, P2, ..., Pn) in operand 2 are enciphered using the AES-encryption algorithm with the 192-bit cryptographic key and the 128-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s-byte plaintext segment to form a s-byte ciphertext segment. The remaining (16 - s) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (16 - s) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a ciphertext segment is produced for every plaintext segment, or until a CPU-determined number of ciphertext segments have been produced.

The ciphertext segments (C1, C2, ..., Cn) are stored in operand 1. The next input block (OCV) is stored into the

chaining-value field of the parameter block. The operation is shown in Figure 7-100.

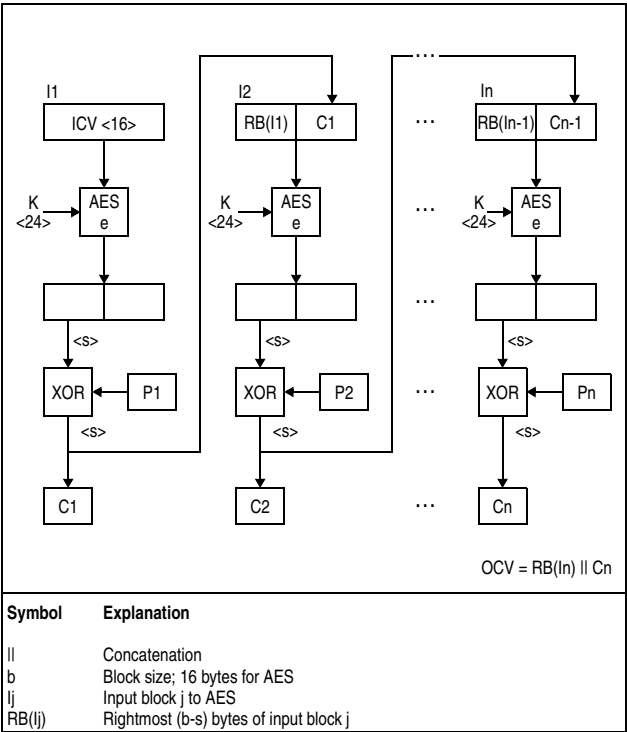


Figure 7-100. KMF-AES-192 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The s -byte ciphertext segments (C1, C2, ..., Cn) in operand 2 are deciphered using the AES-encryption algorithm with the 192-bit cryptographic key and the 128-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s -

byte ciphertext segment to form a s -byte plaintext segment. The remaining $(16 - s)$ bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the $(16 - s)$ rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a plaintext segment is produced for every ciphertext segment, or until a CPU-determined number of plaintext segments have been produced.

The plaintext segments (P1, P2, ..., Pn) are stored in operand 1. The next input block (OCV) is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-101.

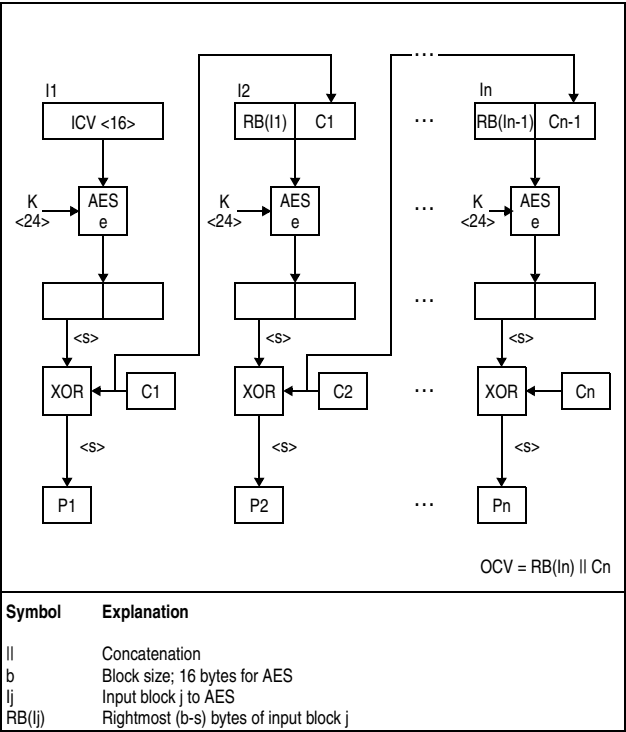


Figure 7-101. KMF-AES-192 Decipher Operation

KMF-AES-256 (Function Code 20)

KMF-Encrypted-AES-256 (Function Code 28)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-6 on page 7-48.

The parameter block used for the KMF-AES-256 function has the following format:

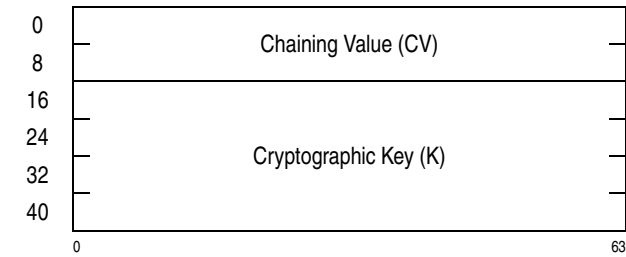


Figure 7-102. Parameter Block for KMF-AES-256

For the KMF-AES-256 function, the chaining value is in byte offset 0-15 of the parameter block and the cryptographic key is in byte offsets 16-47 of the parameter block.

The parameter block used for the KMF-Encrypted-AES-256 function has the following format:

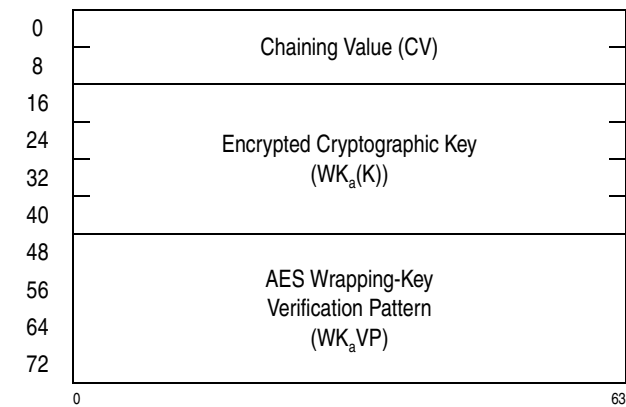


Figure 7-103. Parameter Block for KMF-AES-256

For the KMF-Encrypted-AES-256 function, the contents of byte offsets 48-79 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block contain the chaining value, and the contents of byte offsets 16-47 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The s-byte plaintext segments (P1, P2, ..., Pn) in operand 2 are enci-

phered using the AES-encryption algorithm with the 256-bit cryptographic key and the 128-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s-byte plaintext segment to form a s-byte ciphertext segment. The remaining (16 - s) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (16 - s) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a ciphertext segment is produced for every plaintext segment, or until a CPU-determined number of ciphertext segments have been produced.

The ciphertext segments (C1, C2, ..., Cn) are stored in operand 1. The next input block (OCV) is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-104.

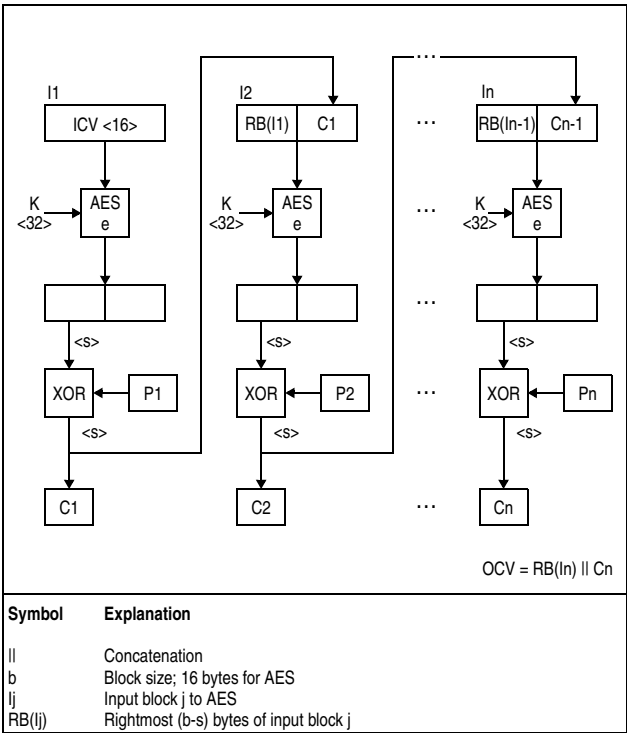


Figure 7-104. KMF-AES-256 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The s-byte cipher-

text segments (C1, C2, ..., Cn) in operand 2 are deciphered using the AES-encryption algorithm with the 256-bit cryptographic key and the 128-bit chaining value, where s is the length of cipher feedback in bytes.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. The s leftmost bytes of each output block are exclusive-ORed with the corresponding s-byte ciphertext segment to form a s-byte plaintext segment. The remaining (16 - s) bytes of each output block are ignored. Each ciphertext segment is concatenated to the right of the (16 - s) rightmost bytes of the previous input block to form the next input block.

The process is repeated with the successive input blocks until a plaintext segment is produced for every ciphertext segment, or until a CPU-determined number of plaintext segments have been produced.

The plaintext segments (P1, P2, ..., Pn) are stored in operand 1. The next input block is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-105.

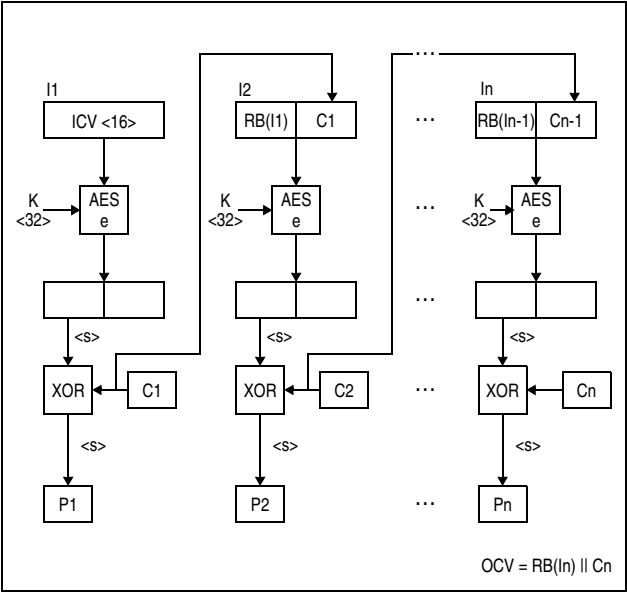


Figure 7-105. KMF-AES-256 Decipher Operation

Symbol	Explanation
	Concatenation
b	Block size; 16 bytes for AES
lj	Input block j to AES
RB(lj)	Rightmost (b-s) bytes of input block j

Figure 7-105. KMF-AES-256 Decipher Operation

Special Conditions

A specification exception is recognized and no other action is taken if any of the following occurs:

1. For DEA or TDEA functions, bits 32-39 of general register 0 specify a value that is zero or greater than 8.
2. For AES functions, bits 32-39 of general register 0 specify a value that is zero or greater than 16.
3. Bits 57-63 of general register 0 specify an unsigned or uninstalled function code.
4. The R₁ or R₂ field designates an odd-numbered register or general register 0.
5. The second operand length is not a multiple of the length of cipher feedback. This specification-exception condition does not apply to the query functions.

Resulting Condition Code:

- 0 Normal completion
- 1 Verification pattern mismatch
- 2 --
- 3 Partial completion

Program Exceptions:

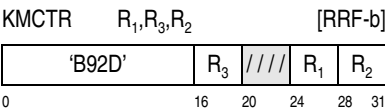
- Access (fetch, operand 2 and cryptographic key; store, operand 1; fetch and store, chaining value)
- Operation (if the message-security-assist extension 4 facility is not installed)
- Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Operation exception.
8.	Specification exception due to invalid function code or invalid register number.
9.	Specification exception due to invalid operand length.
10.	Condition code 0 due to second-operand length originally zero.
11.	Access exceptions for an access to the parameter block, first, or second operand.
12.	Condition code 1 due to verification pattern mismatch.
13.	Condition code 0 due to normal completion (second-operand length originally nonzero, but stepped to zero).
14.	Condition code 3 due to partial completion (second-operand length still nonzero).

Figure 7-106. Priority of Execution: KMF

Programming Note: See the programming notes for CIPHER MESSAGE WITH CHAINING.

CIPHER MESSAGE WITH COUNTER



A function specified by the function code in general register 0 is performed.

Bits 20-23 of the instruction are ignored.

Bit positions 57-63 of general register 0 contain the function code. Figure 7-107 shows the assigned function codes for CIPHER MESSAGE WITH COUNTER. All other function codes are unassigned. For cipher functions, bit 56 is the modifier bit which specifies whether an encryption or a decryption operation is to be performed. The modifier bit is ignored

for all other functions. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

The function codes for CIPHER MESSAGE WITH COUNTER are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	KMCTR-Query	16	—
1	KMCTR-DEA	8	8
2	KMCTR-TDEA-128	16	8
3	KMCTR-TDEA-192	24	8
9	KMCTR-Encrypted-DEA	32	8
10	KMCTR-Encrypted-TDEA-128	40	8
11	KMCTR-Encrypted-TDEA-192	48	8
18	KMCTR-AES-128	16	16
19	KMCTR-AES-192	24	16
20	KMCTR-AES-256	32	16
26	KMCTR-Encrypted-AES-128	48	16
27	KMCTR-Encrypted-AES-192	56	16
28	KMCTR-Encrypted-AES-256	64	16

Explanation:

— Not applicable

Figure 7-107. Function Codes for CIPHER MESSAGE WITH COUNTER

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers R₁, R₂, R₂ + 1, and R₃ are ignored for the query function.

For all other functions, the second operand is ciphered as specified by the function code using a

cryptographic key and counter values, and the result is placed in the first-operand location.

The R_1 field designates a general register and must designate an even-numbered register; otherwise, a specification exception is recognized.

The R_2 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The R_3 field designates a general register and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first, second, and third operands is specified by the contents of the R_1 , R_2 , and R_3 general registers, respectively. The number of bytes in the second-operand location is specified in general register $R_2 + 1$. The first operand and the third operand are the same length as the second operand.

As part of the operation, the addresses in general registers R_1 , R_2 , and R_3 are incremented by the number of bytes processed, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the addresses and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 , R_2 , and R_3 constitute the addresses of the first, second, and third operands, respectively, and the contents of bit positions 0-39 are ignored; bits 40-63 of the updated addresses replace the corresponding bits in general registers R_1 , R_2 , and R_3 , carries out of bit position 40 of the updated address are ignored, and the contents of bit positions 32-39 of general registers R_1 , R_2 , and R_3 are set to zeros. In the 31-bit addressing mode, the contents of bit positions 33-63 of general registers R_1 , R_2 , and R_3 constitute the addresses of the first, second, and third operands, respectively, and the contents of bit positions 0-32 are ignored; bits 33-63 of the updated addresses replace the corresponding bits in general registers R_1 , R_2 , and R_3 , carries out of bit position 33 of the updated address are ignored, and the content of bit position 32 of general registers R_1 , R_2 , and R_3 is set to zero. In the 64-bit addressing mode, the contents of bit positions 0-63 of general registers R_1 , R_2 , and R_3 constitute the addresses of the first, second, and third operands, respectively; bits 0-63 of the updated addresses

replace the contents of general registers R_1 , R_2 , and R_3 , and carries out of bit position 0 are ignored.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 32-63 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which specifies the number of bytes in the first, second, and third operands, and the contents of bit positions 0-31 are ignored; bits 32-63 of the updated value replace the corresponding bits in general register $R_2 + 1$. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register $R_2 + 1$ form a 64-bit unsigned binary integer which specifies the number of bytes in the first, second, and third operands; and the updated value replaces the contents of general register $R_2 + 1$.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , R_2 , $R_2 + 1$, and R_3 always remain unchanged.

Figure 7-108 on page 7-88 shows the contents of the general registers just described.

In the access-register mode, access registers 1, R_1 , R_2 , and R_3 specify the address spaces containing the parameter block, first, second, and third operands, respectively.

The result is obtained as if processing starts at the left end of the first, second, and third operands and proceeds to the right, block by block. The operation is ended when the number of bytes in the second operand as specified in general register $R_2 + 1$ have been processed and placed at the first-operand location (called normal completion) or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually non-zero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless recurrence of this no-progress case.

The results in the first-operand location are unpredictable if any of the following situations occurs:

1. The cryptographic-key field or the encrypted cryptographic-key field overlaps any portion of the first operand.

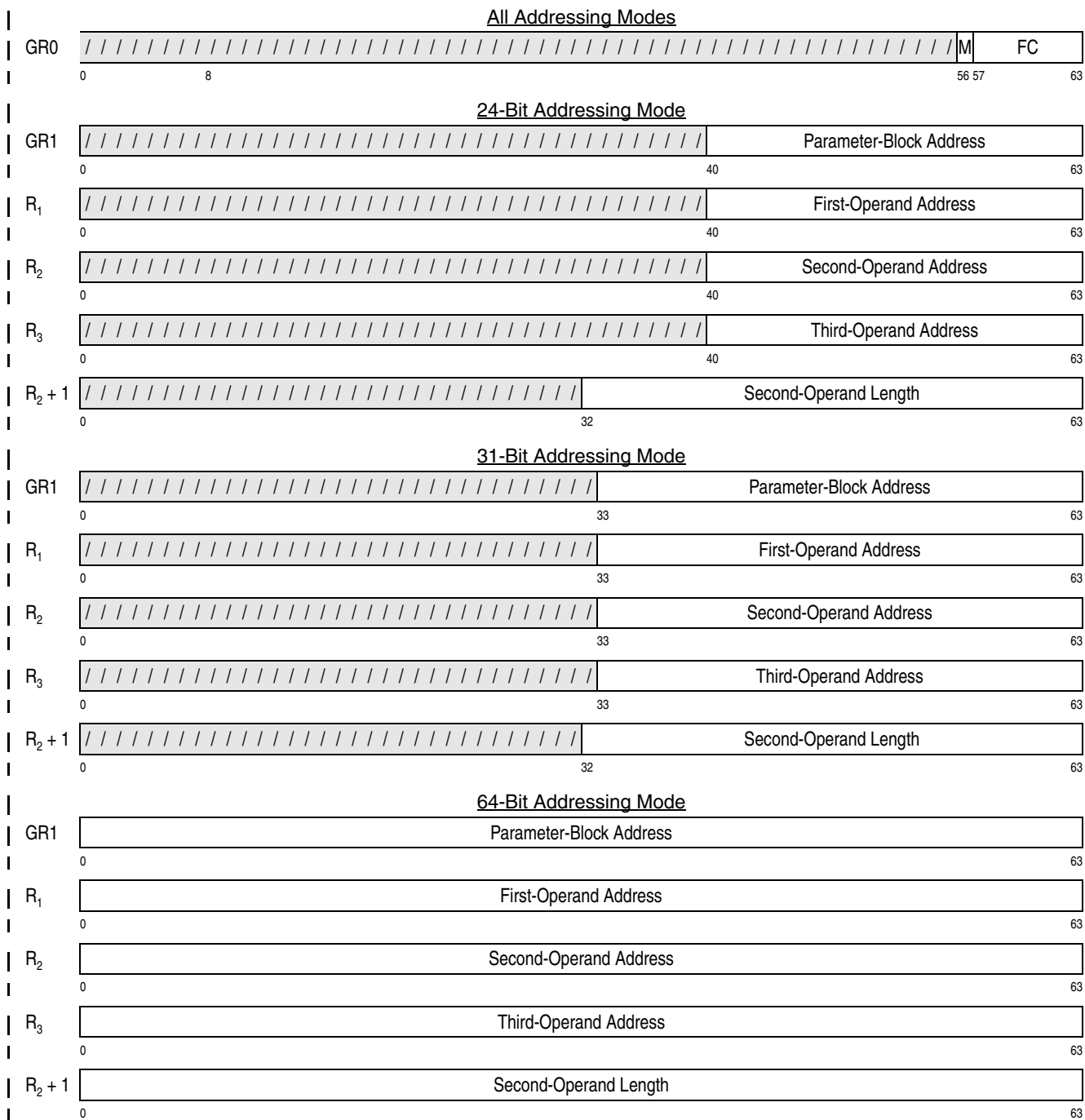


Figure 7-108. General Register Assignment for KMCTR

2. The first and second operands overlap destructively.
3. The first and third operands overlap destructively.

Operands are said to overlap destructively when the first-operand location would be used as a source after data would have been moved into it, assuming

processing to be performed from left to right and one byte at a time.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in $R_2 + 1$ is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in $R_2 + 1$ is nonzero.

When a storage-alteration PER event is recognized, fewer than 4K additional bytes are stored into the first-operand locations before the event is reported.

When the second-operand length is initially zero, the parameter block, first, second, and third operands are not accessed, general registers R_1 , R_2 , $R_2 + 1$, and R_3 are not changed, and condition code 0 is set.

When the contents of any two or all three of the R_1 , R_2 , and R_3 fields are the same, the contents of the designated registers are incremented only by the number of bytes processed, not by a multiple of the number of bytes processed.

As observed by other CPUs and channel programs, references to the parameter block and storage operands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

In certain unusual situations, instruction execution may complete by setting condition code 3 without updating the registers to reflect the last unit of the first, second, and third operands processed. The size of the unit processed in this case depends on the situation and the model, but is limited such that the portion of the first and second operands which have been processed and not reported do not overlap in storage; and the portion of the first and third operands which have been processed and not reported do not overlap in storage. In all cases, change bits are set and PER storage-alteration events are reported, when applicable, for all first-operand locations processed.

Access exceptions may be reported for a larger portion of an operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of an operand nor for locations more than 4K bytes beyond the current location being processed.

Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the CIPHER MESSAGE WITH COUNTER functions. For data-encryption-algorithm (DEA) functions, the DEA-key-parity bit in each byte of the DEA key is ignored, and the operation proceeds normally, regardless of the DEA-key parity of the key.

Further description of the data-encryption algorithm may be found in *Data Encryption Algorithm*, ANSI-X3.92.1981, American National Standard for Information Systems. Further description of the AES standard may be found in *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, dated November 26, 2001.

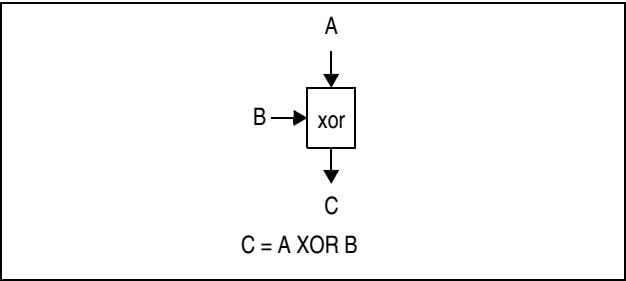


Figure 7-109. Symbol For Bit-Wise Exclusive OR

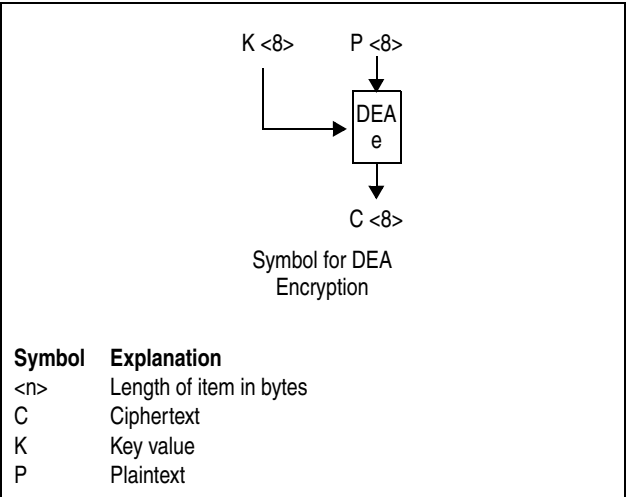


Figure 7-110. Symbols for DEA Encryption

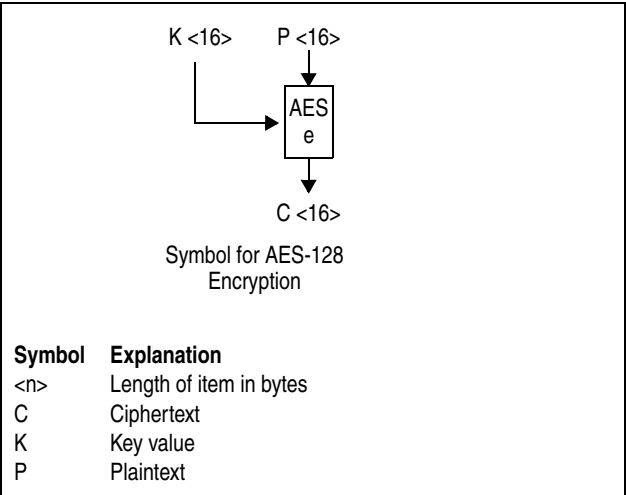


Figure 7-111. Symbols for AES-128 Encryption

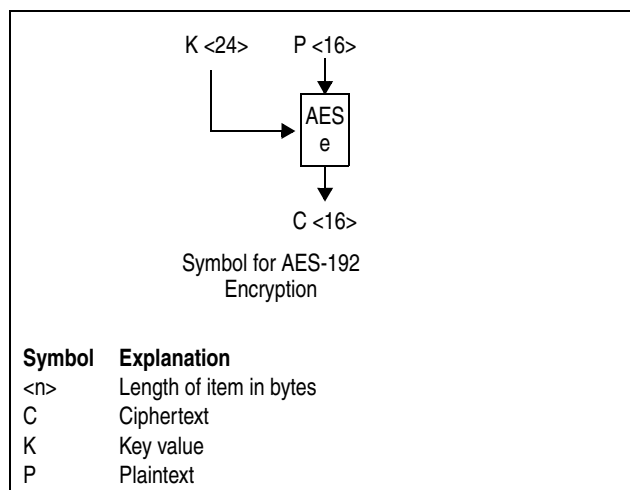


Figure 7-112. Symbols for AES-192 Encryption

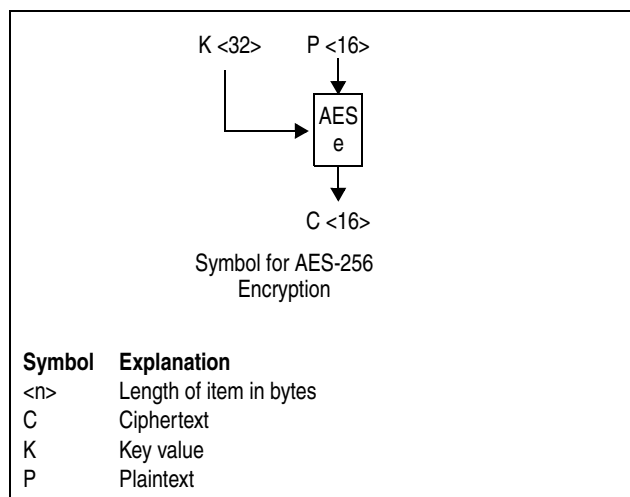


Figure 7-113. Symbols for AES-256 Encryption

KMCTR-Query (Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-108 on page 7-88.

The parameter block used for the function has the following format:

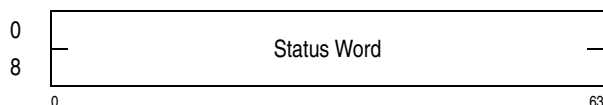


Figure 7-114. Parameter Block for KMCTR-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the CIPHER MESSAGE WITH COUNTER instruction. When a bit is one, the

corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KMCTR-Query function completes; condition code 3 is not applicable to this function.

KMCTR-DEA (Function Code 1)

KMCTR-Encrypted-DEA (Function Code 9)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-108 on page 7-88.

The parameter block used for the KMCTR-DEA function has the following format:

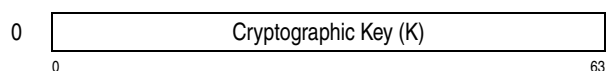


Figure 7-115. Parameter Block for KM-DEA

For the KMCTR-DEA function, the cryptographic key is in byte offsets 0-7 of the parameter block.

The parameter block used for the KMCTR-Encrypted-DEA function has the following format:

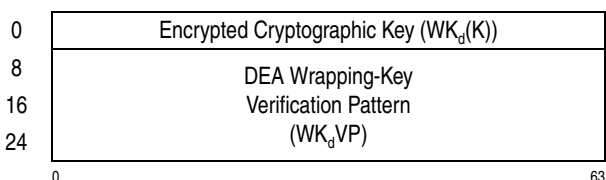


Figure 7-116. Parameter Block for KMCTR-Encrypted-DEA

For the KMCTR-Encrypted-DEA function, the contents of byte offsets 8-31 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-7 of the parameter block are deciphered using the DEA wrapping key to obtain the 64-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P_1, P_2, \dots, P_n) in operand 2 are enciphered

using the DEA algorithm with the 64-bit cryptographic key and using 8-byte counter-value blocks (R1, R2, ..., Rn) in operand 3. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in Figure 7-117.

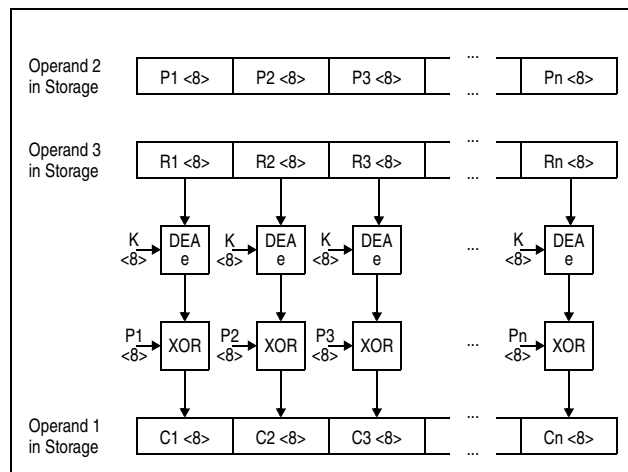


Figure 7-117. KMCTR DEA Encipher Operation Using 64-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the DEA algorithm with the 64-bit cryptographic key and using 8-byte counter-value blocks (R1, R2, ..., Rn) in operand 3. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in Figure 7-118.

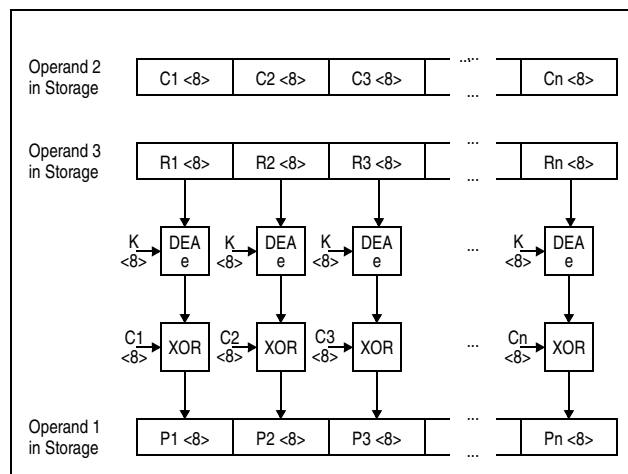


Figure 7-118. KMCTR DEA Decipher Operation Using 64-Bit Key

KMCTR-TDEA-128 (Function Code 2)

KMCTR-Encrypted-TDEA-128 (Function Code 10)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-108 on page 7-88.

The parameter block used for the KMCTR-TDEA-128 function has the following format:

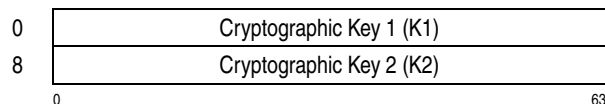


Figure 7-119. Parameter Block for KMCTR-TDEA-128

For the KMCTR-TDEA-128 function, the cryptographic key is in byte offsets 0-15 of the parameter block.

The parameter block used for the KMCTR-Encrypted-TDEA-128 function has the following format:

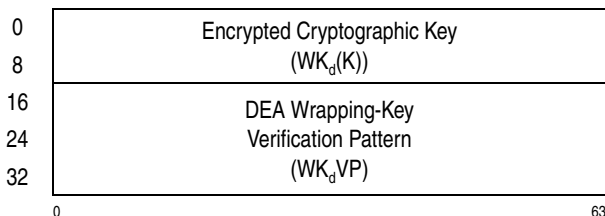


Figure 7-120. Parameter Block for KMCTR Encrypted-TDEA-128

For the KMCTR-Encrypted-TDEA-128 function, the contents of byte offsets 16-39 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-15 of the parameter block are deciphered using the DEA wrapping key to obtain the 128-bit cryptographic key, $K = K1 \parallel K2$, where K1 is the leftmost 64 bits of K and K2 is the rightmost 64 bits of K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered

using the TDEA (triple DEA) algorithm with the two 64-bit cryptographic keys and using 8-byte counter-value blocks (R1, R2, ..., Rn) in operand 3. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in Figure 7-121.

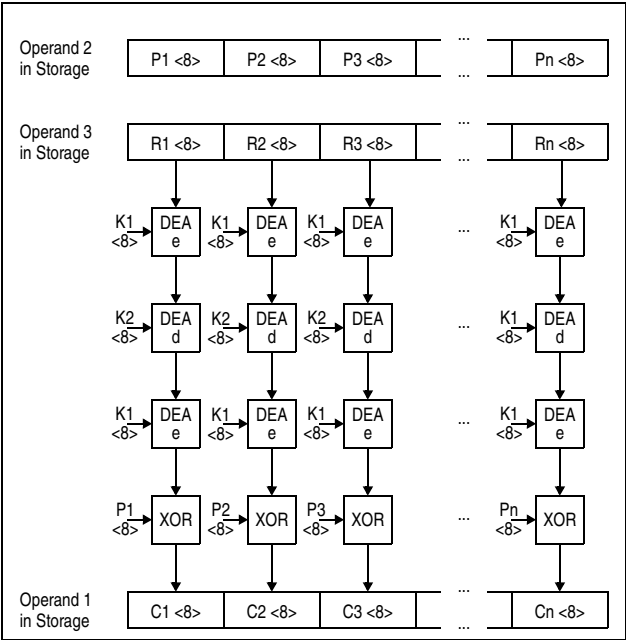


Figure 7-121. KMCTR TDEA Encipher Operation Using 128-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the TDEA algorithm with the two 64-bit cryptographic keys and using 8-byte counter-value blocks (R1, R2, ..., Rn) in operand 3. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The

plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in Figure 7-122.

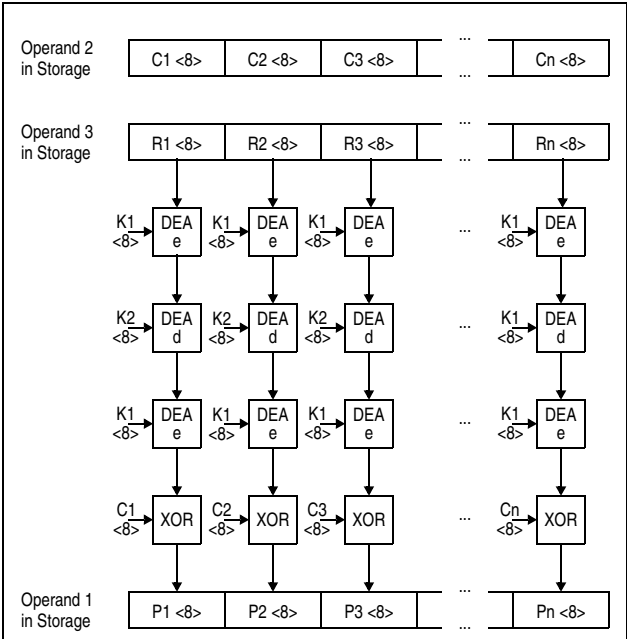


Figure 7-122. KMCTR TDEA DEcipher Operation Using 128-Bit Key

KMCTR-TDEA-192 (Function Code 3)

KMCTR-Encrypted-TDEA-192 (Function Code 11)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-108 on page 7-88.

The parameter block used for the KMCTR-TEDA-192 function has the following format:

0	Cryptographic Key 1 (K1)
8	Cryptographic Key 2 (K2)
16	Cryptographic Key 3 (K3)
0	63

Figure 7-123. Parameter Block for KMCTR-TDEA-192

For the KMCTR-TDEA-192 function, the cryptographic key is in byte offsets 0-23 of the parameter block.

The parameter block used for the KMCTR-Encrypted-TDEA-192 function has the following format:

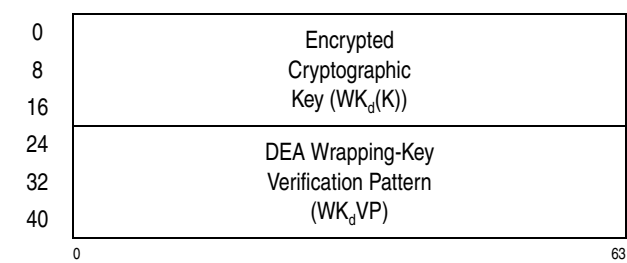


Figure 7-124. Parameter Block for KMCTR-Encrypted-TDEA-192

For the KMCTR-Encrypted-TDEA-192 function, the contents of byte offsets 24-47 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-23 of the parameter block are deciphered using the DEA wrapping key to obtain the 192-bit cryptographic key, $K = K1 \parallel K2 \parallel K3$, where $K1$ is the leftmost 64 bits of K , $K2$ is the middle 64 bits of K , and $K3$ is the rightmost 64 bits of K . (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks ($P1, P2, \dots, Pn$) in operand 2 are enciphered using the TDEA algorithm with the three 64-bit cryptographic keys and using 8-byte counter-value blocks ($R1, R2, \dots, Rn$) in operand 3. Each ciphertext block is independently enciphered; that is, the encipher operation is performed without chaining.

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks ($C1, C2, \dots, Cn$) in operand 1 are deciphered using the TDEA algorithm with the three 64-bit cryptographic keys and using 8-byte counter-value blocks ($R1, R2, \dots, Rn$) in operand 3. Each plaintext block is independently deciphered; that is, the decipher operation is performed without chaining.

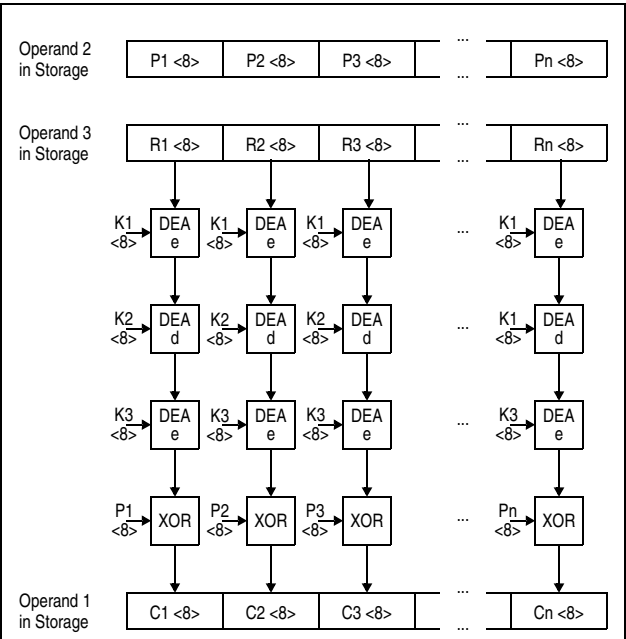


Figure 7-125. KMCTR TDEA Encipher Operation Using 192-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks ($C1, C2, \dots, Cn$) in operand 1 are deciphered using the TDEA algorithm with the three 64-bit cryptographic keys and using 8-byte counter-value blocks ($R1, R2, \dots, Rn$) in operand 3. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining.

The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in Figure 7-126.

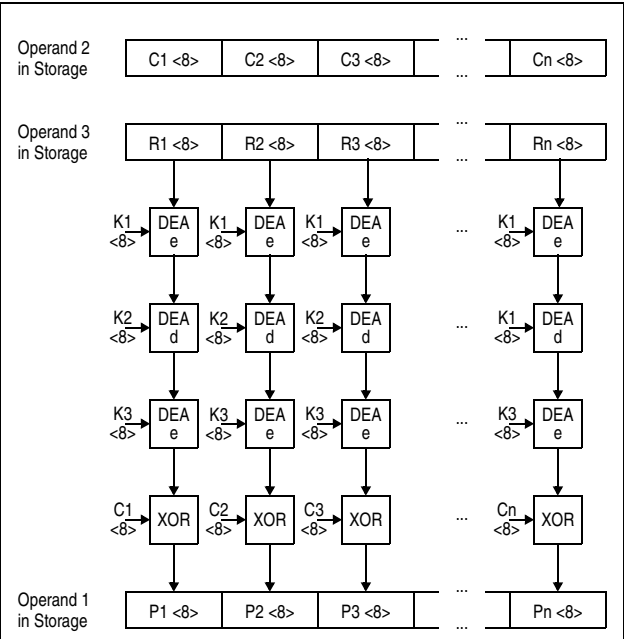


Figure 7-126. KMCTR TDEA Decipher Operation Using 192-Bit Key

KMCTR-AES-128 (Function Code 18)

KMCTR-Encrypted-AES-128 (Function Code 26)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-108 on page 7-88.

The parameter block used for the KMCTR-AES-128 function has the following format:

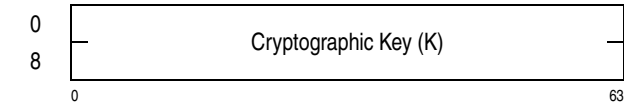


Figure 7-127. Parameter Block for KMCTR-AES-128

For the KMCTR-AES-128 function, the cryptographic key is in byte offsets 0-15 of the parameter block.

The parameter block used for the KMCTR-encrypted-AES-128 function has the following format:

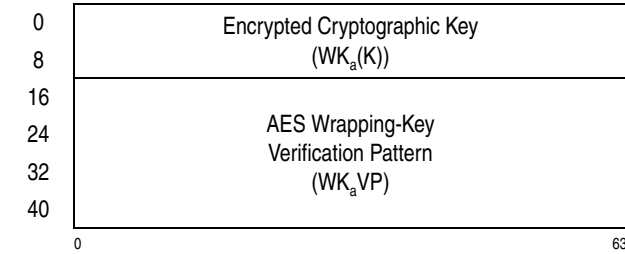


Figure 7-128. Parameter Block for KMCTR-Encrypted-AES-128

For the KM-Encrypted-AES-128 function, the contents of byte offsets 16-47 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-15 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES algorithm with the 128-bit cryptographic key and using 16-byte counter-value blocks (R1, R2, ..., Rn) in operand 3. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The

ciphertext blocks (C1, C2, ..., Cn) are stored in operand 2. The operation is shown in Figure 7-129.

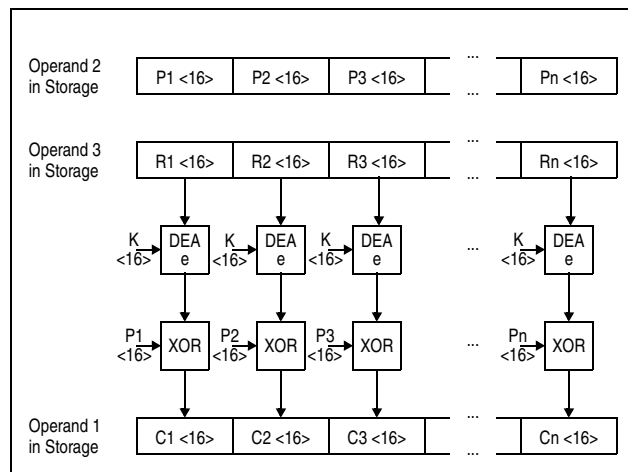


Figure 7-129. KMCTR AES Encipher Operation Using 128-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES algorithm with the 128-bit cryptographic key and using 16-byte counter-value blocks (R1, R2, ..., Rn) in operand 3. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown in Figure 7-130.

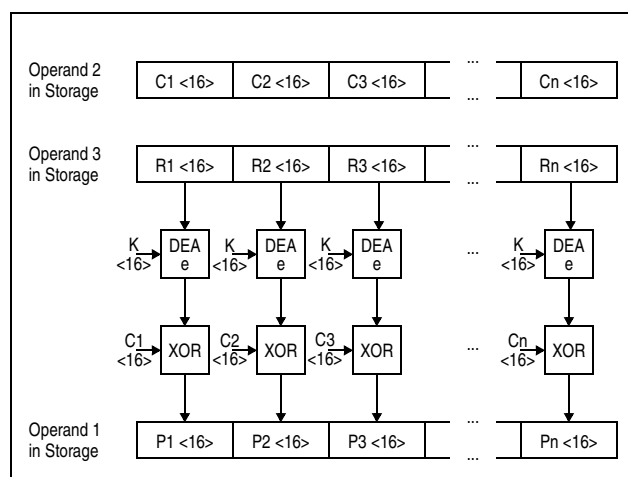


Figure 7-130. KMCTR AES Decipher Operation Using 128-Bit Key

KMCTR-AES-192 (Function Code 19)

KMCTR-Encrypted-AES-192 (Function Code 27)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-108 on page 7-88.

The parameter block used for the KMCTR-AES-192 function has the following format:

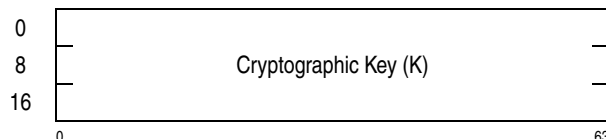


Figure 7-131. Parameter Block for KMCTR-AES-192

For the KMCTR-AES-192 function, the cryptographic key is in byte offsets 0-23 of the parameter block.

The parameter block used for the KMCTR-Encrypted-AES-192 function has the following format:

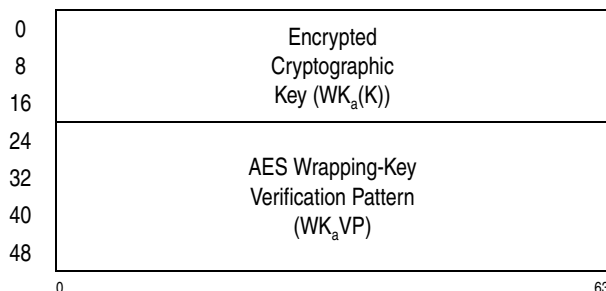


Figure 7-132. Parameter Block for KMCTR-Encrypted-AES-192

For the KMCTR-Encrypted-AES-192 function, the contents of byte offsets 24-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-23 of the parameter block are deciphered using the AES wrapping key to obtain the 192-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plain-

text blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES algorithm with the 192-bit cryptographic key and using 16-byte counter-value blocks (R1, R2, ..., Rn) in operand 3. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The operation is shown in Figure 7-133.

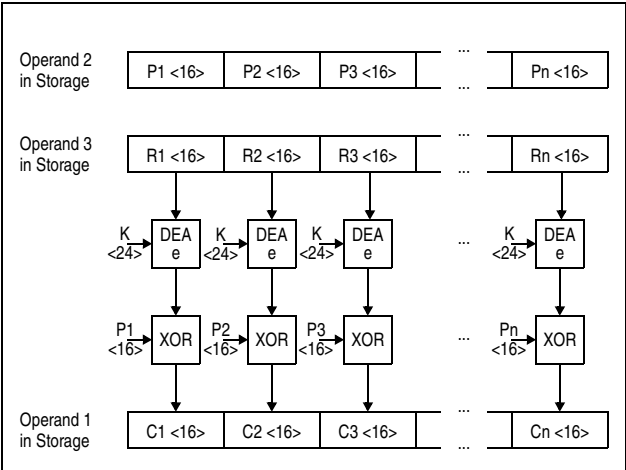


Figure 7-133. KMCTR AES Encipher Operation Using 192-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES algorithm with the 192-bit cryptographic key and using 16-byte counter-value blocks (R1, R2, ..., Rn) in operand 3. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The

The parameter block used for the KMCTR-Encrypted-AES-256 function has the following for-

plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The operation is shown Figure 7-134.

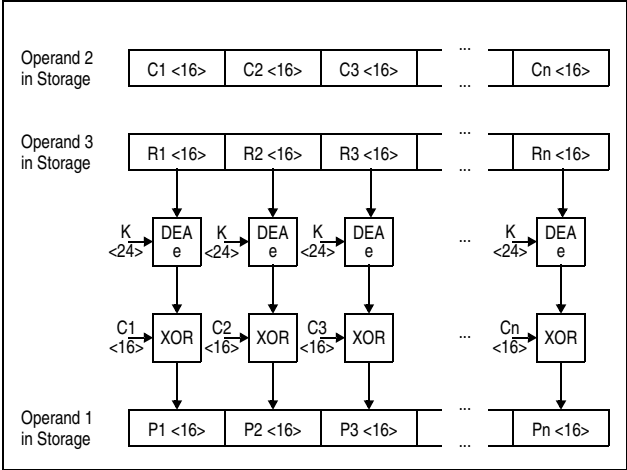


Figure 7-134. KMCTR AES Decipher Operation Using 192-Bit Key

KMCTR-AES-256 (Function Code 20)

KMCTR-Encrypted-AES-256 (Function Code 28)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-108 on page 7-88.

The parameter block used for the KMCTR-AES-256 function has the following format:

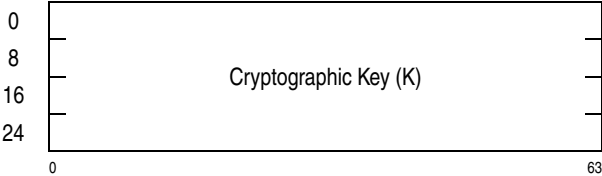


Figure 7-135. Parameter Block for KMCTR-AES-256

For the KMCTR-AES-256 function, the cryptographic key is in byte offsets 0-31 of the parameter block.

mat:

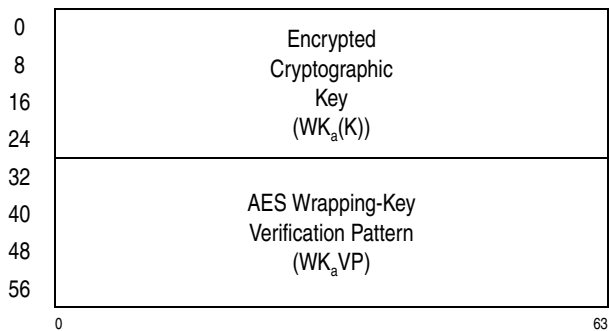


Figure 7-136. Parameter Block for KMCTR-Encrypted-AES-256

For the KMCTR-Encrypted-AES-256 function, the contents of byte offsets 32-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-31 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P_1, P_2, \dots, P_n) in operand 2 are enciphered using the AES algorithm with the 256-bit cryptographic key and using 16-byte counter-value blocks (R_1, R_2, \dots, R_n) in operand 3. Each plaintext block is independently enciphered; that is, the encipher operation is performed without chaining. The

ciphertext blocks (C_1, C_2, \dots, C_n) are stored in operand 1. The operation is shown in Figure 7-137.

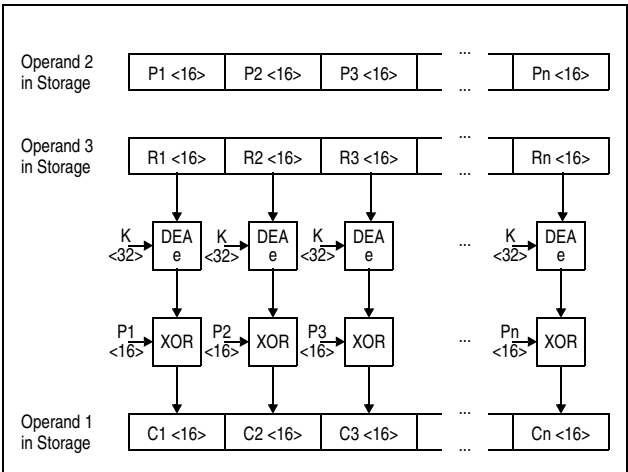


Figure 7-137. KMCTR AES Encipher Operation Using 256-Bit Key

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C_1, C_2, \dots, C_n) in operand 2 are deciphered using the AES algorithm with the 256-bit cryptographic key and using 16-byte counter-value blocks (R_1, R_2, \dots, R_n) in operand 3. Each ciphertext block is independently deciphered; that is, the decipher operation is performed without chaining. The plaintext blocks (P_1, P_2, \dots, P_n) are stored in operand 1. The operation is shown in Figure 7-138

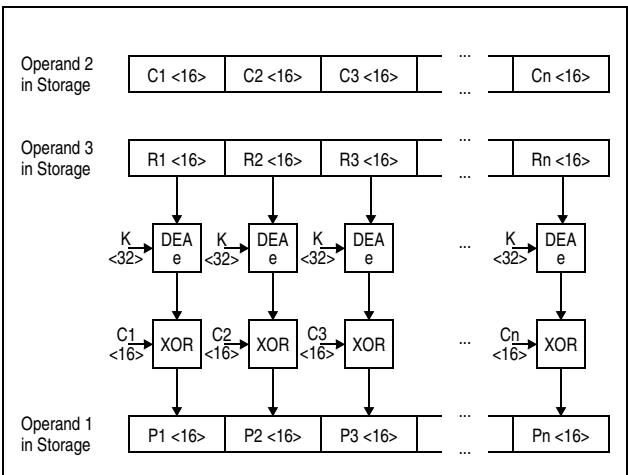


Figure 7-138. KMCTR AES Decipher Operation Using 256-Bit Key

Special Conditions for KMCTR

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bits 57-63 of general register 0 specify an unsigned or uninstalled function code.
2. The R_1 , R_2 , or R_3 field designates an odd-numbered register or general register 0.
3. The second operand length is not a multiple of the data block size of the designated function (see Figure 7-107 on page 7-86 to determine the data block sizes for CIPHER MESSAGE WITH COUNTER functions). This specification-exception condition does not apply to the query functions.

Resulting Condition Code:

- 0 Normal completion
- 1 Verification pattern mismatch
- 2 --
- 3 Partial completion

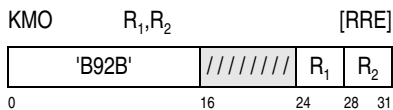
Program Exceptions:

- Access (fetch, operand 2, operand 3, and cryptographic key; store, operand 1)
- Operation (if the message-security-assist extension 4 facility is not installed)
- Specification

- 1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.
- 7.A Access exceptions for second instruction halfword.
- 7.B Operation exception.
8. Specification exception due to invalid function code or invalid register number.
9. Specification exception due to invalid operand length.
10. Condition code 0 due to second-operand length originally zero.
11. Access exceptions for an access to the parameter block, first, second, or third operand.
12. Condition code 1 due to verification pattern mismatch.
13. Condition code 0 due to normal completion (second-operand length originally nonzero, but stepped to zero).
14. Condition code 3 due to partial completion (second-operand length still nonzero).

Figure 7-139. Priority of Execution: KMCTR

CIPHER MESSAGE WITH OFB



A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction are ignored.

Bit positions 57-63 of general register 0 contain the function code. Figure 7-140 shows the assigned function codes for CIPHER MESSAGE WITH OFB (output feedback). All other function codes are unassigned. For cipher functions, bit 56 is the modifier bit which specifies whether an encryption or a decryption operation is to be performed. The modifier bit is ignored for all other functions. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

The function codes for CIPHER MESSAGE WITH OFB are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	KMO-Query	16	—
1	KMO-DEA	16	8
2	KMO-TDEA-128	24	8
3	KMO-TDEA-192	32	8
9	KMO-Encrypted-DEA	40	8
10	KMO-Encrypted-TDEA-128	48	8
11	KMO-Encrypted-TDEA-192	56	8
18	KMO-AES-128	32	16
19	KMO-AES-192	40	16
20	KMO-AES-256	48	16
26	KMO-Encrypted-AES-128	64	16
27	KMO-Encrypted-AES-192	72	16
28	KMO-Encrypted-AES-256	80	16
Explanation:			
— Not applicable			

Figure 7-140. Function Codes for CIPHER MESSAGE WITH OFB

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers R_1 , R_2 , and $R_2 + 1$ are ignored for the query function.

For all other functions, the second operand is ciphered as specified by the function code using a cryptographic key and an initial chaining value in the parameter block, and the result is placed in the first-

operand location. The chaining value is updated as part of the operation.

The R_1 field designates a general register and must designate an even-numbered register; otherwise, a specification exception is recognized.

The R_2 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first and second operands is specified by the contents of the R_1 and R_2 general registers, respectively. The number of bytes in the second-operand location is specified in general register $R_2 + 1$. The first operand is the same length as the second operand.

As part of the operation, the addresses in general registers R_1 and R_2 are incremented by the number of bytes processed, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the addresses and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively, and the contents of bit positions 0-39 are ignored; bits 40-63 of the updated addresses replace the corresponding bits in general registers R_1 and R_2 , carries out of bit position 40 of the updated address are ignored, and the contents of bit positions 32-39 of general registers R_1 and R_2 are set to zeros. In the 31-bit addressing mode, the contents of bit positions 33-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively, and the contents of bit positions 0-32 are ignored; bits 33-63 of the updated addresses replace the corresponding bits in general registers R_1 and R_2 , carries out of bit position 33 of the updated address are ignored, and the content of bit position 32 of general registers R_1 and R_2 is set to zero. In the 64-bit addressing mode, the contents of bit positions 0-63 of general registers R_1 and R_2 constitute the addresses of the first and second operands, respectively; bits 0-63 of the updated addresses replace the contents of general registers R_1 and R_2 , and carries out of bit position 0 are ignored.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 32-63 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which

specifies the number of bytes in the first and second operands, and the contents of bit positions 0-31 are ignored; bits 32-63 of the updated value replace the corresponding bits in general register $R_2 + 1$. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register $R_2 + 1$ form a 64-bit unsigned binary integer which specifies the number of bytes in the first and second operands; and the updated value replaces the contents of general register $R_2 + 1$.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , R_2 , and $R_2 + 1$, always remain unchanged.

Figure 7-141 on page 7-100 shows the contents of the general registers just described.

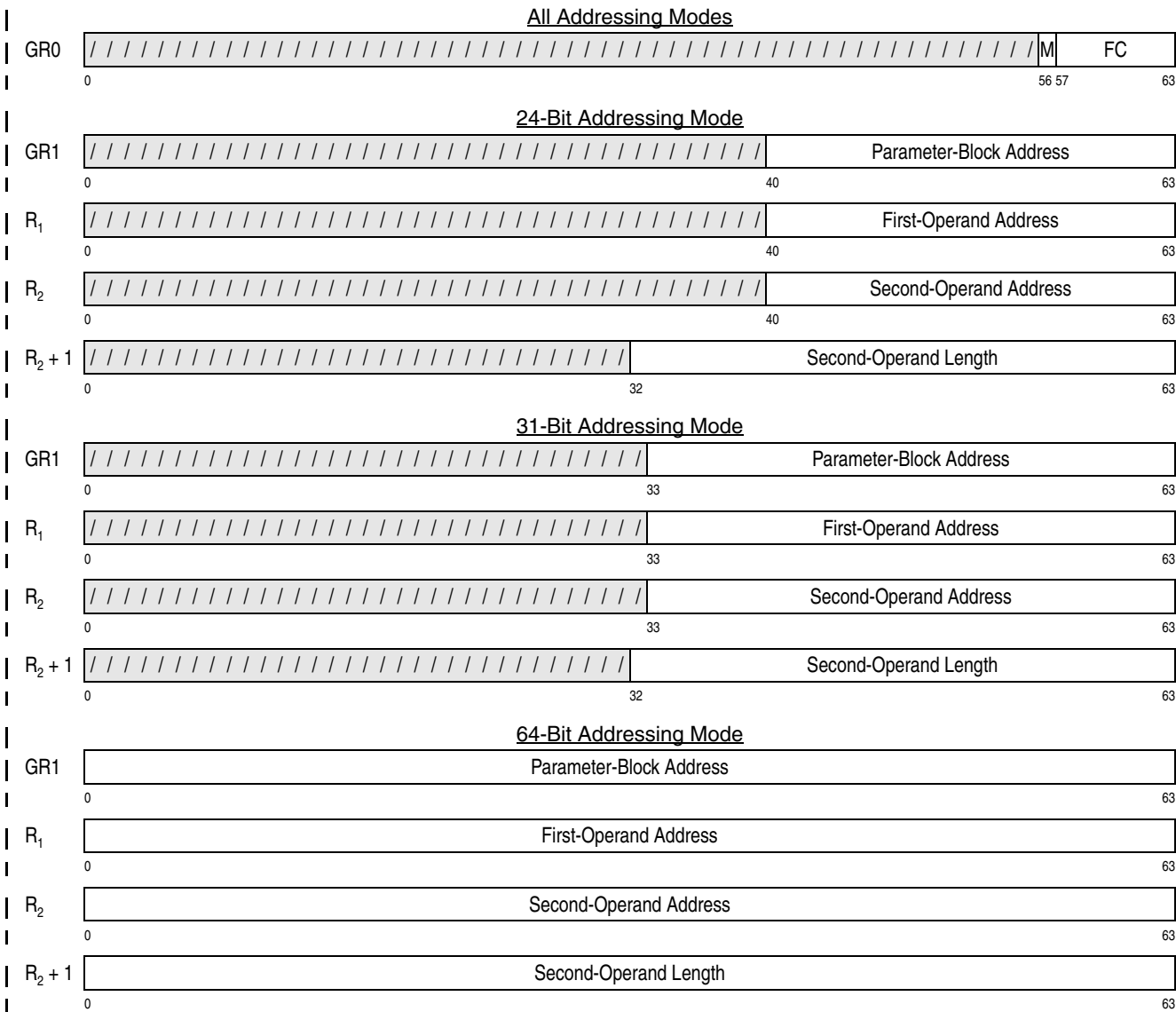


Figure 7-141. General Register Assignment for KMO

In the access-register mode, access registers 1, R_1 , and R_2 specify the address spaces containing the parameter block, first, and second operands, respectively.

The result is obtained as if processing starts at the left end of both the first and second operands and proceeds to the right, block by block. The operation is ended when the number of bytes in the second operand and as specified in general register $R_2 + 1$ have been

processed and placed at the first-operand location (called normal completion) or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually non-zero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless reoccurrence of this no-progress case.

The results in the first-operand location and the chaining-value field are unpredictable if any of the following situations occur:

1. The cryptographic-key field or the encrypted cryptographic-key field overlaps any portion of the first operand.
2. The chaining-value field overlaps any portion of the first operand or the second operand.
3. The first and second operands overlap destructively. Operands are said to overlap destructively when the first-operand location would be used as a source after data would have been moved into it, assuming processing to be performed from left to right and one byte at a time.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in $R_2 + 1$ is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in $R_2 + 1$ is nonzero.

When a storage-alteration PER event is recognized, fewer than 4K additional bytes are stored into the first-operand locations before the event is reported.

When the second-operand length is initially zero, the parameter block, first, and second operands are not accessed, general registers R_1 , R_2 , and $R_2 + 1$ are not changed, and condition code 0 is set.

When the contents of the R_1 and R_2 fields are the same, the contents of the designated registers are incremented only by the number of bytes processed, not by twice the number of bytes processed.

As observed by other CPUs and channel programs, references to the parameter block and storage oper-

ands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

In certain unusual situations, instruction execution may complete by setting condition code 3 without updating the registers and chaining value to reflect the last unit of the first and second operands processed. The size of the unit processed in this case depends on the situation and the model, but is limited such that the portion of the first and second operands which have been processed and not reported do not overlap in storage. In all cases, change bits are set and PER storage-alteration events are reported, when applicable, for all first-operand locations processed.

Access exceptions may be reported for a larger portion of an operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of an operand nor for locations more than 4K bytes beyond the current location being processed.

Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the CIPHER MESSAGE WITH OFB functions. For data-encryption-algorithm (DEA) functions, the DEA-key-parity bit in each byte of the DEA key is ignored, and the operation proceeds normally, regardless of the DEA-key parity of the key.

Further description of the data-encryption algorithm may be found in *Data Encryption Algorithm*, ANSI-X3.92.1981, American National Standard for Information Systems. Further description of the AES standard may be found in *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, dated November 26, 2001.

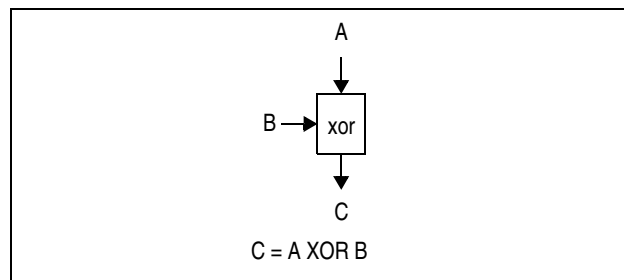


Figure 7-142. Symbol For Bit-Wise Exclusive OR

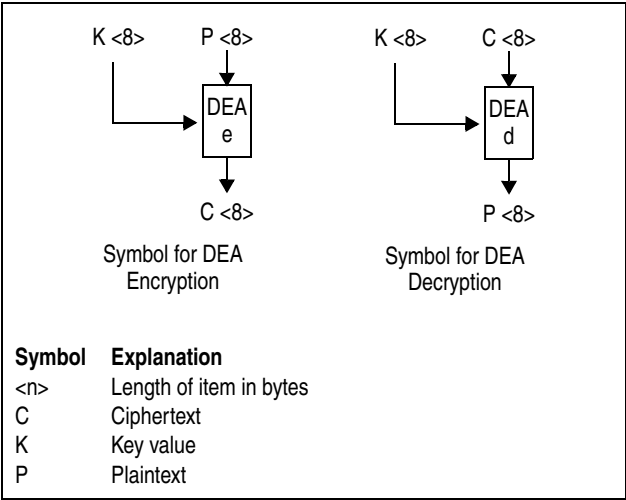


Figure 7-143. Symbols for DEA Encryption and Decryption

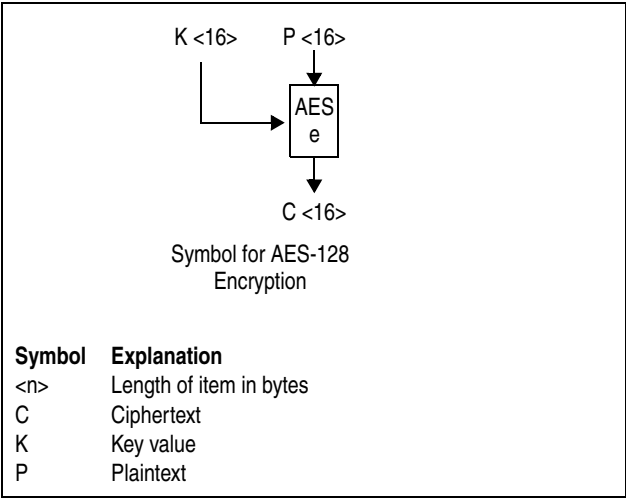


Figure 7-144. Symbols for AES-128 Encryption

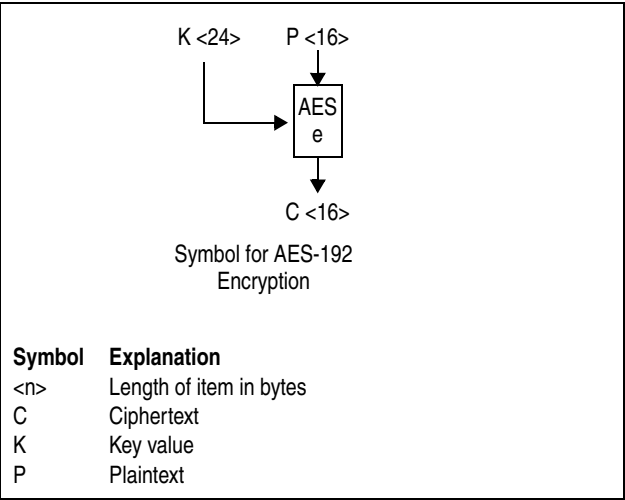


Figure 7-145. Symbols for AES-192 Encryption

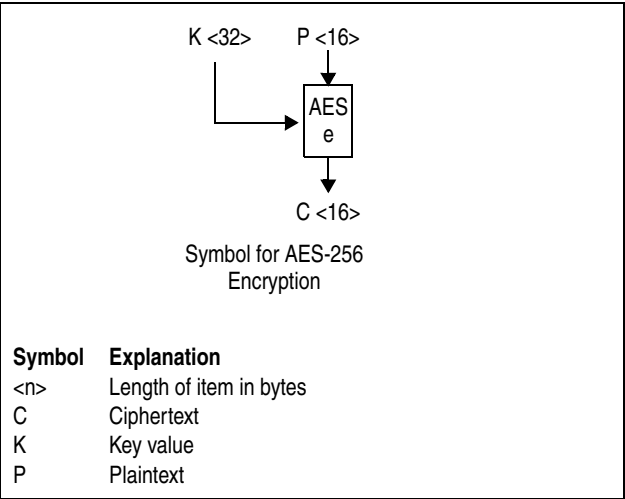


Figure 7-146. Symbols for AES-256 Encryption

KMO-Query (Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-141 on page 7-100.

The parameter block used for the function has the following format:

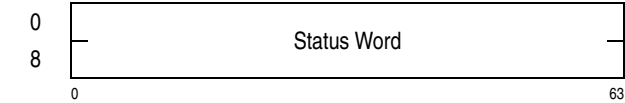


Figure 7-147. Parameter Block for KMO-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the CIPHER MESSAGE WITH OFB instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KMO-Query function completes; condition code 3 is not applicable to this function.

KMO-DEA (Function Code 1)

KMO-Encrypted-DEA (Function Code 9)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-141 on page 7-100.

The parameter block used for the KMO-DEA function has the following format:

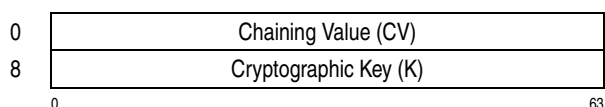


Figure 7-148. Parameter Block for KMO-DEA

For the KMO-DEA function, the chaining value is in byte offset 0-7 of the parameter block and the cryptographic key is in byte offsets 8-15 of the parameter block.

The parameter block used for the KMO-Encrypted - DEA function has the following format:

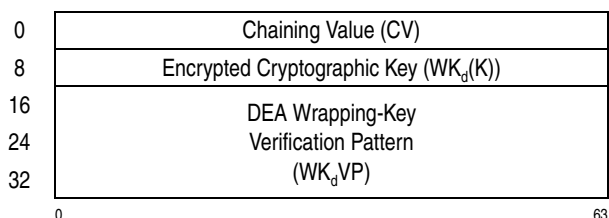


Figure 7-149. Parameter Block for KMO-Encrypted-DEA

For the KMO-Encrypted-DEA function, the contents of byte offsets 16-39 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the chaining value, and the contents of byte offsets 8-15 of the parameter block are deciphered

using the DEA wrapping key to obtain the 64-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P_1, P_2, \dots, P_n) in operand 2 are enciphered using the DEA-encryption algorithm with the 64-bit cryptographic key and the 64-bit chaining value.

The first input block to the DEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding plaintext block to form a ciphertext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a ciphertext block is produced for every plaintext block, or until a CPU-determined number of ciphertext blocks have been produced.

The ciphertext blocks (C_1, C_2, \dots, C_n) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-150.

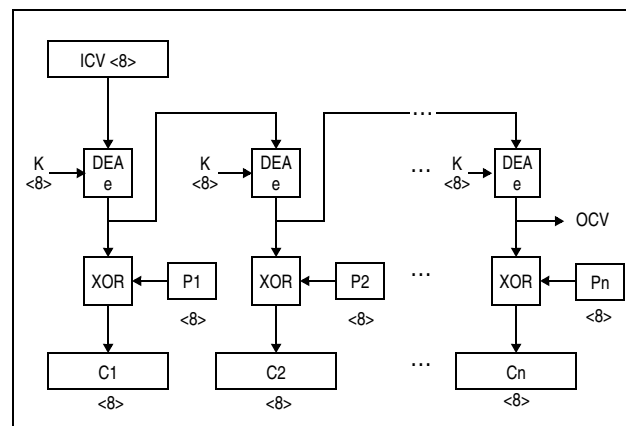


Figure 7-150. KMO-DEA Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C_1, C_2, \dots, C_n) in operand 2 are deciphered using the DEA-encryption algorithm with the 64-bit cryptographic key and the 64-bit chaining value.

The first input block to the DEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding ciphertext block to form a plaintext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a plaintext block is produced for every ciphertext block, or until a CPU-determined number of plaintext blocks have been produced.

The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-151.

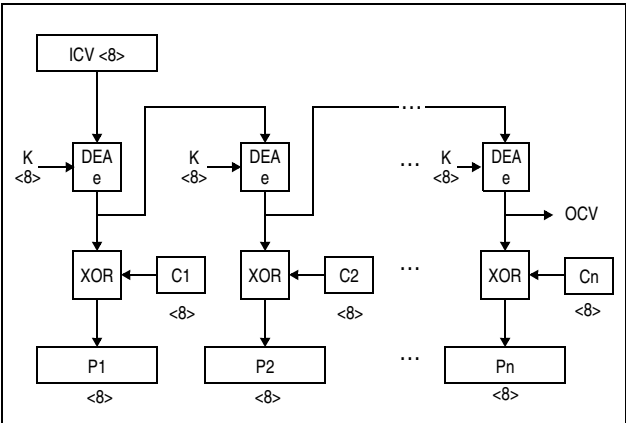


Figure 7-151. KMO-DEA Decipher Operation

KMO-TDEA-128 (Function Code 2)

KMO-Encrypted-TDEA-128 (Function Code 10)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-141 on page 7-100.

The parameter block used for the KMO-TDEA-128 function has the following format:

0	Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)
0	63

Figure 7-152. Parameter Block for KMO-TDEA-128

For the KMO-TDEA-128 function, the chaining value is in byte offset 0-7 of the parameter block and the cryptographic key is in byte offsets 8-23 of the parameter block.

The parameter block used for the KMO-Encrypted-TDEA-128 function has the following format:

0	Chaining Value (CV)
8	Encrypted Cryptographic Key (WK _e (K))
16	
24	DEA Wrapping-Key Verification Pattern (WK _d VP)
32	
40	
0	63

Figure 7-153. Parameter Block for KMO-TDEA-128

For the KMO-Encrypted-TDEA-128 function, the contents of byte offsets 24-47 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the chaining value, and the contents of byte offsets 8-23 of the parameter block are deciphered using the DEA wrapping key to obtain the 128-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the TDEA-encryption algorithm with the 128-bit cryptographic key and the 64-bit chaining value.

The first input block to the TDEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding plaintext block to form a ciphertext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a ciphertext block is produced for every plaintext block, or until a CPU-determined number of ciphertext blocks have been produced.

The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The next input block, called the output

chaining value (OCV), is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-154.

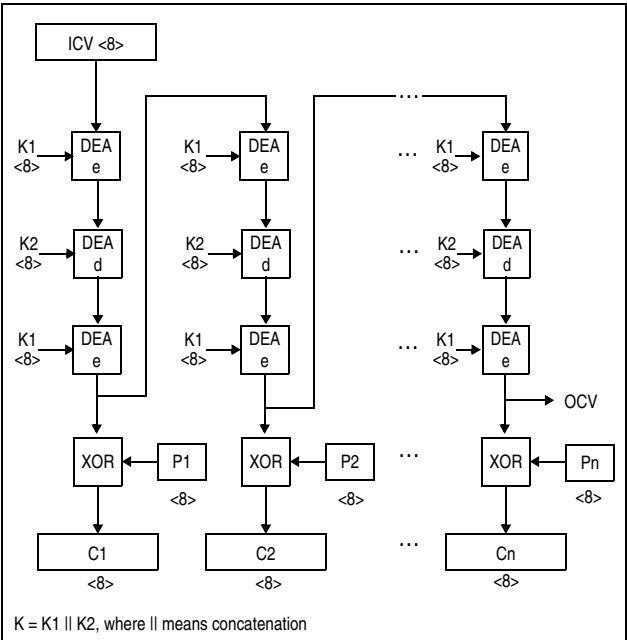


Figure 7-154. KMO-TDEA-128 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the DEA-encryption algorithm with the 128-bit cryptographic key and the 64-bit chaining value.

The first input block to the TDEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding ciphertext block to form a plaintext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a plaintext block is produced for every ciphertext block, or until a CPU-determined number of plaintext blocks have been produced.

The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-

value field of the parameter block. The operation is shown in Figure 7-155.

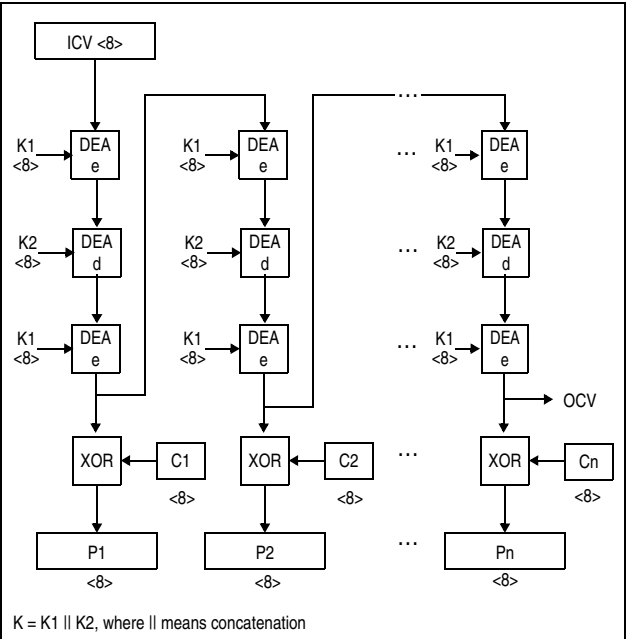


Figure 7-155. KMO-TDEA-128 Decipher Operation

KMO-TDEA-192 (Function Code 3)

KMO-Encrypted-TDEA-192 (Function Code 11)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-141 on page 7-100.

The parameter block used for the KMO-TDEA-192 function has the following format:

0	Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)
24	Cryptographic Key 3 (K3)
63	

Figure 7-156. Parameter Block for KMO-TDEA-192

For the KMF-TDEA-192 function, the chaining value is in byte offset 0-7 of the parameter block and the cryptographic key is in byte offsets 8-31 of the parameter block.

The parameter block used for the KMO-Encrypted-TDEA-192 function has the following format:

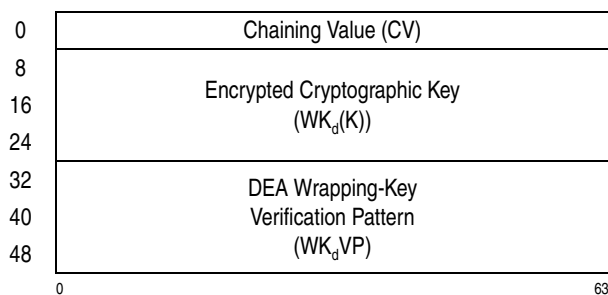


Figure 7-157. Parameter Block for KMO-Encrypted-TDEA-192

For the KMO-Encrypted-TDEA-192 function, the contents of byte offsets 32-55 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the chaining value, and the contents of byte offsets 8-31 of the parameter block are deciphered using the DEA wrapping key to obtain the 192-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 8-byte plaintext blocks (P_1, P_2, \dots, P_n) in operand 2 are enciphered using the TDEA-encryption algorithm with the 192-bit cryptographic key and the 64-bit chaining value.

The first input block to the TDEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding plaintext block to form a ciphertext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a ciphertext block is produced for every plaintext block, or until a CPU-determined number of ciphertext blocks have been produced.

The ciphertext blocks (C_1, C_2, \dots, C_n) are stored in operand 1. The next input block, called the output

chaining value (OCV), is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-158.

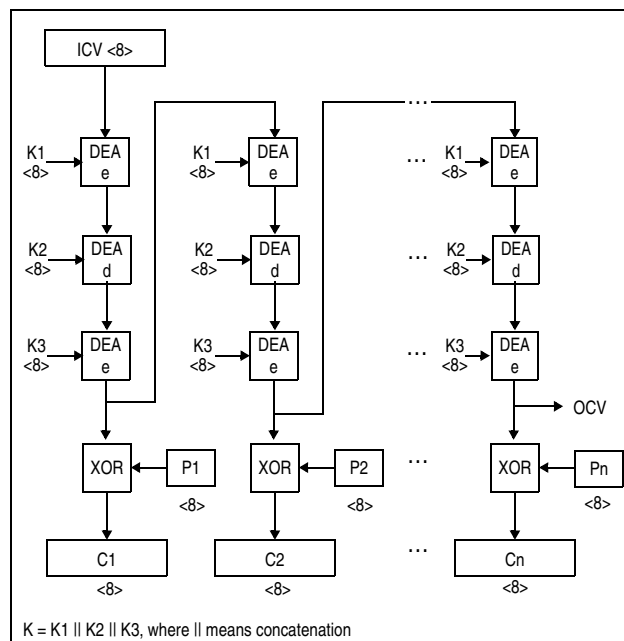


Figure 7-158. KMO-TDEA-192 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The 8-byte ciphertext blocks (C_1, C_2, \dots, C_n) in operand 2 are deciphered using the DEA-encryption algorithm with the 192-bit cryptographic key and the 64-bit chaining value.

The first input block to the TDEA-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding ciphertext block to form a plaintext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a plaintext block is produced for every ciphertext block, or until a CPU-determined number of plaintext blocks have been produced.

The plaintext blocks (P_1, P_2, \dots, P_n) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-

value field of the parameter block. The operation is shown in Figure 7-159.

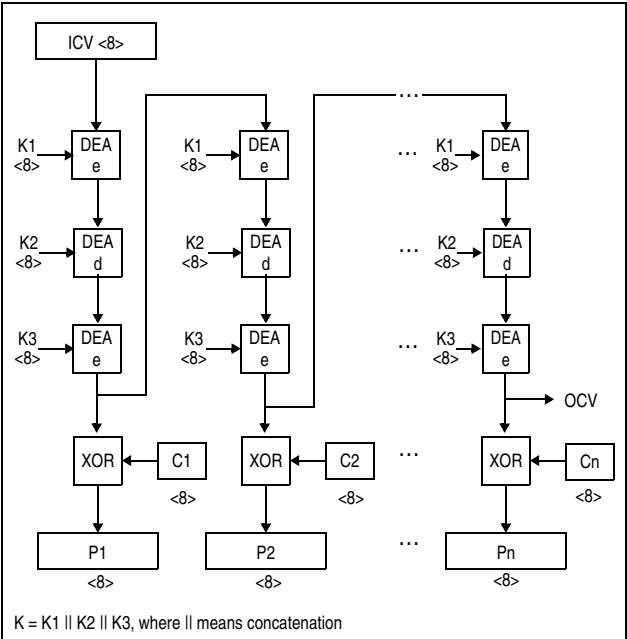


Figure 7-159. KMO-TDEA-192 Decipher Operation

KMO-AES-128 (Function Code 18)

KMO-Encrypted-AES-128 (Function Code 26)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-141 on page 7-100.

The parameter block used for the KMO-AES-128 function has the following format:

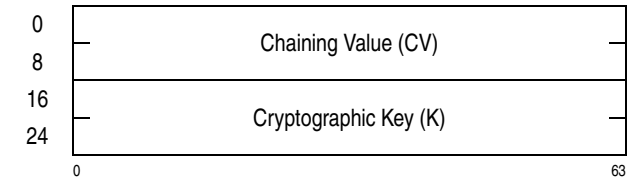


Figure 7-160. Parameter Block for KMO-AES-128

For the KMO-AES-128 function, the chaining value is in byte offset 0-15 of the parameter block and the cryptographic key is in byte offsets 16-31 of the parameter block.

The parameter block used for the KMO-Encrypted-AES-128 function has the following format:

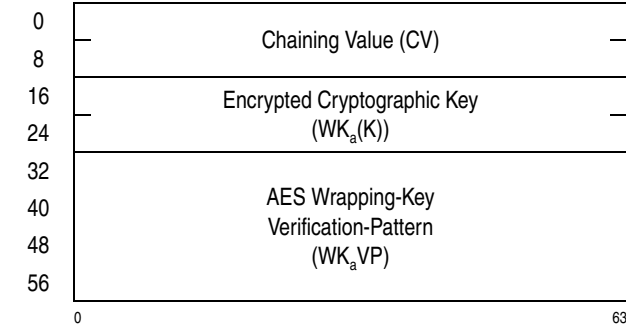


Figure 7-161. Parameter Block for KMO-Encrypted-AES-128

For the KMO-Encrypted-AES-128 function, the contents of byte offsets 32-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block contain the chaining value, and the contents of byte offsets 16-31 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES-encryption algorithm with the 128-bit cryptographic key and the 128-bit chaining value.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding plaintext block to form a ciphertext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a ciphertext block is produced for every plaintext block, or until a CPU-determined number of ciphertext blocks have been produced.

The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-

value field of the parameter block. The operation is shown in Figure 7-162.

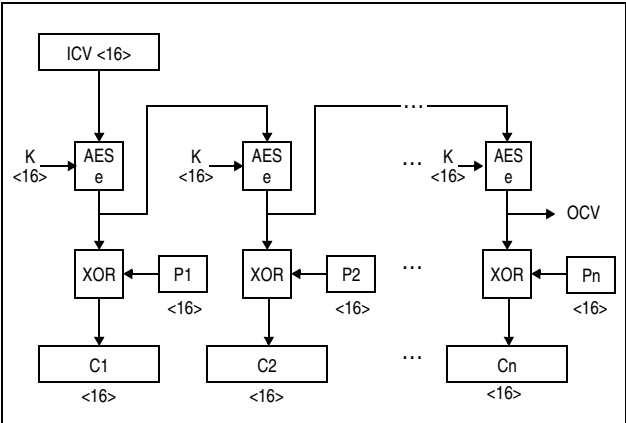


Figure 7-162. KMO-AES-128 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES-encryption algorithm with the 128-bit cryptographic key and the 128-bit chaining value.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding ciphertext block to form a plaintext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a plaintext block is produced for every ciphertext block, or until a CPU-determined number of plaintext blocks have been produced.

The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-

value field of the parameter block. The operation is shown in Figure 7-163.

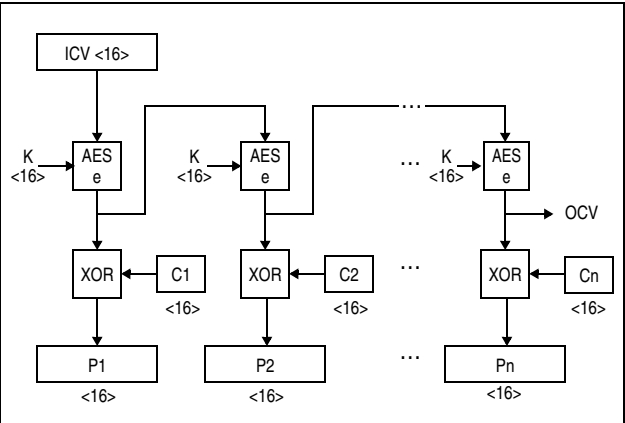


Figure 7-163. KMO-AES-128 Decipher Operation

KMO-AES-192 (Function Code 19)

KMO-Encrypted-AES-192 (Function Code 27)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-141 on page 7-100.

The parameter block used for the KMO-AES-192 function has the following format:

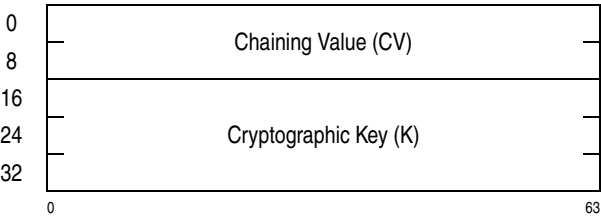


Figure 7-164. Parameter Block for KMO-AES-192

For the KMO-AES-192 function, the chaining value is in byte offset 0-15 of the parameter block and the cryptographic key is in byte offsets 16-39 of the parameter block.

The parameter block used for the KMO-Encrypted-AES-192 function has the following format:

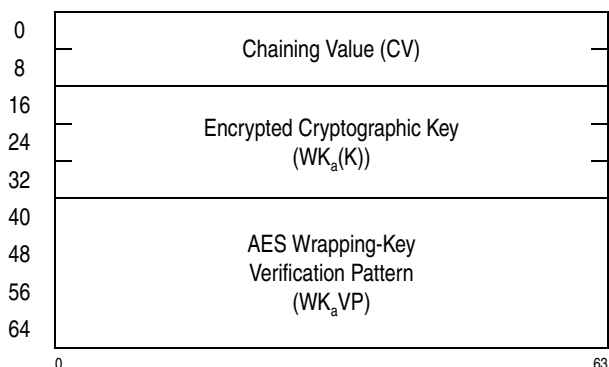


Figure 7-165. Parameter Block for KMO-Encrypted-AES-192

For the KMO-Encrypted-AES-192 function, the contents of byte offsets 40-71 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block contain the chaining value, and the contents of byte offsets 16-39 of the parameter block are deciphered using the AES wrapping key to obtain the 192-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES-encryption algorithm with the 192-bit cryptographic key and the 128-bit chaining value.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding plaintext block to form a ciphertext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a ciphertext block is produced for every

plaintext block, or until a CPU-determined number of ciphertext blocks have been produced.

The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-166.

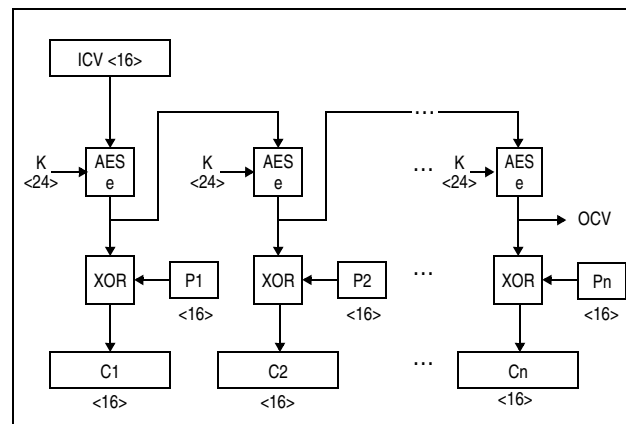


Figure 7-166. KMO-AES-192 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES-encryption algorithm with the 192-bit cryptographic key and the 128-bit chaining value.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding ciphertext block to form a plaintext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a plaintext block is produced for every ciphertext block, or until a CPU-determined number of plaintext blocks have been produced.

The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-

value field of the parameter block. The operation is shown in Figure 7-167.

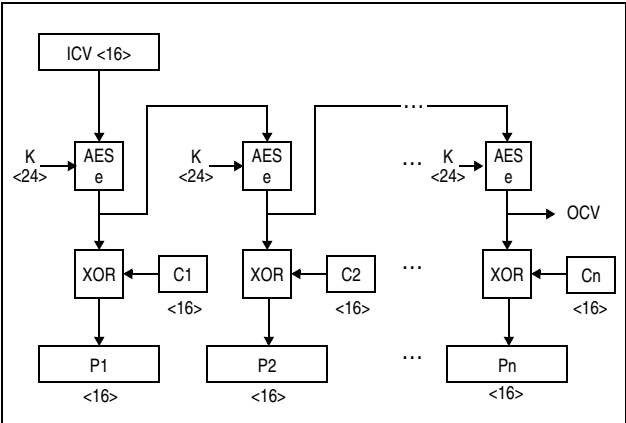


Figure 7-167. KMO-AES-192 Decipher Operation

KMO-AES-256 (Function Code 20)

KMO-Encrypted-AES-256 (Function Code 28)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-141 on page 7-100.

The parameter block used for the KMO-AES-256 function has the following format:

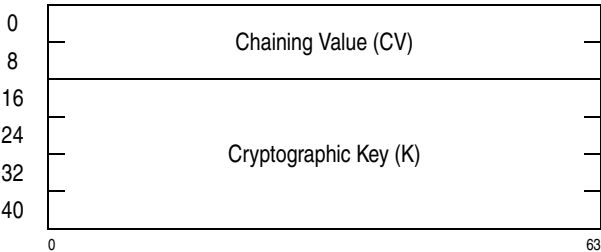


Figure 7-168. Parameter Block for KMO-AES-256

For the KMO-AES-256 function, the chaining value is in byte offset 0-15 of the parameter block and the cryptographic key is in byte offsets 16-47 of the parameter block.

The parameter block used for the KMO-Encrypted-AES-256 function has the following format:

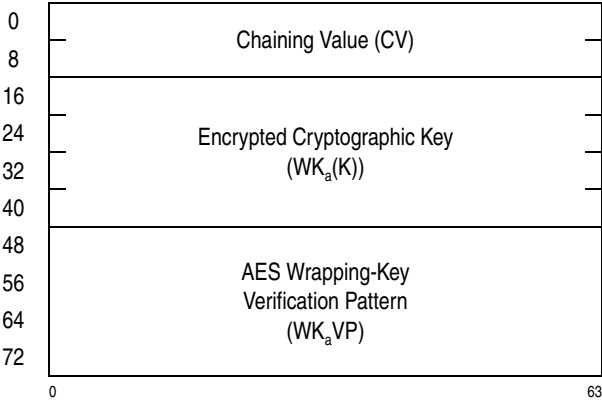


Figure 7-169. Parameter Block for KMO-AES-256

For the KMO-Encrypted-AES-256 function, the contents of byte offsets 48-79 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block contain the chaining value, and the contents of byte offsets 16-47 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The following description applies to both functions.

When the modifier bit in general register 0 is zero, an encipher operation is performed. The 16-byte plaintext blocks (P1, P2, ..., Pn) in operand 2 are enciphered using the AES-encryption algorithm with the 256-bit cryptographic key and the 128-bit chaining value.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding plaintext block to form a ciphertext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a ciphertext block is produced for every plaintext block, or until a CPU-determined number of ciphertext blocks have been produced.

The ciphertext blocks (C1, C2, ..., Cn) are stored in operand 1. The next input block, called the output

chaining value (OCV), is stored into the chaining-value field of the parameter block. The operation is shown in Figure 7-170.

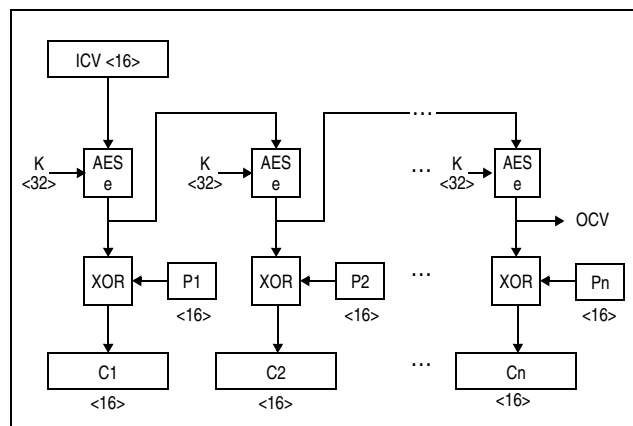


Figure 7-170. KMO-AES-256 Encipher Operation

When the modifier bit in general register 0 is one, a decipher operation is performed. The 16-byte ciphertext blocks (C1, C2, ..., Cn) in operand 2 are deciphered using the AES-encryption algorithm with the 256-bit cryptographic key and the 128-bit chaining value.

The first input block to the AES-encryption algorithm is the initial chaining value (ICV) in the parameter block. Each input block is enciphered to produce an output block. Each output block is exclusive-ORed with the corresponding ciphertext block to form a plaintext block. Each output block is also used as the next input block.

The process is repeated with the successive input blocks until a plaintext block is produced for every ciphertext block, or until a CPU-determined number of plaintext blocks have been produced.

The plaintext blocks (P1, P2, ..., Pn) are stored in operand 1. The next input block, called the output chaining value (OCV), is stored into the chaining-

value field of the parameter block. The operation is shown in Figure 7-171.

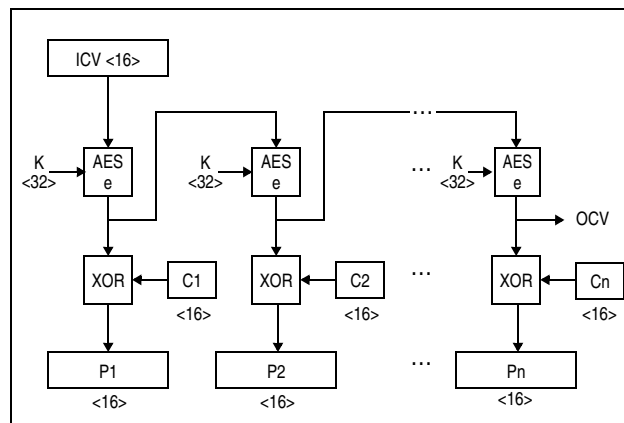


Figure 7-171. KMO-AES-256 Decipher Operation

Special Conditions

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bits 57-63 of general register 0 specify an unassigned or uninstalled function code.
2. The R_1 or R_2 field designates an odd-numbered register or general register 0.
3. The second operand length is not a multiple of the data block size of the designated function (see Figure 7-140 on page 7-99 to determine the data block sizes for CIPHER MESSAGE WITH OFB functions). This specification-exception condition does not apply to the query functions.

Resulting Condition Code:

- | | |
|---|-------------------------------|
| 0 | Normal completion |
| 1 | Verification pattern mismatch |
| 2 | -- |
| 3 | Partial completion |

Program Exceptions:

- Access (fetch, operand 2 and cryptographic key; store, operand 1; fetch and store, chaining value)
- Operation (if the message-security-assist extension 4 facility is not installed)
- Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Operation exception.
8.	Specification exception due to invalid function code or invalid register number.
9.	Specification exception due to invalid operand length.
10.	Condition code 0 due to second-operand length originally zero.
11.	Access exceptions for an access to the parameter block, first, or second operand.
12.	Condition code 1 due to verification pattern mismatch.
13.	Condition code 0 due to normal completion (second-operand length originally nonzero, but stepped to zero).
14.	Condition code 3 due to partial completion (second-operand length still nonzero).

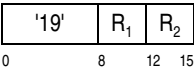
Figure 7-172. Priority of Execution: KMO

Programming Note: See the programming notes for CIPHER MESSAGE WITH CHAINING.

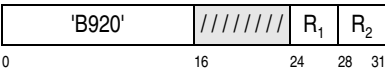
COMPARE

Register-and-register formats:

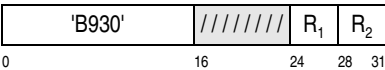
CR R₁,R₂ [RR]



CGR R₁,R₂ [RRE]

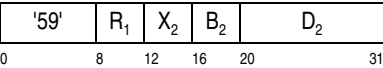


CGFR R₁,R₂ [RRE]

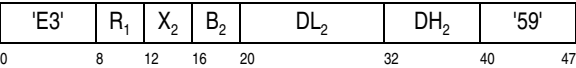


Register-and-storage formats:

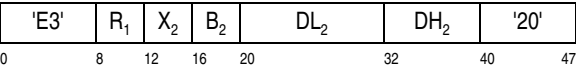
C R₁,D₂(X₂,B₂) [RX-a]



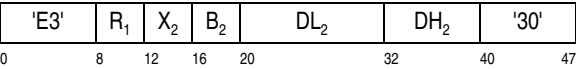
CY R₁,D₂(X₂,B₂) [RXY-a]



CG R₁,D₂(X₂,B₂) [RXY-a]

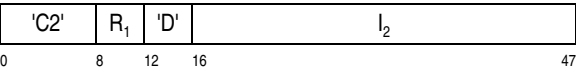


CGF R₁,D₂(X₂,B₂) [RXY-a]

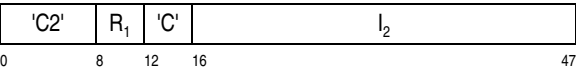


COMPARE IMMEDIATE

CFI R₁,I₂ [RIL-a]

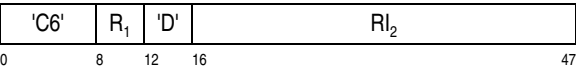


CGFI R₁,I₂ [RIL-a]

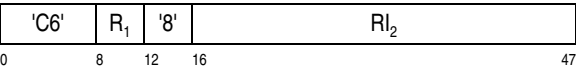


COMPARE RELATIVE LONG

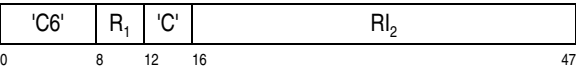
CRL R₁,Rl₂ [RIL-b]



CGRL R₁,Rl₂ [RIL-b]



CGFRL R₁,Rl₂ [RIL-b]



The first operand is compared with the second operand, and the result is indicated in the condition code.

For COMPARE (CR, C, CY), COMPARE IMMEDIATE (CFI), and COMPARE RELATIVE LONG (CRL), the operands are treated as 32-bit signed binary integers. For COMPARE (CGR, CG) and COMPARE RELATIVE LONG (CGRL), they are treated as 64-bit signed binary integers. For COMPARE (CGFR, CGF), COMPARE IMMEDIATE (CGFI), and COMPARE RELATIVE LONG (CGFRL), the second operand is treated as a 32-bit signed binary integer, and the first operand is treated as a 64-bit signed binary integer.

The displacement for C is treated as a 12-bit unsigned binary integer. The displacement for CY, CG, and CGF is treated as a 20-bit signed binary integer.

For COMPARE RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

For COMPARE RELATIVE LONG (CRL, CGFRL), the second operand must be aligned on a word boundary, and for COMPARE RELATIVE LONG (CGRL), the second operand must be aligned on a doubleword boundary; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 --

Program Exceptions:

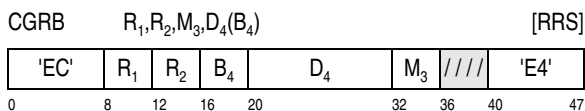
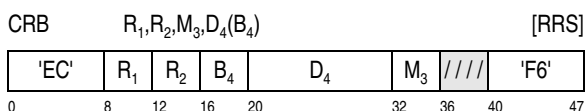
- Access (operand 2 of C, CY, CG, CGF, CRL, CGRL, and CGFRL only)
- Operation (CY, if the long-displacement facility is not installed; CFI and CGFI, if the extended-immediate facility is not installed; CGFRL, CGRL, and CRL, if the general-instructions-extension facility is not installed)

- Specification (CGFRL, CGRL, and CRL only)

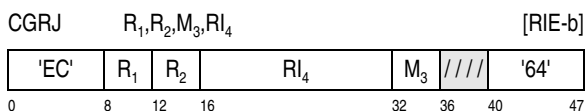
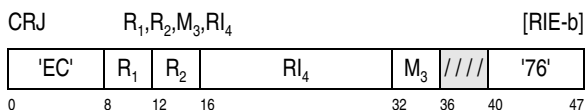
Programming Notes:

1. For COMPARE RELATIVE LONG, the second operand must be aligned on an integral boundary corresponding to the operand's size.
2. When COMPARE RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.

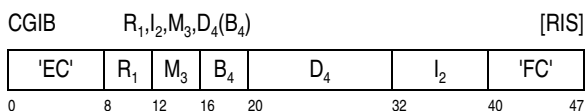
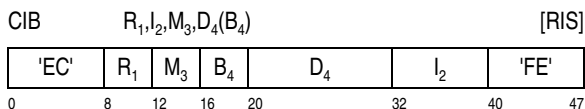
COMPARE AND BRANCH



COMPARE AND BRANCH RELATIVE



COMPARE IMMEDIATE AND BRANCH



COMPARE IMMEDIATE AND BRANCH RELATIVE



The first operand is compared with the second operand. If the mask bit in the M₃ field corresponding to the comparison result is one, the instruction address in the current PSW is replaced by the branch address specified by the fourth operand; otherwise, normal instruction sequencing proceeds with the updated instruction address.

For COMPARE AND BRANCH (CRB), COMPARE AND BRANCH RELATIVE (CRJ), COMPARE IMMEDIATE AND BRANCH (CIB) and COMPARE IMMEDIATE AND BRANCH RELATIVE (CIJ), the first operand is treated as a 32-bit signed binary integer. For COMPARE AND BRANCH (CGRB), COMPARE AND BRANCH RELATIVE (CGRJ), COMPARE IMMEDIATE AND BRANCH (CGIB) and COMPARE IMMEDIATE AND BRANCH RELATIVE (CGIJ), the first operand is treated as a 64-bit signed binary integer.

For COMPARE AND BRANCH (CRB) and COMPARE AND BRANCH RELATIVE (CRJ), the second operand is treated as a 32-bit signed binary integer. For COMPARE AND BRANCH (CGRB) and COMPARE AND BRANCH RELATIVE (CGRJ), the second operand is treated as a 64-bit signed binary integer. For COMPARE IMMEDIATE AND BRANCH and COMPARE IMMEDIATE AND BRANCH RELATIVE, the second operand is treated as an 8-bit signed binary integer.

The comparison results and corresponding M₃ bits are as follows:

Comparison Result	M ₃ Bit
Equal	0
First operand low	1
First operand high	2

Bit 3 of the M₃ field is reserved and should be zero; otherwise, the program may not operate compatibly in the future.

For COMPARE AND BRANCH and COMPARE IMMEDIATE AND BRANCH, the fourth operand is used as the branch address. For COMPARE AND BRANCH RELATIVE and COMPARE IMMEDIATE AND BRANCH RELATIVE, the contents of the I₄ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

Condition Code: The code remains unchanged.

Program Exceptions:

- Operation (if the general-instructions-extension facility is not installed).

Programming Notes:

1. When bit positions 0-2 of the M₃ field contain zeros, the instruction acts as a NOP, however this is not the preferred instruction with which to create a NOP. When bit positions 0-2 of the M₃ field contain 111 binary, a branch always occurs.
2. When COMPARE AND BRANCH RELATIVE or COMPARE IMMEDIATE AND BRANCH RELATIVE are the target of an execute-type instruction, the branch is relative to the target address. See "Branch-Address Generation" on page 5-10.
3. The high-level assembler (HLASM) provides extended-mnemonic formats for all of the multiple-operation instructions which perform a compare (arithmetic or logical) followed by a branch or trap. In the extended-mnemonic format, the mask field is not explicitly coded; rather the branch mask is implicitly indicated by the extended mnemonic.

The extended-mnemonic names are formed by concatenating the base mnemonic, followed by one of the suffix characters, as shown below.

Suffix Chars	Meaning	Mask Field
E	Equal	8
H	First operand high	2
L	First operand low	4
NE	Not equal	6
NH	First operand not high	12

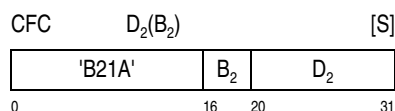
Suffix Chars	Meaning	Mask Field
NL	First operand not low	10

Consider the following example of a COMPARE IMMEDIATE AND BRANCH RELATIVE (CGIJ) instruction that is coded using an extended mnemonic:

```
CGIJNE 10,-123,SKIP
:
SKIP DS 0H (128 bytes past CGIJNE)
```

The implied mask field corresponding to the mnemonic suffix (NE) is 6, thus the generated code is ECA60040857C hex.

COMPARE AND FORM CODEWORD



General register 2 contains an index, which is used along with contents of general registers 1 and 3 to designate the starting addresses of two fields in storage, called the first and third operands. The first and third operands are logically compared, and a codeword is formed for use in sort/merge algorithms.

The second-operand address is not used to address data. Bits 49-62 of the second-operand address, with one rightmost and one leftmost zero appended, are used as a 16-bit index limit. Bit 63 of the second-operand address is the operand-control bit. When bit 63 is zero, the codeword is formed from the high operand; when bit 63 is one, the codeword is formed from the low operand. The remainder of the second-operand address is ignored.

General registers 1 and 3 contain the base addresses of the first and third operands. Bits 48-63 of general register 2 are used as an index for addressing both the first and third operands. General registers 1, 2, and 3 must all initially contain even values; otherwise, a specification exception is recognized.

In the access-register mode, access register 1 specifies the address space containing the first and third operands.

The size of the units by which the first and third operands are compared, the size of the resulting codeword, and the participation of bits 0-31 of general registers 1, 2, and 3 in the operation depend on the addressing mode. In the 24-bit or 31-bit addressing mode, the comparison unit is two bytes, the codeword is four bytes, and bits 0-31 are ignored and remain unchanged. In the 64-bit addressing mode, the comparison unit is six bytes, the codeword is eight bytes, and bits 0-31 are used in and may be changed by the operation.

Operation in the 24-Bit or 31-Bit Addressing Mode

The operation consists in comparing the first and third operands halfword by halfword and incrementing the index until an unequal pair of halfwords is found or the index exceeds the index limit. This proceeds in units of operation, between which interruptions may occur.

At the start of a unit of operation, the index, bits 48-63 of general register 2, is logically compared with the index limit. If the index is larger, the instruction is completed by placing bits 32-63 of general register 3, with bit 32 set to one, in bit positions 32-63 of general register 2, and by setting condition code 0.

If the index is less than or equal to the index limit, the index is applied to the first-operand and third-operand base addresses to locate the current pair of halfwords to be compared. The index, with 48 leftmost zeros appended, and bits 32-63 of general register 1, with 32 leftmost zeros appended, are added to form a 64-bit intermediate value. A carry out of bit position 32, if any, is ignored. The address of the current first-operand halfword is generated from the intermediate value by following the normal rules for operand address generation. The address of the current third-operand halfword is formed in the same manner by adding bits 32-63 of general register 3 and the index.

The current first-operand and third-operand halfwords are logically compared. If they are equal, the contents of general register 2 are incremented by 2, and a unit of operation ends.

If the compare values are unequal, the contents of general register 2 are incremented by 2 and then shifted left logically by 16 bit positions. The shifting occurs only within bit positions 32-63. If the operand-control bit is zero, (1) the one's complement of the higher halfword is placed in bit positions 48-63 of

general register 2, and (2) if operand 1 was higher, bits 32-63 of general registers 1 and 3 are interchanged. If the operand-control bit is one, (1) the lower halfword is placed in bit positions 48-63 of general register 2, and (2) if operand 1 was lower, bits 32-63 of general registers 1 and 3 are interchanged.

For the purpose of recognizing access exceptions, operand 1 and operand 3 are both considered to have a length equal to 2 more than the value of the index limit minus the index.

Operation in the 64-bit Addressing Mode

The operation consists in comparing the first and third operands in units of six bytes at a time and incrementing the index until an unequal pair of six-byte units is found or the index exceeds the index limit. This proceeds in units of operation, between which interruptions may occur.

At the start of a unit of operation, the index, bits 48-63 of general register 2, is logically compared with the index limit. If the index is larger, the instruction is completed by placing bits 0-63 of general register 3, with bit 0 set to one, in bit positions 0-63 of general register 2, and by setting condition code 0.

If the index is less than or equal to the index limit, the index is applied to the first-operand and third-operand base addresses to locate the current pair of six-byte units to be compared. The index, with 48 left-most zeros appended, and bits 0-63 of general register 1 are added to form the 64-bit address of the current first-operand six-byte unit. A carry out of bit position 0, if any, is ignored. The address of the current third-operand six-byte unit is formed in the same manner by adding bits 0-63 of general register 3 and the index.

The current first-operand and third-operand six-byte units are logically compared. If they are equal, the contents of general register 2 are incremented by 6, and a unit of operation ends.

If the compare values are unequal, the contents of general register 2 are incremented by 6 and then shifted left logically by 48 bit positions. If the operand-control bit is zero, (1) the one's complement of the higher six-byte unit is placed in bit positions 16-63 of general register 2, and (2) if operand 1 was higher, bits 0-63 of general registers 1 and 3 are interchanged. If the operand-control bit is one, (1) the lower six-byte unit is placed in bit positions 16-63 of

general register 2, and (2) if operand 1 was lower, bits 0-63 of general registers 1 and 3 are interchanged.

For the purpose of recognizing access exceptions, operand 1 and operand 3 are both considered to have a length equal to 6 more than the value of the index limit minus the index.

Specifications Independent of Addressing Mode

The condition code is unpredictable if the instruction is interrupted.

When the index is initially larger than the index limit, access exceptions are not recognized for the storage operands. For operands longer than 4K bytes, access exceptions are not recognized more than 4K bytes beyond the byte being processed. Access exceptions are not recognized when a specification-exception condition exists.

If the B_2 field designates general register 2, it is unpredictable whether or not the index limit is recomputed; thus, in this case the operand length is unpredictable. However, in no case can the operands exceed 2^{15} bytes in length.

Resulting Condition Code:

- 0 Operands equal
- 1 Operand-control bit zero and operand 1 low, or operand-control bit one and operand 3 low
- 2 Operand-control bit zero and operand 1 high, or operand-control bit one and operand 3 high
- 3 --

Program Exceptions:

- Access (fetch, operands 1 and 3)
- Specification

Programming Notes:

1. An example of the use of COMPARE AND FORM CODEWORD is given in "Sorting Instructions" in Appendix A, "Number Representation and Instruction-Use Examples."
2. The offset of the halfword or six-byte unit (depending on the addressing mode) of the first and third operands at which comparison is to begin should be placed in bit positions 48-63 of general register 2 before executing COMPARE

AND FORM CODEWORD. The index limit derived from the second-operand address should be the offset of the last halfword or six-byte unit of the first and third operands for which comparison can be made. When the operands do not compare equal, the leftmost 16 bits of the codeword formed in general register 2 (bits 32-47 of the register in the 24-bit or 31-bit addressing mode, or bits 0-15 in the 64-bit addressing mode) by the execution of COMPARE AND FORM CODEWORD gives the offset of the first halfword or six-byte unit not compared. If the codewords compare equal in an UPDATE TREE operation, bit positions 32-47 of general register 2 in the 24-bit or 31-bit addressing mode, or bit positions 0-15 in the 64-bit addressing mode, will contain the offset at which another COMPARE AND FORM CODEWORD should resume comparison for breaking codeword ties. Operand-control-bit values of zero or one are used for sorting operands in ascending or descending order, respectively. Refer to “Sorting Instructions” on page A-53 for a discussion of the use of codewords in sorting.

3. The condition code indicates the results of comparing operands up to 32,768 bytes long. Equal operands result in a negative codeword in bit positions 32-63 of general register 2 in the 24-bit or 31-bit addressing mode, or in bit positions 0-63 in the 64-bit addressing mode. A negative codeword also results in the 24-bit or 31-bit mode when the index limit is 32,766 and the operands that are compared differ in only their last two bytes, or in the 64-bit mode when the limit is 32,762 and the operands differ in only their last six bytes. If this latter codeword is used by UPDATE TREE, an incorrect result may be indicated in general registers 0 and 1. Therefore, the index limit should not exceed 32,764 in the 24-bit or 31-bit mode, or 32,760 in the 64-bit mode, when the resulting codeword is to be used by UPDATE TREE.
4. Special precautions should be taken if COMPARE AND FORM CODEWORD is made the target of an execute-type instruction. See the programming note concerning interruptible instructions under EXECUTE.
5. Further programming notes concerning interruptible instructions are included in “Interruptible Instructions” in Chapter 5, “Program Execution.”
6. The storage-operand references of COMPARE AND FORM CODEWORD may be multiple-access references. (See “Storage-Operand Consistency”.)
7. Figure 7-173 on page 7-118 and Figure 7-174 on page 7-119 contain summaries of the operation in the 24-bit or 31-bit addressing mode, and Figure 7-175 on page 7-120 and Figure 7-176 on page 7-121 contain summaries of the operation in the 64-bit addressing mode.

Operand- Control Bit	Relation	Resulting Condition Code	Result in GR2 (Bits 32-63)	Result in GR1 (Bits 32-63)	Result in GR3 (Bits 32-63)
0	op1 = op3	0	OGR3b1	–	–
0	op1 < op3	1	X, nop3	–	–
0	op1 > op3	2	X, nop1	OGR3	OGR1
1	op1 = op3	0	OGR3b1	–	–
1	op1 < op3	2	X, top1	OGR3	OGR1
1	op1 > op3	1	X, top3	–	–
Explanation: – The bits remain unchanged. OGR1 The original value of GR1 bits 32-63. OGR3 The original value of GR3 bits 32-63. OBR3b1 The original value of GR3 bits 32-63 with bit 32 set to one. X Bits 32-47 of GR2 are 2 more than the index of the first unequal halfword. nop1 Bits 48-63 of GR2 are the one's complement of the first unequal halfword in operand 1. nop3 Bits 48-63 of GR2 are the one's complement of the first unequal halfword in operand 3. top1 Bits 48-63 of GR2 are the first unequal halfword in operand 1. top3 Bits 48-63 of GR2 are the first unequal halfword in operand 3.					

Figure 7-173. Operation of COMPARE AND FORM CODEWORD in the 24-Bit or 31-bit Addressing Mode

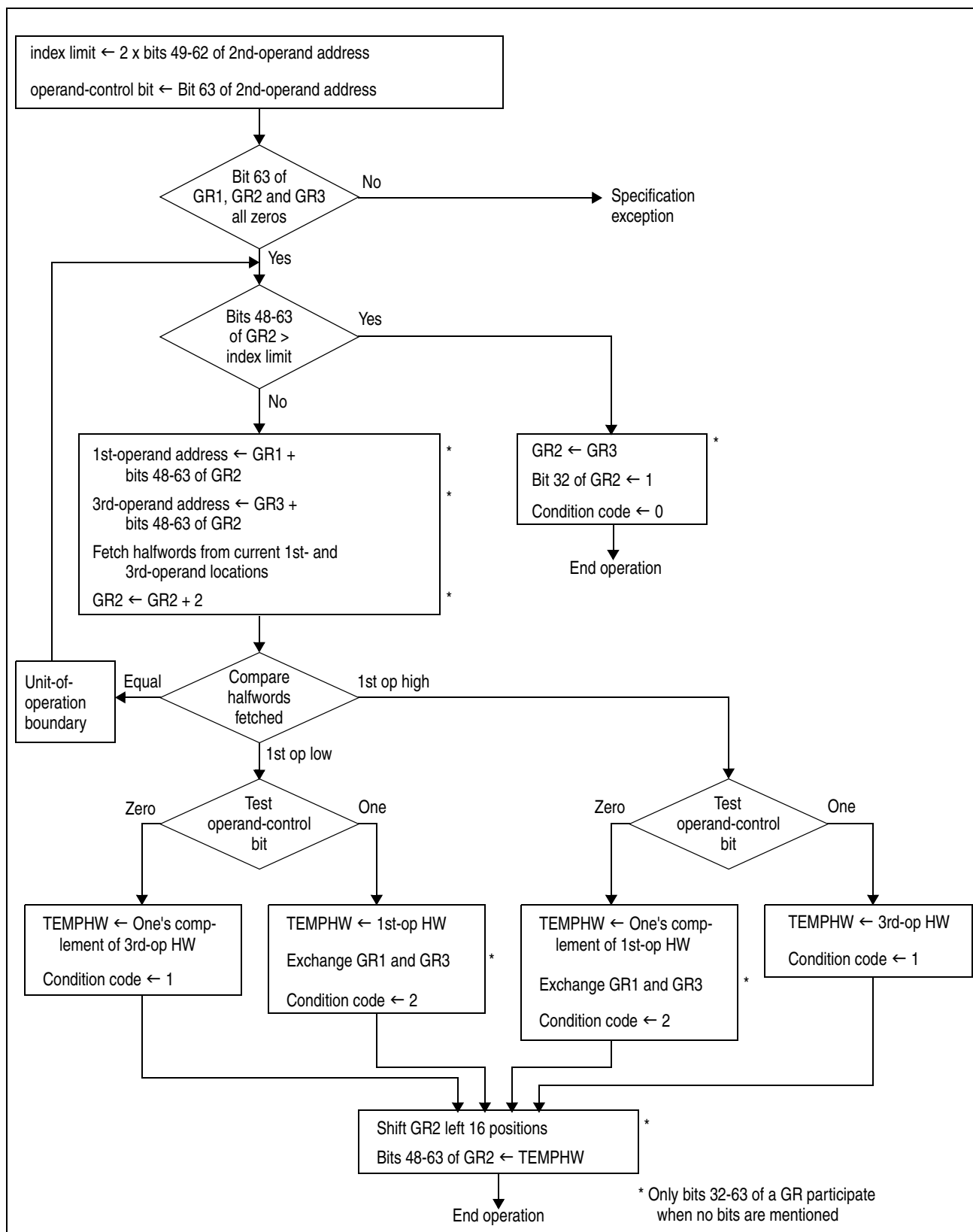


Figure 7-174. Execution of COMPARE AND FORM CODEWORD in the 24-Bit or 31-bit Addressing Mode

Operand- Control Bit	Relation	Resulting Condition Code	Result in GR2 (Bits 0-63)	Result in GR1 (Bits 0-63)	Result in GR3 (Bits 0-63)
0	op1 = op3	0	OGR3b1	–	–
0	op1 < op3	1	X, nop3	–	–
0	op1 > op3	2	X, nop1	OGR3	OGR1
1	op1 = op3	0	OGR3b1	–	–
1	op1 < op3	2	X, top1	OGR3	OGR1
1	op1 > op3	1	X, top3	–	–
Explanation: – The bits remain unchanged. OGR1 The original value of GR1 bits 0-63. OGR3 The original value of GR3 bits 0-63. OBR3b1 The original value of GR3 bits 0-63 with bit 0 set to one. X Bits 0-15 of GR2 are 6 more than the index of the first unequal six-byte unit. nop1 Bits 16-63 of GR2 are the one's complement of the first unequal six-byte unit in operand 1. nop3 Bits 16-63 of GR2 are the one's complement of the first unequal six-byte unit in operand 3. top1 Bits 16-63 of GR2 are the first unequal six-byte unit in operand 1. top3 Bits 16-63 of GR2 are the first unequal six-byte unit in operand 3.					

Figure 7-175. Operation of COMPARE AND FORM CODEWORD in the 64-bit Addressing Mode

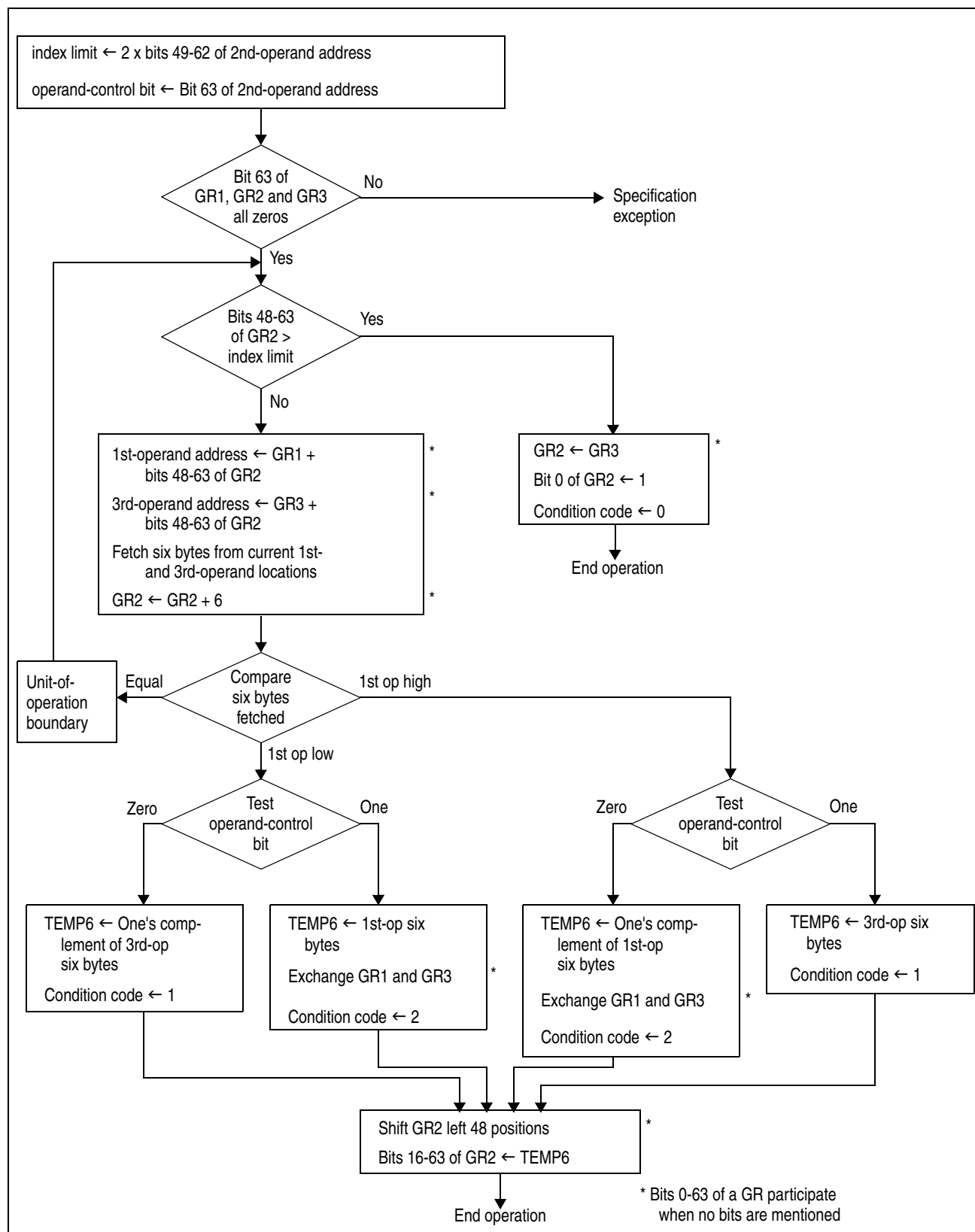
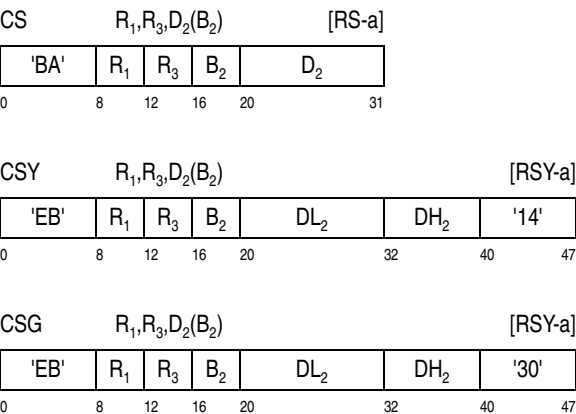
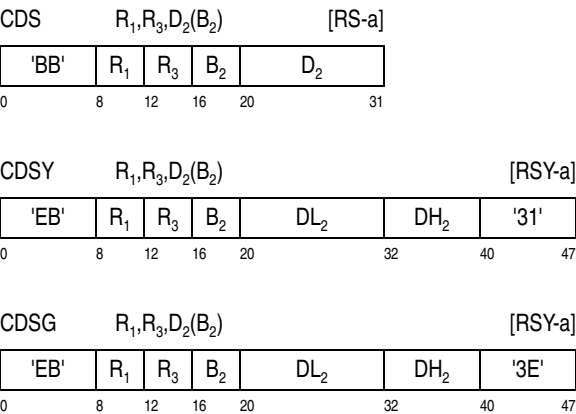


Figure 7-176. Execution of COMPARE AND FORM CODEWORD in the 64-bit Addressing Mode

COMPARE AND SWAP



COMPARE DOUBLE AND SWAP



The first and second operands are compared. If they are equal, the third operand is stored at the second-operand location. If they are unequal, the second operand is loaded into the first-operand location. The result of the comparison is indicated in the condition code.

For COMPARE AND SWAP (CS, CSY), the first and third operands are 32 bits in length, with each operand occupying bit positions 32-63 of a general register. The second operand is a word in storage.

For COMPARE AND SWAP (CSG), the first and third operands are 64 bits in length, with each operand occupying bit positions 0-63 of a general register. The second operand is a doubleword in storage.

For COMPARE DOUBLE AND SWAP (CDS, CDSY), the first and third operands are 64 bits in length. The first 32 bits of an operand occupy bit positions 32-63 of the even-numbered register of an even-odd pair of general registers, and the second 32 bits occupy bit positions 32-63 of the odd-numbered register of the pair. The second operand is a doubleword in storage.

For COMPARE DOUBLE AND SWAP (CDSG), the first and third operands are 128 bits in length. The first 64 bits of an operand occupy bit positions 0-63 of the even-numbered register of an even-odd pair of general registers, and the second 64 bits occupy bit positions 0-63 of the odd-numbered register of the pair. The second operand is a quadword in storage.

When an equal comparison occurs, the third operand is stored at the second-operand location. The fetch of the second operand for purposes of comparison and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs.

When the result of the comparison is unequal, the second operand is loaded at the first-operand location, and the second-operand location remains unchanged. However, on some models, the contents may be fetched and subsequently stored back unchanged at the second-operand location. This update appears to be a block-concurrent interlocked-update reference as observed by other CPUs.

A serialization function is performed before the operand is fetched and again after the operation is completed.

The displacement for CS and CDS is treated as a 12-bit unsigned binary integer. The displacement for CSY, CSG, CDSY, and CDSG is treated as a 20-bit signed binary integer.

The second operand of COMPARE AND SWAP (CS, CSY) must be designated on a word boundary. The second operand of COMPARE AND SWAP (CSG) and COMPARE DOUBLE AND SWAP (CDS, CDSY) must be designated on a doubleword boundary. The second operand of COMPARE DOUBLE AND SWAP (CDSG) must be designated on a quadword boundary. The R_1 and R_3 fields for COMPARE DOUBLE AND SWAP must each designate an even-numbered register. Otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 First and second operands equal, second operand replaced by third operand
- 1 First and second operands unequal, first operand replaced by second operand
- 2 --
- 3 --

Program Exceptions:

- Access (fetch and store, operand 2)
- Operation (CSY and CDSY, if the long-displacement facility is not installed)
- Specification

Programming Notes:

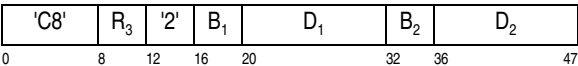
1. Several examples of the use of the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. Some of the following notes are worded, with respect to operand size, for CS, CSY, CDS, and CDSY. Similar notes, worded for a larger operand size, would apply to CSG and CDSG.
3. COMPARE AND SWAP can be used by CPU programs sharing common storage areas in either a multiprogramming or multiprocessing environment. Two examples are:
 - a. By performing the following procedure, a CPU program can modify the contents of a storage location even though the possibility exists that the CPU program may be interrupted by another CPU program that will update the location or that another CPU program may simultaneously update the location. First, the entire word containing the byte or bytes to be updated is loaded into a general register. Next, the updated value is computed and placed in another general register. Then COMPARE AND SWAP is executed with the R_1 field designating the register that contains the original value and the R_3 field designating the register that contains the updated value. If the update has been successful, condition code 0 is set. If the storage location no longer contains the original value, the update has not been successful, the general register designated by the R_1 field of the COMPARE AND SWAP instruction contains the new current value of the storage location, and condition code 1 is set. When condition code 1 is set, the CPU program can repeat the procedure using the new current value.
 - b. COMPARE AND SWAP can be used for controlled sharing of a common storage area, including the capability of leaving a message (in a chained list of messages) when the common area is in use. To accomplish this, a word in storage can be used as a control word, with a zero value in the word indicating that the common area is not in use and that no messages exist, a negative value indicating that the area is in use and that no messages exist, and a nonzero positive value indicating that the common area is in use and that the value is the address of the most recent message added to the list. Thus, any number of CPU programs desiring to seize the area can use COMPARE AND SWAP to update the control word to indicate that the area is in use or to add messages to the list. The single CPU program which has seized the area can also safely use COMPARE AND SWAP to remove messages from the list.
4. COMPARE DOUBLE AND SWAP can be used in a manner similar to that described for COMPARE AND SWAP. In addition, it has another use. Consider a chained list, with a control word used to address the first message in the list, as described in programming note 3.b above. If multiple CPU programs are to be permitted to delete messages by using COMPARE AND SWAP (and not just the single CPU program which has seized the common area), there is a possibility the list will be incorrectly updated. This would occur if, for example, after one CPU program has fetched the address of the most recent message in order to remove the message, another CPU program removes the first two messages and then adds the first message back into the chain. The first CPU program, on continuing, cannot easily detect that the list is changed. By increasing the size of the control word to a doubleword containing both the first message address and a word with a change number that is incremented for each modification of the list, and by using COMPARE DOUBLE AND SWAP to update both fields together, the possibility of the list being incorrectly updated is reduced to a negligible

level. That is, an incorrect update can occur only if the first CPU program is delayed while changes exactly equal in number to a multiple of 2^{32} take place *and* only if the last change places the original message address in the control word.

- 5. COMPARE AND SWAP and COMPARE DOUBLE AND SWAP do not interlock against storage accesses by channel programs. Therefore, the instructions should not be used to update a location at which a channel program may store, since the channel-program data may be lost.
- 6. To ensure successful updating of a common storage field by two or more CPUs, all updates must be done by means of an interlocked-update reference (the section “Interlocked-Update References” on page 5-94 lists the instructions that perform an interlocked-update reference). For example, if one CPU executes OR IMMEDIATE and another CPU executes COMPARE AND SWAP to update the same byte, the fetch by OR IMMEDIATE may occur either before the fetch by COMPARE AND SWAP or between the fetch and the store by COMPARE AND SWAP, and then the store by OR IMMEDIATE may occur after the store by COMPARE AND SWAP, in which case the change made by COMPARE AND SWAP is lost.
- 7. For the case of a condition-code setting of 1, COMPARE AND SWAP and COMPARE DOUBLE AND SWAP may or may not, depending on the model, cause any of the following to occur for the second-operand location: a PER storage-alteration event may be recognized; a protection exception for storing may be recognized; and, provided no access exception exists, the change bit may be set to one. Because the contents of storage remain unchanged, the change bit may or may not be one when a PER storage-alteration event is recognized.
- 8. The performance of CDSG on some models may be significantly slower than that of CSG. When quadword consistency is not required by the program, alternate code sequences should be used.

COMPARE AND SWAP AND STORE

CSST $D_1(B_1), D_2(B_2), R_3$ [SSF]



The first operand is compared with the third operand. If they are equal, the replacement value is stored into the first-operand location, and subsequently, the store value is placed into the second-operand location. If they are not equal, then the first operand is loaded into the third-operand location, and the second operand is not changed. The result is indicated by the condition code.

The size of the first and third operands, the size of the replacement value, and the alignment of the first operand are determined by the contents of a function code in bits 56-63 of general register 0. The assigned function codes are as follows:

- When the function code is 0, the operands are 32 bits in length. The first operand is a word in storage, and the third operand is in bits 32-63 of general register R_3 . The replacement value is in bytes 0-3 of the parameter list.
- When the function code is 1, the operands are 64 bits in length. The first operand is a doubleword in storage, and the third operand is in bits 0-63 of general register R_3 . The replacement value is in bytes 0-7 of the parameter list.
- When the function code is 2 and the compare-and-swap-and-store facility 2 is installed, the operands are 128 bits in length. The first operand is a quadword in storage, and the third operand is in bits 0-63 of general registers R_3 and $R_3 + 1$. The replacement value is in bytes 0-15 of the parameter list.

The size of the store value and the alignment of the second operand are determined by a store characteristic (SC) in bits 48-55 of general register 0. The store characteristic is expressed as a power of two; the assigned store characteristics are 0, 1, 2, and 3. An SC of 0 means that one byte is stored on a byte boundary; an SC of 1 means that two bytes are stored on a halfword boundary, an SC of 2 means that four bytes are stored on a word boundary, and an SC of 3 means that eight bytes are stored on a doubleword boundary. When the CSST facility 2 is installed, an SC of 4 is also assigned; an SC of 4 means that sixteen bytes are stored on a quadword boundary. The store value begins at byte 16 of the parameter list.

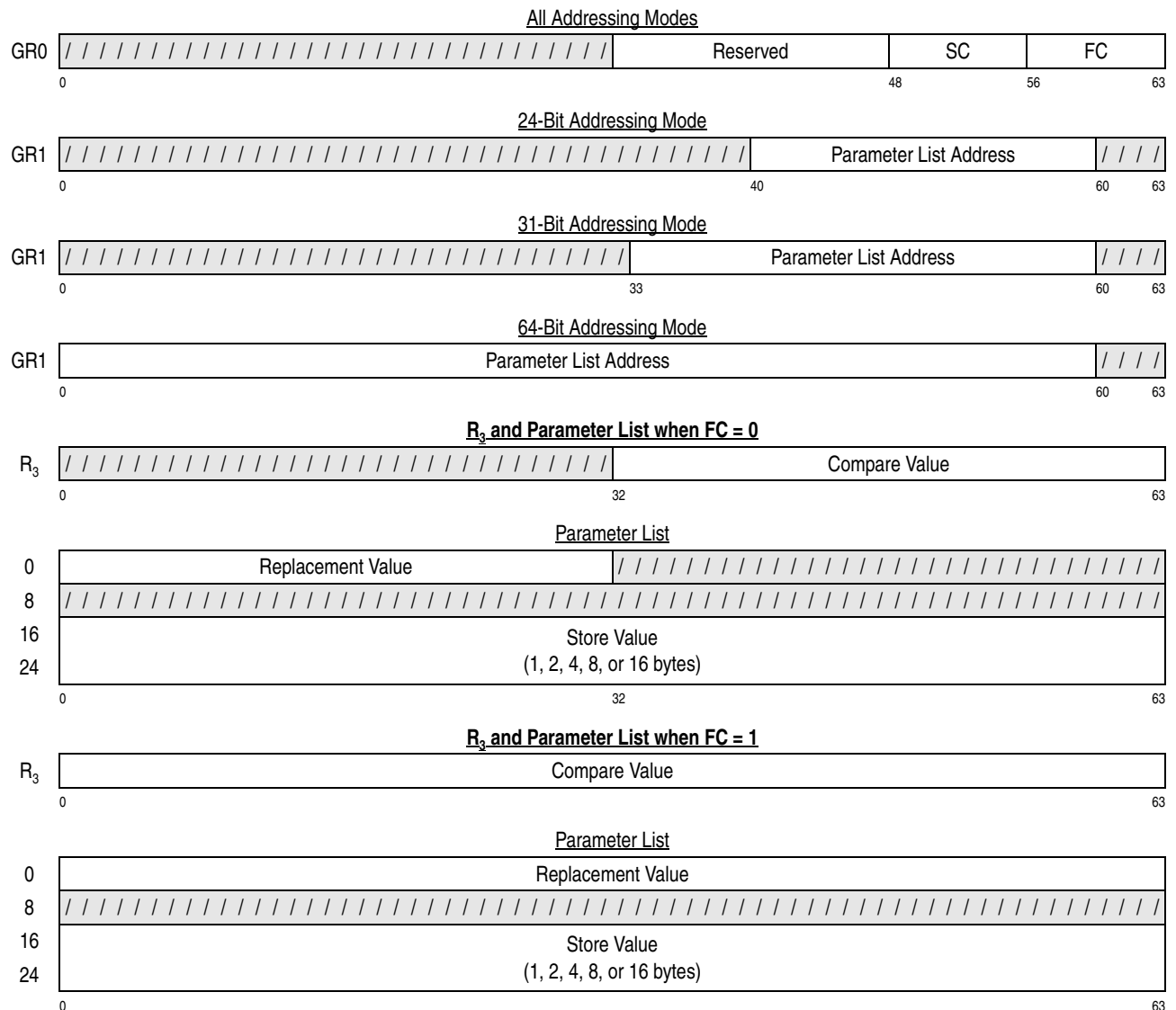
Bits 0-31 of general register 0 are ignored. Bits 32-47 of general register 0 and bits 60-63 of general register 1 are reserved and should contain zeros; other-

General register 1 contains the logical address of a parameter list that contains the replacement value and the store value. The parameter list is on a quad-word boundary. In the access-register mode, access register 1 specifies the address space containing the parameter list.

The handling of the parameter-list address is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-59 of

general register 1, with four zeros appended to the right, constitute the address, and bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-59 of the register, with four zeros appended to the right, constitute the address, and bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-59 of the register, with four zeros appended to the right, constitute the address.

The contents of the registers and parameter list just described are shown below.



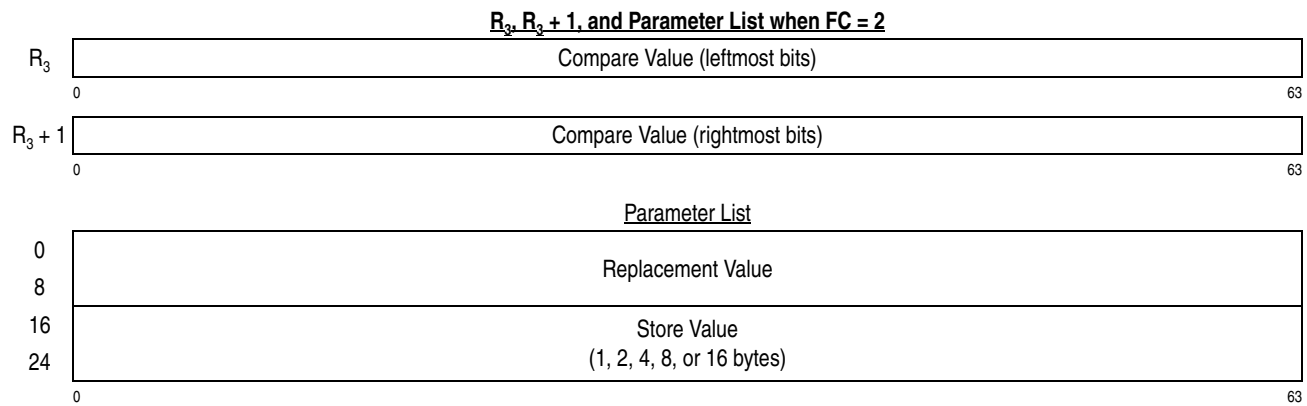


Figure 7-177. Register Contents for COMPARE AND SWAP AND STORE

When an equal comparison occurs, the replacement value is stored at the first-operand location, and the store value is stored at the second-operand location. The fetch of the first operand for purposes of comparison, and the store of the replacement value into the first-operand location, both appear to be a block-concurrent interlocked-update reference as observed by other CPUs. The store of the store value appears to be block-concurrent as observed by other CPUs.

When the result of the comparison is unequal, the first operand is loaded into the third-operand location, and the first operand remains unchanged. However, on some models, the contents of the first operand may be fetched and subsequently stored back unchanged at the first-operand location. This update appears to be a block-concurrent interlocked-update reference as observed by other CPUs.

As observed by this CPU and by other CPUs, all fetches appear to occur before all stores, and the store into the first operand appears to occur before the store into the second operand. Access-exception conditions are recognized for the entire 32-byte parameter list and for the second operand, regardless of whether the first and third operands are or are not equal. For the second operand, a PER storage-alteration event is recognized, and a change bit is set, only if a store occurs.

A serialization function is performed before the operation begins and again after the operation is completed.

Special Conditions

A specification exception is recognized for any of the following conditions:

- The function code specifies an unassigned value.
- The store characteristic specifies an unassigned value.
- The function code is 0, and the first operand is not designated on a word boundary.
- The function code is 1, and the first operand is not designated on a doubleword boundary.
- The function code is 2, and any of the following is true:
 - The compare-and-swap-and-store facility 2 is not installed.
 - The first operand is not designated on a quadword boundary.
 - The R₃ field does not designate the even-numbered register of an even-odd register pair.
- The second operand is not designated on an integral boundary corresponding to the size of the store value.

For all of the above conditions, and for all addressing and protection exceptions, the operation is suppressed.

Resulting Condition Code:

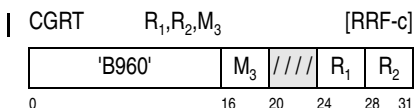
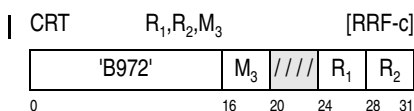
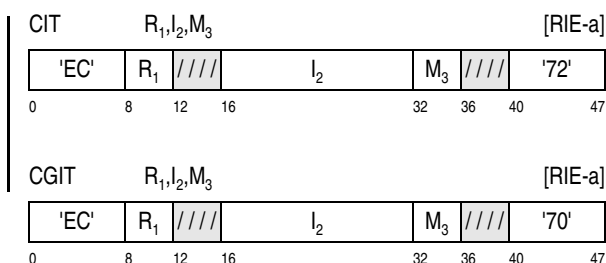
- 0 First and third operands equal; first operand replaced by replacement value, and second operand replaced by store value
- 1 First and third operands unequal; third operand replaced by first operand
- 2 --
- 3 --

Program Exceptions:

- Access (fetch, parameter list; fetch and store, operand 1; store, operand 2)
- Operation (if the compare-and-swap-and-store facility is not installed)
- Specification

Programming Notes:

1. COMPARE AND SWAP AND STORE may be used in conjunction with other COMPARE AND SWAP or COMPARE DOUBLE AND SWAP instructions to manipulate locks, queue pointers, or other fields that require interlocked updates.
2. COMPARE AND SWAP AND STORE should not be used to manipulate fields that are also manipulated by PERFORM LOCKED OPERATION.
3. The store value is intended to provide a separate “footprint” of the interlocked-update operation in a location apart from the first operand in a single unit of operation.
4. The performance of COMPARE AND SWAP AND STORE may be significantly slower than the that of separate COMPARE AND SWAP, BRANCH ON CONDITION, and STORE instructions.
5. COMPARE AND SWAP AND STORE should only be used when an interruption between the compare-and-swap operation and the store operation cannot be tolerated, and other means of disabling for interruptions are not practical.

COMPARE AND TRAP**COMPARE IMMEDIATE AND TRAP**

The first operand is compared with the second operand. If the mask bit in the M_3 field corresponding to the comparison result is one, a compare-and-trap-instruction data exception is recognized, and the operation is completed; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The comparison results and corresponding M_3 bits are as follows:

Comparison Result	M_3 Bit
Equal	0
First operand low	1
First operand high	2

Bit 3 of the M_3 field is reserved and should be zero; otherwise, the program may not operate compatibly in the future.

For COMPARE AND TRAP (CRT) and COMPARE IMMEDIATE AND TRAP (CIT), the first operand is treated as a 32-bit signed binary integer. For COMPARE AND TRAP (CGRT) and COMPARE IMMEDIATE AND TRAP (CGIT), the first operand is treated as a 64-bit signed binary integer.

For COMPARE IMMEDIATE AND TRAP, the second operand is treated as a 16-bit signed binary integer. For COMPARE AND TRAP (CRT), the second operand is treated as a 32-bit signed binary integer. For COMPARE AND TRAP (CGRT), the second operand is treated as a 64-bit signed binary integer.

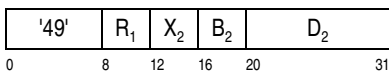
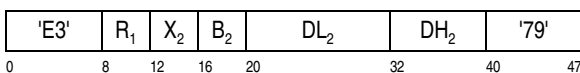
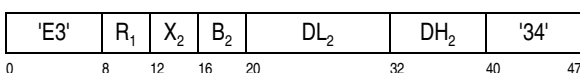
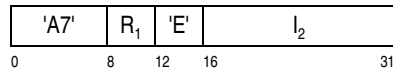
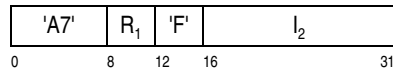
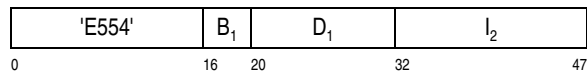
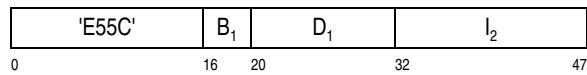
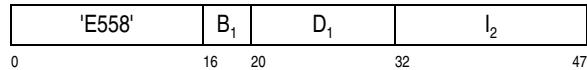
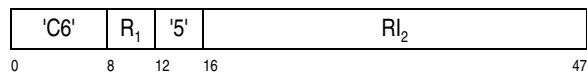
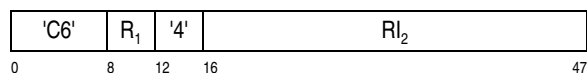
Condition Code: The code remains unchanged.

Program Exceptions:

- Data
- Operation (if the general-instructions-extension facility is not installed)

Programming Notes:

1. Possible uses of COMPARE AND TRAP and COMPARE IMMEDIATE AND TRAP include checking for pointers containing zero (null pointer), or array range checking.
2. On most models, when a data exception is not recognized, the performance of COMPARE AND TRAP and COMPARE IMMEDIATE AND TRAP is typically similar to other COMPARE instructions. When a data exception is recognized, the instruction may be significantly slower than an equivalent COMPARE instruction followed by a BRANCH ON CONDITION instruction.
3. When bit positions 0-2 of the M_3 field contain zeros, the instruction acts as a NOP, however this is not the preferred instruction with which to create a NOP. When bit positions 0-2 of the M_3 field contain 111 binary, a data exception is always recognized.
4. When a data exception is recognized, the compare-and-trap data-exception code (DXC FF hex) is stored, as described in "Data Exception" on page 6-21.
5. The discussion of extended mnemonics in programming note 3 for COMPARE AND BRANCH on page 7-114 also applies to the compare-and-trap instructions.

COMPARE HALFWORDCH $R_1, D_2(X_2, B_2)$ [RX-a]CHY $R_1, D_2(X_2, B_2)$ [RXY-a]CGH $R_1, D_2(X_2, B_2)$ [RXY-a]**COMPARE HALFWORD IMMEDIATE****Register-and-immediate formats:**CHI R_1, I_2 [RI-a]CGHI R_1, I_2 [RI-a]**Storage-and-immediate formats:**CHHSI $D_1(B_1), I_2$ [SIL]CHSI $D_1(B_1), I_2$ [SIL]CGHSI $D_1(B_1), I_2$ [SIL]**COMPARE HALFWORD RELATIVE LONG**CHRL R_1, RI_2 [RIL-b]CGHRL R_1, RI_2 [RIL-b]

The first operand is compared with the second operand, and the result is indicated in the condition code. The second operand is two bytes in length and is treated as a 16-bit signed binary integer.

For COMPARE HALFWORD IMMEDIATE (CHHSI), the first operand is treated as a 16-bit signed binary integer. For COMPARE HALFWORD (CH, CHY), COMPARE HALFWORD IMMEDIATE (CHI and

CHSI), and COMPARE HALFWORD RELATIVE LONG (CHRL), the first operand is treated as a 32-bit signed binary integer. For COMPARE HALFWORD (CGH), COMPARE HALFWORD IMMEDIATE (CGHI and CGHSI), and COMPARE HALFWORD RELATIVE LONG (CGHRL), the first operand is treated as a 64-bit signed binary integer.

The displacement for COMPARE HALFWORD (CH) and for COMPARE HALFWORD IMMEDIATE (CGHSI, CHHSI, CHSI) is treated as a 12-bit unsigned binary integer. The displacement for COMPARE HALFWORD (CHY, CGH) is treated as a 20-bit signed binary integer.

For COMPARE HALFWORD RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 --

Program Exceptions:

- Access (fetch, operand 1 of CGHSI, CHHSI, and CHSI only; fetch, operand 2 of CGH, CGHRL, CH, CHRL, CHY only)
- Operation (CHY, if the long-displacement facility is not installed; CGH, CHRL, CGHRL, CGHSI, CHHSI, and CHSI, if the general-instructions-extension facility is not installed)

Programming Notes:

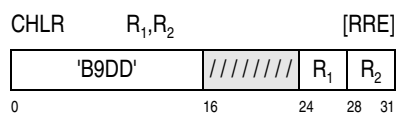
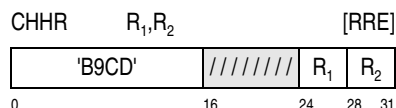
- An example of the use of the COMPARE HALFWORD instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
- For COMPARE HALFWORD RELATIVE LONG, the second operand is necessarily aligned on an

integral boundary corresponding to the operand's size.

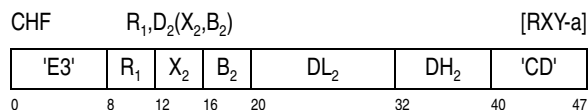
- When COMPARE HALFWORD RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.

COMPARE HIGH

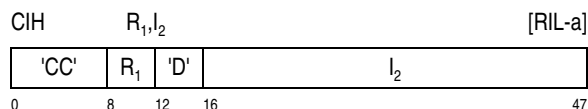
Register-and-register formats:



Register-and-storage format:



COMPARE IMMEDIATE HIGH



The first operand is compared with the second operand, and the result is indicated in the condition code.

The operands are treated as 32-bit signed binary integers. The first operand is in bit positions 0-31 of general register R_1 ; bit positions 32-63 of the register are ignored. For COMPARE HIGH (CHHR), the second operand is in bit positions 0-31 of general register R_2 ; bit positions 32-63 of the register are ignored. For COMPARE HIGH (CHLR), the second operand is in bit positions 32-63 of general register R_2 ; bit positions 0-31 of the register are ignored.

The displacement for CHF is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 --

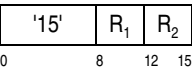
Program Exceptions:

- Access (operand 2 of CHF only)
- Operation (if the high-word facility is not installed)

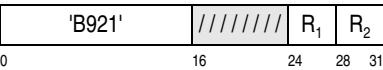
COMPARE LOGICAL

Register-and-register formats:

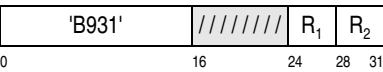
CLR R₁,R₂ [RR]



CLGR R₁,R₂ [RRE]

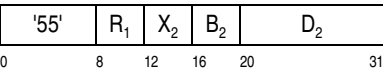


CLGFR R₁,R₂ [RRE]

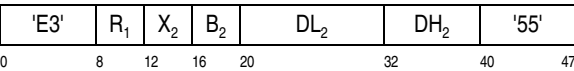


Register-and-storage formats:

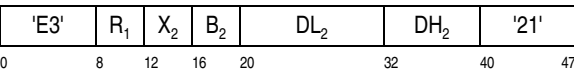
CL R₁,D₂(X₂,B₂) [RX-a]



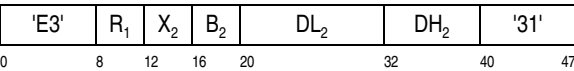
CLY R₁,D₂(X₂,B₂) [RXY-a]



CLG R₁,D₂(X₂,B₂) [RXY-a]

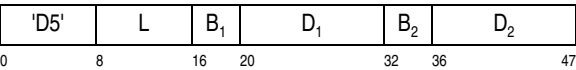


CLGF R₁,D₂(X₂,B₂) [RXY-a]



Storage-and-storage format:

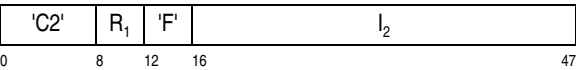
CLC D₁(L,B₁),D₂(B₂) [SS-a]



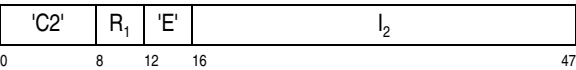
COMPARE LOGICAL IMMEDIATE

Register-and-immediate formats:

CLFI R₁,I₂ [RIL-a]

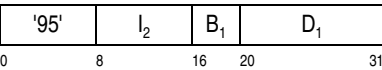


CLGFI R₁,I₂ [RIL-a]

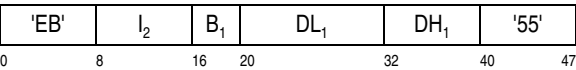


Storage-and-immediate formats:

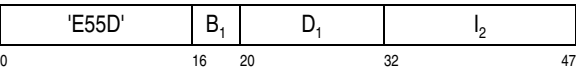
CLI D₁(B₁),I₂ [SI]



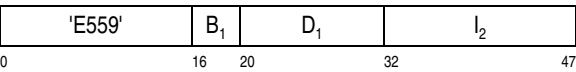
CLIIY D₁(B₁),I₂ [SIY]



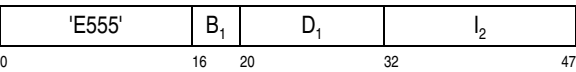
CLFHSI D₁(B₁),I₂ [SIL]



CLGHSI D₁(B₁),I₂ [SIL]

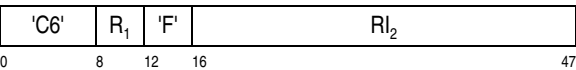


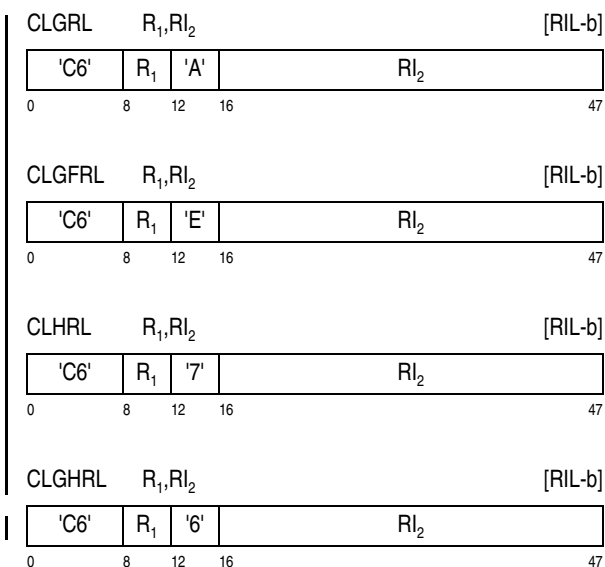
CLHHSI D₁(B₁),I₂ [SIL]



COMPARE LOGICAL RELATIVE LONG

CLRL R₁,R₁I₂ [RIL-b]





The first operand is compared with the second operand, and the result is indicated in the condition code.

For COMPARE LOGICAL (CLR, CL, CLY), COMPARE LOGICAL IMMEDIATE (CLFI), and COMPARE LOGICAL RELATIVE LONG (CLRL), the operands are treated as 32 bits. For COMPARE LOGICAL (CLGR, CLG) and COMPARE LOGICAL RELATIVE LONG (CLGRL), the operands are treated as 64 bits. For COMPARE LOGICAL (CLGFR, CLGF), COMPARE LOGICAL IMMEDIATE (CLGFI), and COMPARE LOGICAL RELATIVE LONG (CLGFRL), the first operand is treated as 64 bits, and the second operand is treated as 32 bits with 32 zeros appended on the left. For COMPARE LOGICAL IMMEDIATE (CLHHSI), the operands are treated as 16 bits. For COMPARE LOGICAL IMMEDIATE (CLFHHSI) and COMPARE LOGICAL RELATIVE LONG (CLHRL), the first operand is treated as 32 bits, and the second operand is treated as 16 bits with 16 zeros appended on the left. For COMPARE LOGICAL IMMEDIATE (CLGHSI) and COMPARE LOGICAL RELATIVE LONG (CLGHRL), the first operand is treated as 64 bits, and the second operand is treated as 16 bits with 48 zeros appended on the left. For COMPARE LOGICAL IMMEDIATE (CLI, CLIY), the operands are treated as 8 bits.

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the fields is reached. For COMPARE LOGICAL (CL, CLY, CLG, CLGF, CLC), COMPARE LOGICAL IMMEDIATE (CLFHHSI, CLGHSI, CLHHSI) and COMPARE LOGICAL RELATIVE LONG, access excep-

tions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte.

The displacements for CL, CLI, CLFHHSI, CLGHSI, CLHHSI, and both operands of CLC are treated as 12-bit unsigned binary integers. The displacement for CLY, CLG, CLGF, and CLIY is treated as a 20-bit signed binary integer.

For COMPARE LOGICAL RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

For COMPARE LOGICAL RELATIVE LONG (CLRL, CLGFRL), the second operand must be aligned on a word boundary, and for COMPARE LOGICAL RELATIVE LONG (CLGRL), the second operand must be aligned on a doubleword boundary; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 --

Program Exceptions:

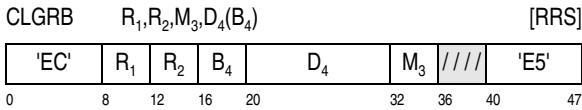
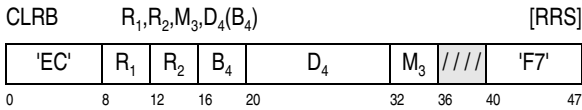
- Access (fetch, operand 2, CL, CLC, CLG, CLGF, CLGFRL, CLGHRL, CLGRL, CLHRL, CLRL, and CLY; fetch, operand 1, CLC, CLFHHSI, CLGHSI, CLHHSI, CLI, and CLIY)
- Operation (CLY and CLIY, if the long-displacement facility is not installed; CLFI and CLGFI, if the extended-immediate facility is not installed; CLFHHSI, CLGFRL, CLGHSI, CLGHRL, CLGRL, CLHHSI, CLHRL, and CLRL, if the general-instructions-extension facility is not installed)
- Specification (CLRL, CLGFRL, CLGRL)

Programming Notes:

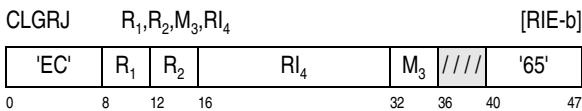
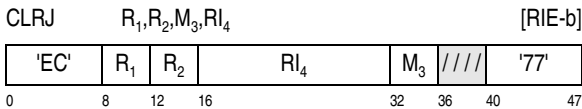
1. Examples of the use of the COMPARE LOGICAL instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."

- COMPARE LOGICAL treats all bits of each operand alike as part of a field of unstructured logical data. For COMPARE LOGICAL (CLC), the comparison may extend to field lengths of 256 bytes.
- For COMPARE LOGICAL RELATIVE LONG, the second operand must be aligned on an integral boundary corresponding to the operand's size.
- When COMPARE LOGICAL RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.

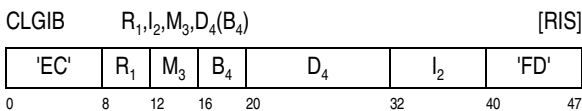
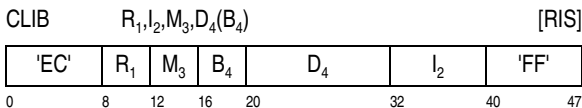
COMPARE LOGICAL AND BRANCH



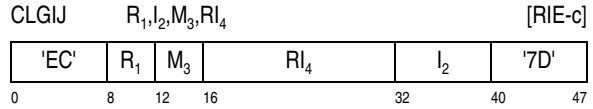
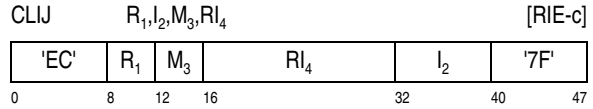
COMPARE LOGICAL AND BRANCH RELATIVE



COMPARE LOGICAL IMMEDIATE AND BRANCH



COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE



The first operand is compared with the second operand. If the mask bit in the M_3 field corresponding to the comparison result is one, the instruction address in the current PSW is replaced by the branch address specified by the fourth operand; otherwise, normal instruction sequencing proceeds with the updated instruction address.

For COMPARE LOGICAL AND BRANCH (CLRB), COMPARE LOGICAL AND BRANCH RELATIVE (CLRJ), COMPARE LOGICAL IMMEDIATE AND BRANCH (CLIB), and COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (CLIJ), the first operand is treated as a 32-bit unsigned binary integer. For COMPARE LOGICAL AND BRANCH (CLGRB), COMPARE LOGICAL AND BRANCH RELATIVE (CLGRJ), COMPARE LOGICAL IMMEDIATE AND BRANCH (CLGIB), and COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (CLGIJ), the first operand is treated as a 64-bit unsigned binary integer.

For COMPARE LOGICAL AND BRANCH (CLRB) and COMPARE LOGICAL AND BRANCH RELATIVE (CLRJ), the second operand is treated as a 32-bit unsigned binary integer. For COMPARE LOGICAL AND BRANCH (CLGRB) and COMPARE LOGICAL AND BRANCH RELATIVE (CLGRJ), the second operand is treated as a 64-bit unsigned binary integer. For COMPARE LOGICAL IMMEDIATE AND BRANCH and COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE, the second operand is treated as an 8-bit unsigned binary integer. When the size of the second operand is smaller than the size of the first operand, the second operand is extended on the left with zeros to match the size of the first operand.

The comparison results and corresponding M_3 bits are as follows:

Comparison Result	M_3 Bit
Equal	0
First operand low	1
First operand high	2

Bit 3 of the M_3 field is reserved and should be zero; otherwise, the program may not operate compatibly in the future.

For COMPARE LOGICAL AND BRANCH and COMPARE LOGICAL IMMEDIATE AND BRANCH, the fourth operand is used as the branch address. For COMPARE LOGICAL AND BRANCH RELATIVE and COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE, the contents of the I_4 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the branch address.

Condition Code: The code remains unchanged.

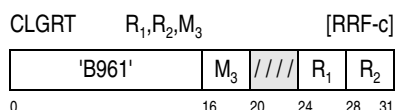
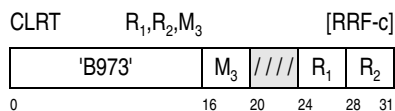
Program Exceptions:

- Operation (if the general-instructions-extension facility is not installed).

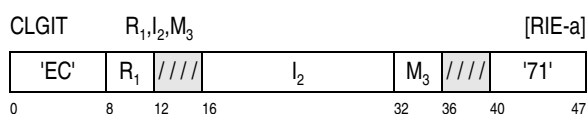
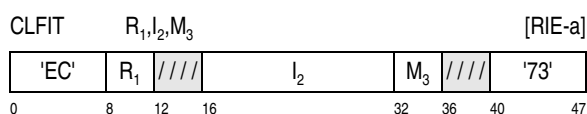
Programming Notes:

1. When bit positions 0-2 of the M_3 field contain zeros, the instruction acts as a NOP, however this is not the preferred instruction with which to create a NOP. When bit positions 0-2 of the M_3 field contain 111 binary, a branch always occurs.
2. When COMPARE LOGICAL AND BRANCH RELATIVE or COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE are the target of an execute-type instruction, the branch is relative to the target address. See "Branch-Address Generation" on page 5-10.
3. The discussion of extended mnemonics in programming note 3 for COMPARE AND BRANCH on page 7-114 also applies to the compare-logical-and-branch instructions.

COMPARE LOGICAL AND TRAP



COMPARE LOGICAL IMMEDIATE AND TRAP



The first operand is compared with the second operand. If the mask bit in the M_3 field corresponding to the comparison result is one, a compare-and-trap-instruction data exception is recognized, and the operation is completed; otherwise, normal instruction sequencing proceeds with the updated instruction address.

The comparison results and corresponding M_3 bits are as follows:

Comparison Result	M_3 Bit
Equal	0
First operand low	1
First operand high	2

Bit 3 of the M_3 field is reserved and should be zero; otherwise, the program may not operate compatibly in the future.

For COMPARE LOGICAL AND TRAP (CLRT) and COMPARE LOGICAL IMMEDIATE AND TRAP (CLFIT), the first operand is treated as a 32-bit unsigned binary integer. For COMPARE LOGICAL AND TRAP (CLGRT) and COMPARE LOGICAL IMMEDIATE AND TRAP (CLGIT), the first operand is treated as a 64-bit unsigned binary integer.

For COMPARE LOGICAL IMMEDIATE AND TRAP, the second operand is treated as a 16-bit unsigned binary integer. For COMPARE LOGICAL AND TRAP (CLRT), the second operand is treated as a 32-bit unsigned binary integer. For COMPARE LOGICAL AND TRAP (CLGRT), the second operand is treated as a 64-bit unsigned binary integer. When the size of the second operand is smaller than the size of the first operand, the second operand is extended on the left with zeros to match the size of the first operand.

Condition Code: The code remains unchanged.

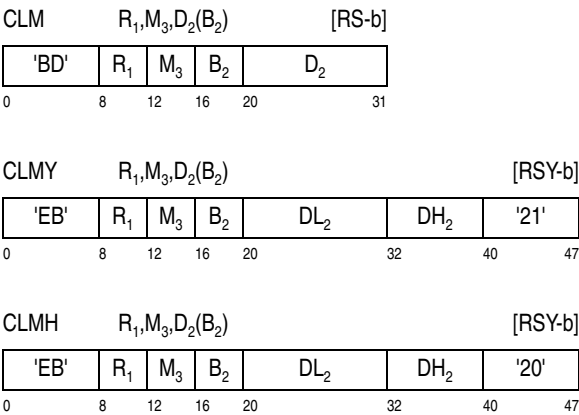
Program Exceptions:

- Data
- Operation (if the general-instructions-extension facility is not installed)

Programming Notes:

1. Possible uses of COMPARE LOGICAL AND TRAP and COMPARE LOGICAL IMMEDIATE AND TRAP include checking for pointers containing zero (null pointer), or array range checking.
2. On most models, when a data exception is not recognized, the performance of COMPARE LOGICAL AND TRAP and COMPARE LOGICAL IMMEDIATE AND TRAP is typically similar to other COMPARE LOGICAL instructions. When a data exception is recognized, the instruction may be significantly slower than an equivalent COMPARE LOGICAL instruction followed by a BRANCH ON CONDITION instruction.
3. When bit positions 0-2 of the M_3 field contain zeros, the instruction acts as a NOP, however this is not the preferred instruction with which to create a NOP. When bit positions 0-2 of the M_3 field contain 111 binary, a data exception is always recognized.
4. When a data exception is recognized, the compare-and-trap data-exception code (DXC FF hex) is stored, as described in “Data Exception” on page 6-21.
5. The discussion of extended mnemonics in programming note 3 for COMPARE AND BRANCH on page 7-114 also applies to the compare-logical-and-trap instructions.

COMPARE LOGICAL CHARACTERS UNDER MASK



The first operand is compared with the second operand and under control of a mask, and the result is indicated in the condition code.

The contents of the M_3 field are used as a mask. These four bits, left to right, correspond one for one with four bytes, left to right, of general register R_1 . For COMPARE LOGICAL CHARACTERS UNDER MASK (CLM, CLMY), the four bytes to which the mask bits correspond are in bit positions 32-63 of general register R_1 . For COMPARE LOGICAL CHARACTERS UNDER MASK (CLMH), the four bytes are in the high-order half, bit positions 0-31, of the register. The byte positions corresponding to ones in the mask are considered as a contiguous field and are compared with the second operand. The second operand is a contiguous field in storage, starting at the second-operand address and equal in length to the number of ones in the mask. The bytes in the general register corresponding to zeros in the mask do not participate in the operation.

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the fields is reached.

When the mask is not zero, exceptions associated with storage-operand access are recognized for no more than the number of bytes specified by the mask. Access exceptions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte. When the mask is zero, access exceptions are recognized for one byte at the second-operand address.

The displacement for CLM is treated as a 12-bit unsigned binary integer. The displacement for CLMY and CLMH is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 Operands equal, or mask bits all zeros
- 1 First operand low
- 2 First operand high
- 3 --

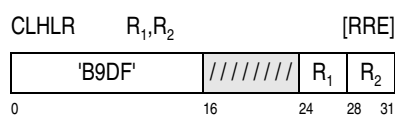
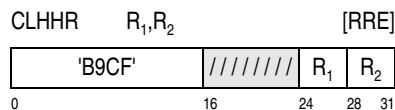
Program Exceptions:

- Access (fetch, operand 2)
- Operation (CLMY, if the long-displacement facility is not installed)

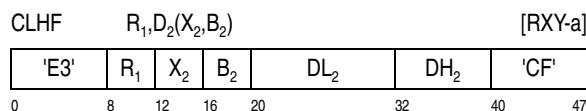
Programming Note: An example of the use of the COMPARE LOGICAL CHARACTERS UNDER MASK instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”

COMPARE LOGICAL HIGH

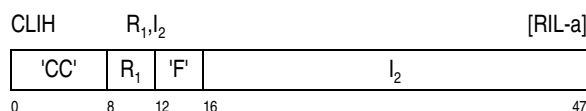
Register-and-register formats:



Register-and-storage format:



COMPARE LOGICAL IMMEDIATE HIGH



The first operand is compared with the second operand, and the result is indicated in the condition code.

The operands are treated as 32-bit unsigned binary integers. The first operand is in bit positions 0-31 of general register R_1 ; bit positions 32-63 of the register are ignored. For COMPARE LOGICAL HIGH (CLHHR), the second operand is in bit positions 0-31 of general register R_2 ; bit positions 32-63 of the register are ignored. For COMPARE LOGICAL HIGH (CLHLR), the second operand is in bit positions 32-63 of general register R_2 ; bit positions 0-31 of the register are ignored.

The displacement for CLHF is treated as a 20-bit signed binary integer.

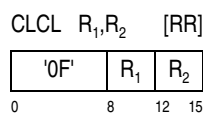
Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 --

Program Exceptions:

- Access (operand 2 of CLHF only)
- Operation (if the high-word facility is not installed)

COMPARE LOGICAL LONG



The first operand is compared with the second operand, and the result is indicated in the condition code. The shorter operand is considered to be extended on the right with padding bytes.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers R_1 and R_2 , respectively. The number of bytes in the first-operand and second-operand locations is specified by unsigned binary integers in bit positions 40-63 of general registers $R_1 + 1$ and $R_2 + 1$, respectively. Bit positions 32-39 of general

register $R_2 + 1$ contain the padding byte. The contents of bit positions 0-39 of general register $R_1 + 1$ and of bit positions 0-31 of general register $R_2 + 1$ are ignored.

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 consti-

tute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-178 on page 7-136.

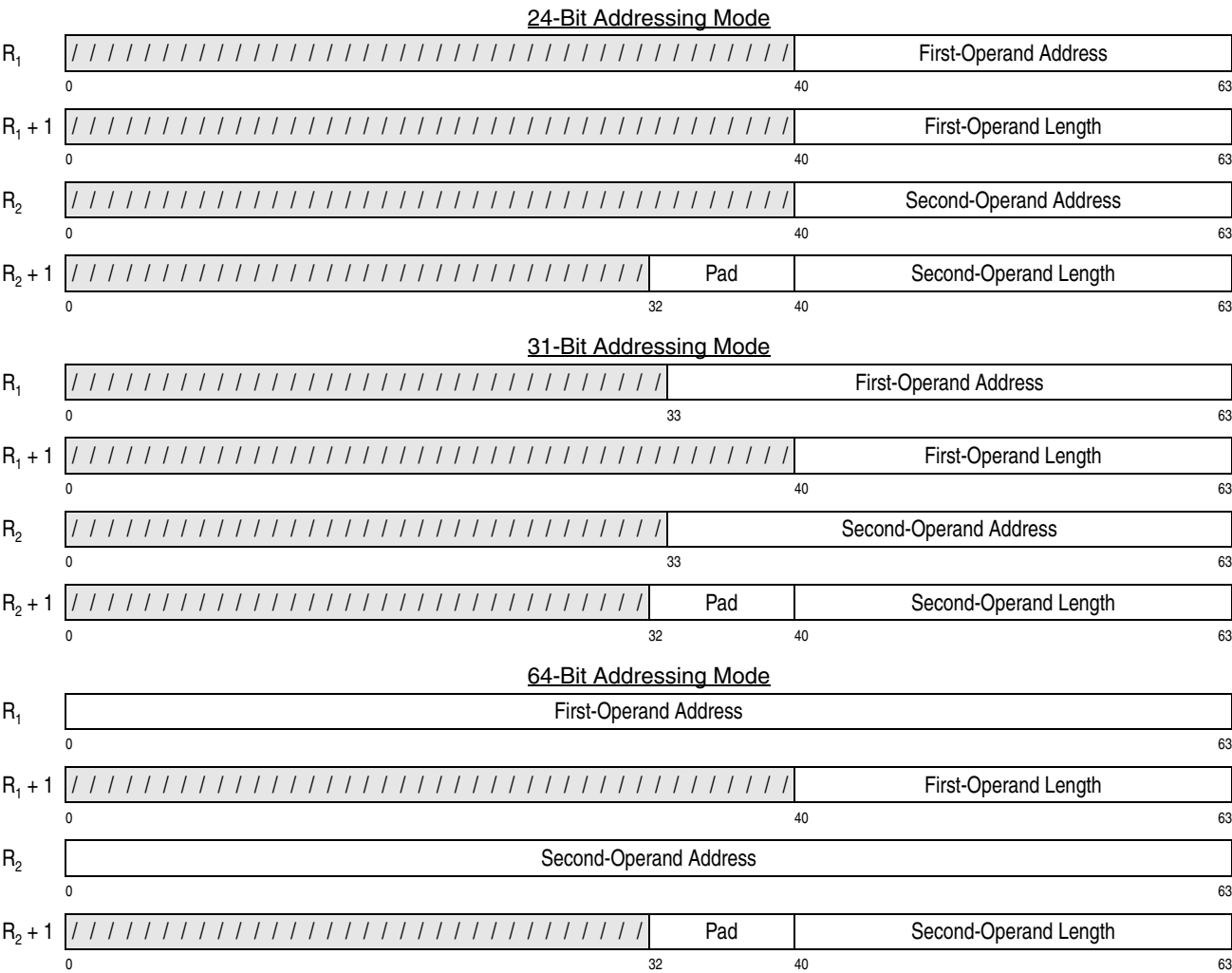


Figure 7-178. Register Contents for COMPARE LOGICAL LONG

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the longer operand is reached. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of padding bytes.

If both operands are of zero length, the operands are considered to be equal.

The execution of the instruction is interruptible. When an interruption occurs, other than one that follows termination, the lengths in general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes compared, and the addresses in general registers R_1 and

R_2 are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. In the 24-bit or 31-bit addressing mode, the leftmost bits which are not part of the address in bit positions 32-63 of general registers R_1 and R_2 are set to zeros, and the contents of bit positions 0-31 remain unchanged. In any addressing mode, the contents of bit positions 0-39 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged, and the condition code is unpredictable. If the operation is interrupted after the shorter operand has been exhausted, the length field pertaining to the shorter operand is zero, and its address is updated accordingly.

If the operation ends because of an inequality, the address fields in general registers R_1 and R_2 at completion identify the first unequal byte in each operand. The lengths in bit positions 40-63 of general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes that were equal, unless the inequality occurred with the padding byte, in which case the length field for the shorter operand is set to zero. The addresses in general registers R_1 and R_2 are incremented by the amounts by which the corresponding length fields were reduced.

If the two operands, including the padding byte, if necessary, are equal, both length fields are made zero at completion, and the addresses are incremented by the corresponding operand-length values.

At the completion of the operation, in the 24-bit or 31-bit addressing mode, the leftmost bits which are not part of the address in bit positions 32-63 of general registers R_1 and R_2 are set to zeros, even when one or both of the initial length values are zero. In any addressing mode, the contents of bit positions 0-39 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged.

Access exceptions for the portion of a storage operand to the right of the first unequal byte may or may not be recognized. For operands longer than 2K bytes, access exceptions are not recognized more than 2K bytes beyond the byte being processed. Access exceptions are not indicated for locations more than 2K bytes beyond the first unequal byte.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

- 0 Operands equal, or both zero length
- 1 First operand low
- 2 First operand high
- 3 --

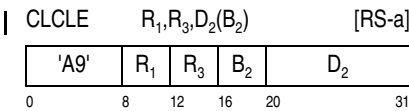
Program Exceptions:

- Access (fetch, operands 1 and 2)
- Specification

Programming Notes:

1. An example of the use of the COMPARE LOGICAL LONG instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When the R_1 and R_2 fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, except that the contents of the designated registers are incremented or decremented only by the number of bytes compared, not by twice the number of bytes compared. In the absence of dynamic modification of the operand area by another CPU or by a channel program, condition code 0 is set. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed.
3. Special precautions should be taken when COMPARE LOGICAL LONG is made the target of an execute-type instruction. See the programming note concerning interruptible instructions under EXECUTE.
4. Other programming notes concerning interruptible instructions are included in "Interruptible Instructions" in Chapter 5, "Program Execution."
5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

COMPARE LOGICAL LONG EXTENDED



The first operand is compared with the third operand until unequal bytes are compared, the end of the longer operand is reached, or a CPU-determined number of bytes have been compared, whichever occurs first. The shorter operand is considered to be extended on the right with padding bytes. The result is indicated in the condition code.

The R₁ and R₃ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and third operand is designated by the contents of general registers R₁ and R₃, respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 32-63 of general registers R₁ + 1 and R₃ + 1, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and third-operand locations is specified by the entire contents of general registers R₁ + 1 and R₃ + 1, respectively, and those contents are treated as 64-bit unsigned binary integers.

The handling of the addresses in general registers R₁ and R₃ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R₁ and R₃ constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The second-operand address is not used to address data; instead, the rightmost eight bits of the second-operand address, bits 56-63, are the padding byte. Bits 0-55 of the second-operand address are ignored.

The contents of the registers and address just described are shown in Figure 7-179 on page 7-139.

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found, the end of the longer operand is reached, or a CPU-determined number of bytes have been compared, whichever occurs first. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of padding bytes.

If both operands are of zero length, the operands are considered to be equal.

If the operation ends because of an inequality, the address fields in general registers R₁ and R₃ at completion identify the first unequal byte in each operand. The lengths in bit positions 32-63, in the 24-bit or 31-bit addressing mode, or in bit positions 0-63, in the 64-bit addressing mode, of general registers R₁ + 1 and R₃ + 1 are decremented by the number of bytes that were equal, unless the inequality occurred with the padding byte, in which case the length field for the shorter operand is set to zero. The addresses in general registers R₁ and R₃ are incremented by the amounts by which the corresponding length fields were decremented. Condition code 1 is set if the first operand is low, or condition code 2 is set if the first operand is high.

If the two operands, including the padding byte, if necessary, are equal, both length fields are made zero at completion, and the addresses are incremented by the corresponding operand-length values. Condition code 0 is set.

If the operation is completed because a CPU-determined number of bytes have been compared without finding an inequality or reaching the end of the longer operand, the lengths in general registers R₁ + 1 and R₃ + 1 are decremented by the number of bytes compared, and the addresses in general registers R₁ and R₃ are incremented by the same number, so that the instruction, when reexecuted, resumes at the next bytes to be compared. If the operation is completed after the shorter operand has been exhausted, the length field pertaining to the shorter operand is zero, and the operand address is updated accordingly. Condition code 3 is set.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R₁, R₁ + 1, R₃, and R₃ + 1, always remain unchanged.

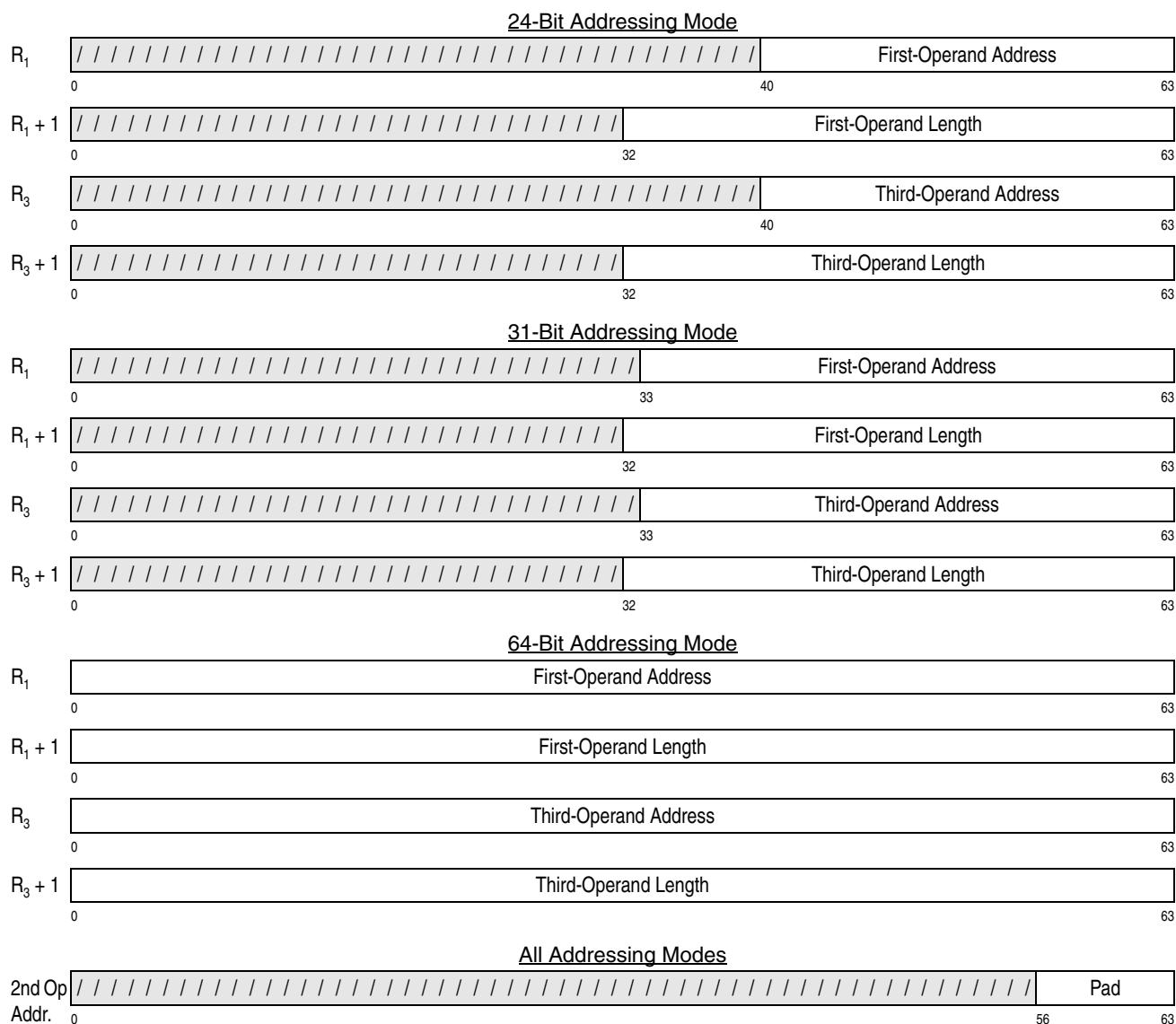


Figure 7-179. Register Contents and Second-Operand Address for COMPARE LOGICAL LONG EXTENDED

The padding byte may be formed from $D_2(B_2)$ multiple times during the execution of the instruction, and the registers designated by R_1 and R_3 may be updated multiple times. Therefore, if B_2 equals R_1 , $R_1 + 1$, R_3 , or $R_3 + 1$ and is subject to change during the execution of the instruction, the results are unpredictable.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed. The maximum amount is approximately 4K bytes of either operand.

At the completion of the operation in the 24-bit or 31-bit addressing mode, the leftmost bits which are not part of the address in bit positions 32-63 of general registers R_1 and R_3 may be set to zeros or may remain unchanged from their original values, even when one or both of the initial length values are zero.

Access exceptions for the portion of a storage operand to the right of the first unequal byte may or may not be recognized. For operands longer than 4K bytes, access exceptions are not recognized more than 4K bytes beyond the byte being processed. Access exceptions are not indicated for locations more than 4K bytes beyond the first unequal byte.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

- 0 All bytes compared; operands equal, or both zero length
- 1 All bytes compared, first operand low
- 2 All bytes compared, first operand high
- 3 CPU-determined number of bytes compared without finding an inequality

Program Exceptions:

- Access (fetch, operands 1 and 3)
- Specification

Programming Notes:

- COMPARE LOGICAL LONG EXTENDED is intended for use in place of COMPARE LOGICAL LONG when the operand lengths are specified as 32-bit or 64-bit binary integers. COMPARE LOGICAL LONG EXTENDED sets condition code 3 in cases in which COMPARE LOGICAL LONG would be interrupted.
- When condition code 3 is set, the program can simply branch back to the instruction to continue the comparison. The program need not determine the number of bytes that were compared.
- The function of not processing more than approximately 4K bytes of either operand is intended to permit software polling of a flag that may be set by a program on another CPU during long operations.
- When the R₁ and R₃ fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, except that the contents of the designated registers are incremented or decremented only by the number of bytes compared, not by twice the number of bytes compared. In the absence of dynamic modification of the operand area by another CPU or by a channel program, the condition code is finally set to 0 after possible settings to 3. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed. If storage

is not accessed, condition code 3 may or may not be set regardless of the operand length.

- In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

COMPARE LOGICAL LONG UNICODE



The first operand is compared with the third operand until unequal two-byte Unicode characters are compared, the end of the longer operand is reached, or a CPU-determined number of characters have been compared, whichever occurs first. The shorter operand is considered to be extended on the right with two-byte padding characters. The result is indicated in the condition code.

The R₁ and R₃ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost character of the first operand and third operand is designated by the contents of general registers R₁ and R₃, respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 32-63 of general registers R₁ + 1 and R₃ + 1, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 0-63 of general registers R₁ + 1 and R₃ + 1, respectively, and those contents are treated as 64-bit unsigned binary integers.

The contents of general registers R₁ + 1 and R₃ + 1 must specify an even number of bytes; otherwise, a specification exception is recognized.

The handling of the addresses in general registers R₁ and R₃ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R₁ and R₃ consti-

tute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of the registers constitute the address.

The second-operand address is not used to address data; instead, the rightmost 16 bits of the second-operand address, bits 48-63, are the two-byte padding character. Bits 0-47 of the second-operand address are ignored.

The contents of the registers and address just described are shown in Figure 7-180 on page 7-141.

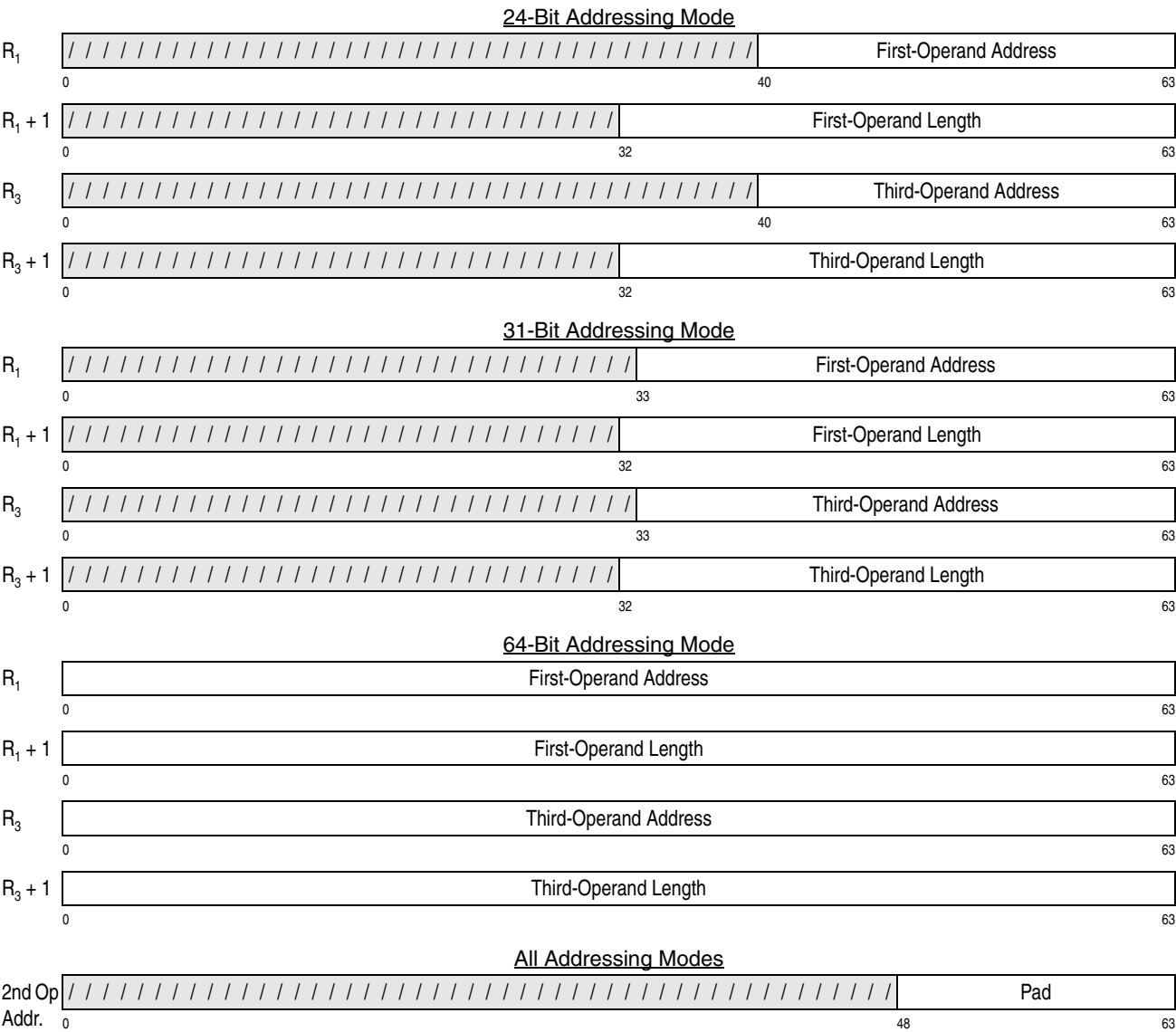


Figure 7-180. Register Contents and Second-Operand Address for COMPARE LOGICAL LONG UNICODE

The comparison proceeds left to right, character by character, and ends as soon as an inequality is found, the end of the longer operand is reached, or a CPU-determined number of characters have been compared, whichever occurs first. If the operands are not of the same length, the shorter operand is con-

sidered to be extended on the right with the appropriate number of two-byte padding characters.

If both operands are of zero length, the operands are considered to be equal.

If the operation ends because of an inequality, the address fields in general registers R_1 and R_3 at completion identify the first unequal two-byte character in each operand. The lengths in bit positions 32-63, in the 24-bit or 31-bit addressing mode, or in bit positions 0-63, in the 64-bit addressing mode, of general registers $R_1 + 1$ and $R_3 + 1$ are decremented by 2 times the number of characters that were equal, unless the inequality occurred with the two-byte padding character, in which case the length field for the shorter operand is set to zero. The addresses in general registers R_1 and R_3 are incremented by the amounts by which the corresponding length fields were decremented. Condition code 1 is set if the first operand is low, or condition code 2 is set if the first operand is high.

If the two operands, including the two-byte padding character, if necessary, are equal, both length fields are made zero at completion, and the addresses are incremented by the corresponding operand-length values. Condition code 0 is set.

If the operation is completed because a CPU-determined number of characters have been compared without finding an inequality or reaching the end of the longer operand, the lengths in general registers $R_1 + 1$ and $R_3 + 1$ are decremented by 2 times the number of characters compared, and the addresses in general registers R_1 and R_3 are incremented by the same number, so that the instruction, when reexecuted, resumes at the next characters to be compared. If the operation is completed after the shorter operand has been exhausted, the length field pertaining to the shorter operand is zero, and the operand address is updated accordingly. Condition code 3 is set.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, R_3 , and $R_3 + 1$, always remain unchanged.

The two-byte padding character may be formed from $D_2(B_2)$ multiple times during the execution of the instruction, and the registers designated by R_1 and R_3 may be updated multiple times. Therefore, if B_2 equals R_1 , $R_1 + 1$, R_3 , or $R_3 + 1$ and is subject to change during the execution of the instruction, the results are unpredictable.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may

be a different amount each time the instruction is executed.

At the completion of the operation in the 24-bit or 31-bit addressing mode, the leftmost bits which are not part of the address in bit positions 32-63 of general registers R_1 and R_3 may be set to zeros or may remain unchanged from their original values, including the case when one or both of the initial length values are zero.

Access exceptions for the portion of a storage operand to the right of the first unequal character may or may not be recognized. For operands longer than 4K bytes, access exceptions are not recognized more than 4K bytes beyond the character being processed. Access exceptions are not indicated for locations more than 4K bytes beyond the first unequal character.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field or length associated with that operand is odd.

Resulting Condition Code:

- 0 All characters compared; operands equal, or both zero length
- 1 First operand low
- 2 First operand high
- 3 CPU-determined number of characters compared without finding an inequality

Program Exceptions:

- Access (fetch, operands 1 and 3)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

Programming Notes:

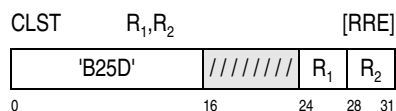
1. COMPARE LOGICAL LONG UNICODE is intended for use in place of COMPARE LOGICAL LONG or COMPARE LOGICAL LONG EXTENDED when two-byte characters are to be compared. The characters may be Unicode characters or any other double-byte characters. COMPARE LOGICAL LONG UNICODE sets condition code 3 in cases in which COMPARE LOGICAL LONG would be interrupted.

2. When condition code 3 is set, the program can simply branch back to the instruction to continue the comparison. The program need not determine the number of characters that were compared.
3. When the R_1 and R_3 fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, except that the contents of the designated registers are incremented or decremented only by 2 times the number of characters compared, not by 4 times the number of characters compared. In the absence of dynamic modification of the operand area by another CPU or by a channel program, the condition code is finally set to 0 after possible settings to 3. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed. If storage is not accessed, condition code 3 may or may not be set regardless of the operand length.
4. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
5. The padding character may be represented in the displacement field of the instruction. The following example illustrates padding with a Unicode space character.

CLCLU 6,8,X'020'

When the B_2 field of the instruction designates general register 0, and the long-displacement facility is not installed, the padding character is limited to a character whose representation is less than or equal to FFF hex.

COMPARE LOGICAL STRING



The first operand is compared with the second operand until unequal bytes are compared, the end of either operand is reached, or a CPU-determined number of bytes have been compared, whichever occurs first. The CPU-determined number is at least 256. The result is indicated in the condition code.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers R_1 and R_2 , respectively.

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The first and second operands may be of the same or different lengths. The end of an operand is indicated by an ending character in the last byte position of the operand. The ending character to be used to determine the end of an operand is specified in bit positions 56-63 of general register 0. Bit positions 32-55 of general register 0 are reserved for possible future extensions and must contain all zeros; otherwise, a specification exception is recognized.

The operation proceeds left to right, byte by byte, and ends as soon as the ending character is encountered in either or both operands, unequal bytes which do not include an ending character are compared, or a CPU-determined number of bytes have been compared, whichever occurs first. The CPU-determined number is at least 256. When the ending character is encountered simultaneously in both operands, including when it is in the first byte position of the operands, the operands are of the same length and are considered to be equal, and condition code 0 is set. When the ending character is encountered in only one operand, that operand, which is the shorter operand, is considered to be low, and condition code 1 or 2 is set. Condition code 1 is set if the first operand is low, or condition code 2 is set if the second operand is low. Similarly, when unequal bytes which do not include an ending character are compared, condition code 1 is set if the lower byte is in the first operand, or condition code 2 is set if the lower byte is in the second operand. When a CPU-determined number of bytes have been compared, condition code 3 is set.

When condition code 1 or 2 is set, the address of the last byte processed in the first and second operands is placed in general registers R_1 and R_2 , respectively. That is, when condition code 1 is set, the address of the ending character or first unequal byte in the first

operand, whichever was encountered, is placed in general register R_1 , and the address of the second-operand byte corresponding in position to the first-operand byte is placed in general register R_2 . When condition code 2 is set, the address of the ending character or first unequal byte in the second operand, whichever was encountered, is placed in general register R_2 , and the address of the first-operand byte corresponding in position to the second-operand byte is placed in general register R_1 . When condition code 3 is set, the address of the next byte to be processed in the first and second operands is placed in general registers R_1 and R_2 , respectively. Whenever an address is placed in a general register, bits 32-39 of the register, in the 24-bit addressing mode, or bit 32 in the 31-bit addressing mode, are set to zeros. Bits 0-31 of the R_1 and R_2 registers always remain unchanged in the 24-bit or 31-bit mode.

When condition code 0 is set, the contents of general registers R_1 and R_2 remain unchanged.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

Access exceptions for the first and second operands are recognized only for that portion of the operand which is necessarily examined in the operation.

Resulting Condition Code:

- 0 Entire operands equal; general registers R_1 and R_2 unchanged
- 1 First operand low; general registers R_1 and R_2 updated with addresses of last bytes processed
- 2 First operand high; general registers R_1 and R_2 updated with addresses of last bytes processed
- 3 CPU-determined number of bytes equal; general registers R_1 and R_2 updated with addresses of next bytes

Program Exceptions:

- Access (fetch, operands 1 and 2)
- Specification

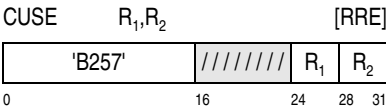
Programming Notes:

- 1. Several examples of the use of the COMPARE LOGICAL STRING instruction are given in

Appendix A, "Number Representation and Instruction-Use Examples."

- 2. When condition code 0 is set, no indication is given of the position of either ending character.
- 3. When condition code 3 is set, the program can simply branch back to the instruction to continue the comparison. The program need not determine the number of bytes that were compared.
- 4. R_1 or R_2 may be zero, in which case general register 0 is treated as containing an address and also the ending character.
- 5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

COMPARE UNTIL SUBSTRING EQUAL



The first operand is compared with the second operand until equal substrings (sequences of bytes) of a specified length are found, the end of the longer operand is reached, or a CPU-determined number of unequal bytes have been compared, whichever occurs first. The shorter operand is considered to be extended on the right with padding bytes. The CPU-determined number is at least 256. The result is indicated in the condition code.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is specified by the contents of the R_1 and R_2 general registers, respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the 32-bit signed binary integer in bit positions 32-63 of general registers $R_1 + 1$ and $R_2 + 1$, respectively. In the 64-bit addressing mode, the number of bytes is specified by the 64-bit signed binary integer in bit positions 0-63 of those registers. When an operand length is negative, it is treated as

zero, and it remains unchanged upon completion of the instruction.

Bits 56-63 of general register 0 specify the unsigned substring length, a value of 0-255, in bytes. Bits 56-63 of general register 1 are the padding byte. Bits 0-55 of general registers 0 and 1 are ignored.

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions

40-63 of general registers R_1 and R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-181 on page 7-145.

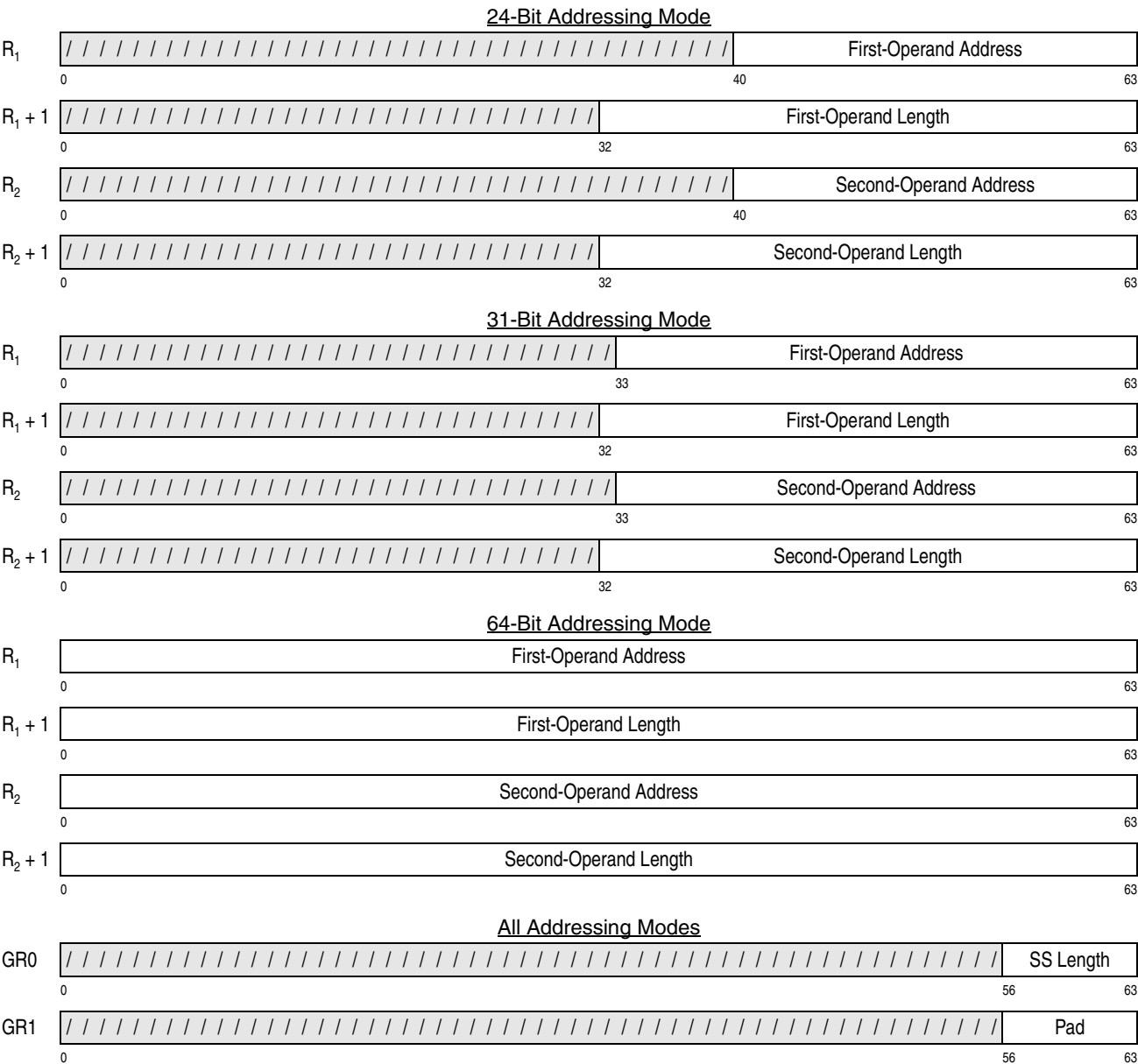


Figure 7-181. Register Contents and Second-Operand Address for COMPARE UNTIL SUBSTRING EQUAL

The result is obtained as if the operands were processed from left to right. However, multiple accesses may be made to all or some of the bytes of each operand.

The comparison proceeds left to right, byte by byte, and ends as soon as (1) equal substrings of the specified length are found, (2) the end of the longer operand is reached without finding equal substrings of the specified length, or (3) the last bytes compared are unequal, and a CPU-determined number of bytes have been compared. The CPU-determined number is at least 256. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of padding bytes.

If the operation ends because equal substrings of the specified length were found, the condition code is set to 0. If the operation ends because the end of the longer operand was reached without finding equal substrings of the specified length, the condition code is set to 1 if equal bytes were the last bytes compared, or it is set to 2 if unequal bytes were the last bytes compared. If the operation ends because unequal bytes were compared when a CPU-determined number of bytes had been compared, the condition code is set to 3.

If the specified substring length is zero, it is considered that equal substrings of the specified length were found, and condition code 0 is set.

If both operands are of zero length but the specified substring length is not zero, it is considered that the end of the longer operand was reached when unequal bytes were the last bytes compared, and condition code 2 is set.

If equal bytes have been compared but then unequal bytes are compared, it is considered that all bytes so far compared are unequal.

At the completion of the operation, the operand-length fields in the $R_1 + 1$ and $R_2 + 1$ registers are decremented by the number of unequal bytes compared (including equal bytes before unequal bytes compared), and the addresses in the R_1 and R_2 registers are incremented by the same number. However, in the case when a byte of the longer operand is compared against the padding byte, the length field for the shorter operand is not decremented below zero, and the corresponding address is not incre-

mented above the address of the first byte after the shorter operand. In the 24-bit or 31-bit addressing mode, the leftmost bits which are not part of the addresses in bit positions 32-63 of registers R_1 and R_2 are set to zeros, even if the substring length is zero or both operand lengths are initially zero.

Thus, when condition code 0 or 1 is set, the resulting addresses in the R_1 and R_2 registers designate the first bytes of equal substrings in the two operands, and the lengths in the $R_1 + 1$ and $R_2 + 1$ registers have been decremented by the number of bytes preceding the equal substrings, except when the equal substring in the shorter operand begins with the padding byte, in which case the length field for the shorter operand is zero, and the corresponding address field has been incremented by the operand length. When condition code 2 is set, each address field designates the first byte after the corresponding operand, and both length fields are zero. When condition code 3 is set, each address field designates the first byte after the last compared byte of the corresponding operand, and both length fields have been decremented by the number of bytes compared, except that a length field is not decremented below zero.

When the contents of the R_1 and R_2 fields are the same, the first and second operands may be compared, or the condition code may be set to 0 or 1 without comparing the operands.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, R_2 , and $R_2 + 1$, always remain unchanged.

The substring length and padding byte may be fetched from general registers 0 and 1 multiple times during the execution of the instruction, and the registers designated by R_1 and R_2 may be updated multiple times. Therefore, if R_1 or R_2 is zero, the results are unpredictable.

When condition code 3 is set, the general registers used by the instruction have been set so that the remainder of the operands can be processed by simply branching back and reexecuting the instruction.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

The execution of the instruction is interruptible when the last bytes compared are unequal; it is not interruptible when the last bytes compared are equal. When an interruption occurs, other than one that follows termination, the contents of the registers designated by the R_1 and R_2 fields are updated the same as upon normal completion of the instruction, so that the instruction, when reexecuted, resumes at the point of interruption. The condition code is unpredictable.

Access exceptions for the portion of a storage operand to the right of the last byte processed may or may not be recognized. For operands longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Although the operand address and length fields remain unchanged when a zero substring length is specified, the recognition of access exceptions is not necessarily prevented.

Resulting Condition Code:

- 0 Equal substrings of specified length found
- 1 End of longer operand reached when last bytes compared are equal
- 2 End of longer operand reached when last bytes compared are unequal
- 3 Last bytes compared are unequal, and CPU-determined number of bytes compared

Program Exceptions:

- Access (fetch, operands 1 and 2)
- Specification

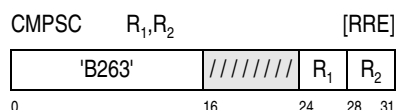
Programming Notes:

1. When the R_1 and R_2 fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, and, in the absence of dynamic modification of the operand area by another CPU or by a channel program, condition code 0, 1, or 2 is set (as explained in the next note). However, it is unpredictable whether access exceptions are recognized for the oper-

and since the operation can be completed without storage being accessed.

2. If the contents of the R_1 and R_2 fields are the same and the operand length is nonzero, and provided that another CPU or a channel program is not changing an operand, condition code 0 is set if the operand length is equal to or greater than the specified substring length, or condition code 1 is set if the operand length is less than the specified substring length. Whether or not R_1 equals R_2 , if both operand lengths are zero, condition code 0 is set if the specified substring length is zero, or condition code 2 is set if the specified substring length is nonzero. In all of these cases, the addresses in the R_1 and R_2 registers and the lengths in the $R_1 + 1$ and $R_2 + 1$ registers remain unchanged.
3. Special precautions should be taken when COMPARE UNTIL SUBSTRING EQUAL is made the target of an execute-type instruction. See the programming note concerning interruptible instructions under EXECUTE.
4. Other programming notes concerning interruptible instructions are included in "Interruptible Instructions" on page 5-20.
5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
6. The storage-operand references of COMPARE UNTIL SUBSTRING EQUAL may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)

COMPRESSION CALL



This definition assumes knowledge of the introductory information and information about dictionary formats in *Enterprise Systems Architecture/390 Data Compression*, SA22-7208-01.

The second operand is compressed or expanded, depending on a specification in general register 0, and the results are placed at the first-operand location. The compressed-data operand normally consists of index symbols corresponding to entries in a

dictionary designated by an address in general register 1. This dictionary is a compression dictionary during a compression operation or an expansion dictionary during an expansion operation. During compression when format-1 sibling descriptors are specified in general register 0, an expansion dictionary immediately follows the compression dictionary. During compression when the symbol-translation option is specified in general register 0, the index symbols resulting from compression are translated to interchange symbols by means of a symbol-translation table designated by the address and an offset in general register 1, and it is the interchange symbols that are placed at the first-operand location. The number of bits in a symbol in the compressed-data operand is specified in general register 0. The operation proceeds until the end of either operand is reached or a CPU-determined amount of data has been processed, whichever occurs first. The results are indicated in the condition code.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-

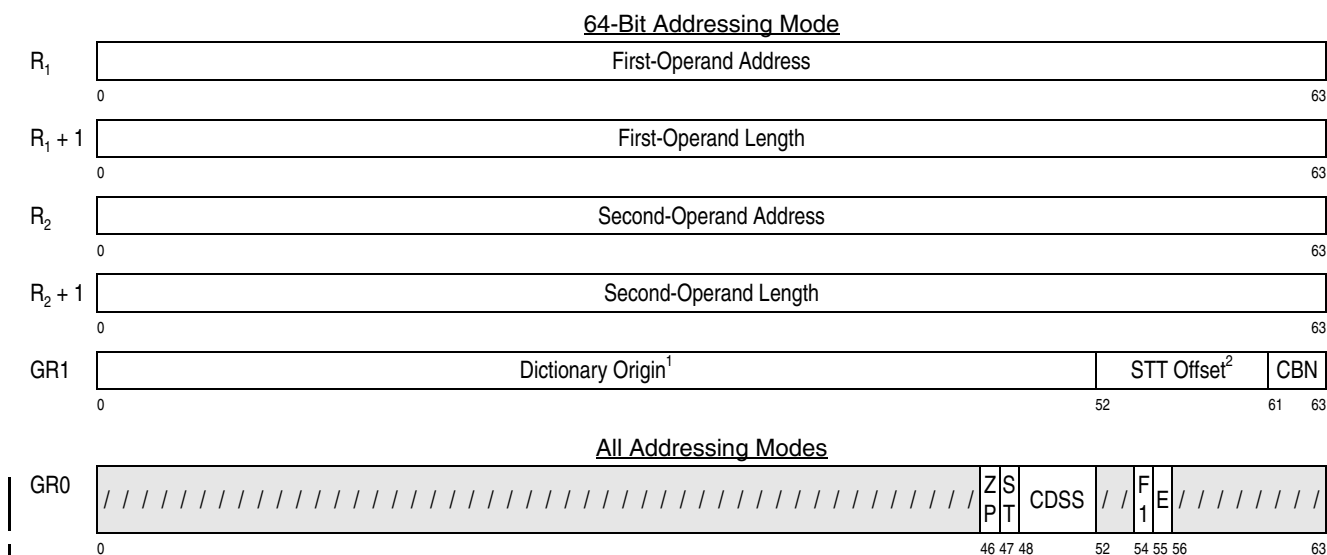
numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte containing any bit of the first operand and second operand is designated by an address in general registers R_1 and R_2 , respectively. The number of bytes containing any bits of the first operand and second operand is specified by bits 32-63 of general registers $R_1 + 1$ and $R_2 + 1$, respectively, in the 24-bit or 31-bit addressing mode or by bits 0-63 of the registers in the 64-bit addressing mode. The contents of general registers $R_1 + 1$ and $R_2 + 1$ are treated as 32-bit unsigned binary integers in the 24-bit or 31-bit addressing mode or as 64-bit unsigned binary integers in the 64-bit addressing mode.

The location of the leftmost byte of the compression dictionary during compression, or of the expansion dictionary during expansion, is designated on a 4K-byte boundary by an address in general register 1.



Figure 7-182. Register Contents for COMPRESSION CALL (Part 1 of 2)



Explanation:

- 1 Compression dictionary during compression, or expansion dictionary during expansion
2 STT offset field is only meaningful during compression when the ST bit is one. When the ST bit is zero during compression, or during expansion, the STT-offset field is ignored.

CBN Compressed-data bit number

CEN	Compressed data bit number
CDSS	Compressed-data symbol size, and dictionary size when symbol translation not specified

CDSS (binary)	Symbol Size	Dictionary
0000	Causes a specification exception to be recognized	
0001	9 bits	512 entries, 4K bytes
0010	10 bits	1K entries, 8K bytes
0011	11 bits	2K entries, 16K bytes
0100	12 bits	4K entries, 32K bytes
0101	13 bits	8K entries, 64K bytes
0110-1111	Causes a specification exception to be recognized	

E Expansion operation

E	Expansion operation
F1	Format-1 sibling descriptors (ignored during expansion)

ST	Symbol-translation option (ignored during expansion)
----	--

STT Symbol-translation-table (STT offset ignored during expansion)

ZP Zero padding control (when the CMPSC-enhancement facility is installed).

Figure 7-182. Register Contents for COMPRESSION CALL (Part 2 of 2)

The handling of the addresses in general registers R_1 , R_2 , and 1 is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-63 of registers R_1 and R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of registers R_1 and R_2 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of registers R_1 and R_2 constitute the address. In the 24-bit addressing mode, the contents of bit positions 40-51 of register 1, with 12 rightmost zeros appended, constitute the address, and the contents of bit positions 0-39 and

52-63 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-51 of register 1, with 12 rightmost zeros appended, constitute the address, and the contents of bit positions 0-32 and 52-63 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-51 of register 1, with 12 rightmost zeros appended, constitute the address, and the contents of bit positions 52-63 are ignored.

Although the contents of bit positions 52-63 of general register 1 are ignored as just described, those contents are used as follows. The contents of bit positions 61-63 of the register are the compressed-data bit number (CBN). At the beginning of the oper-

ation, the CBN designates the leftmost bit within the leftmost byte of the compressed-data operand. The compressed-data operand is the first operand during compression, or it is the second operand during expansion. When the symbol-translation option is specified during compression, the contents of bit positions 52-60 of the register, with seven rightmost zeros appended, are the byte offset from the beginning of the compression dictionary to the leftmost byte of the symbol-translation table. Symbol translation cannot be specified during expansion, and the contents of bit positions 52-60 are ignored during expansion.

The contents of the registers just described and also of general register 0 are shown in Figure 7-182 on page 7-148.

Bit 55 (E) of general register 0 specifies the compression operation if zero or the expansion operation if one.

When the CMPSC-enhancement facility is installed, bit 46 of general register 0 is the zero-padding (ZP) control. When the ZP control is zero, zero padding of the first operand is not performed. When the ZP control is one, zero padding of the first operand may be performed, as described below. When the CMPSC-enhancement facility is not installed, bit 46 of general register 0 is ignored.

Bit 47 (ST) of general register 0 is the symbol-translation-option bit. During compression when bit 47 is zero, the operation produces indexes, called index symbols, to compression-dictionary entries that represent character strings, and the operation then places the index symbols in the compressed-data operand. During compression when bit 47 is one, the operation still produces index symbols but then translates the index symbols to other symbols, called interchange symbols, that it then places in the compressed-data operand. This symbol translation is done by using the symbol-translation table specified by the address and offset in general register 1. Bit 47 and the offset in general register 1 are ignored during expansion. During expansion, the compressed-data operand always contains index symbols that designate entries in the expansion dictionary.

Bits 48-51 (CDSS) of general register 0 specify the number of bits in the index symbols or interchange symbols in the compressed-data operand, as shown in the figure. Bits 48-51 must not have any of the values 0000 or 0110-1111 binary; otherwise, a specifi-

cation exception is recognized. When symbol-translation is not specified, bits 48-51 also specify, as shown in the figure, the number of eight-byte entries in each of the compression and expansion dictionaries, and, thus, they specify the size in bytes of each of the dictionaries. When symbol translation is specified, the compression dictionary is considered to extend to the beginning of the symbol-translation table, that is, the size in bytes of the compression dictionary is the offset in bit positions 52-60 of general register 1, with seven rightmost zeros appended. The size in bytes of the symbol-translation table is considered to be one fourth that of the compression dictionary. However, the offset in general register 1 must be at least as large as the size of the compression dictionary would be if symbol translation were not specified and the CDSS were one less than it actually is, and, when the CDSS is 0001 binary, the offset must be at least 2K bytes; otherwise, the results are unpredictable. For example, if the CDSS is 0101, the offset must be at least 32K bytes.

Bit 54 (F1) of general register 0 specifies that the compression dictionary contains format-0 sibling descriptors if the bit is zero or format-1 sibling descriptors if the bit is one. Sibling descriptors are used during the compression operation. A format-0 sibling descriptor is eight bytes at an index position in the compression dictionary. A format-1 sibling descriptor is 16 bytes, with the first eight bytes at an index position in the compression dictionary and the second eight bytes at the same index position in the expansion dictionary. During compression when bit 54 is one, an expansion dictionary is considered to immediately follow the compression dictionary specified by the address in general register 1. Bit 54 is ignored during expansion.

Bits 47 and 54 of general register 0 must not both be ones; otherwise, the results are unpredictable.

The unused bit positions in general register 0 are reserved for possible future extensions and should contain zeros; otherwise, the program may not operate compatibly in the future.

In the access-register mode, the contents of access register R_1 are used for accessing the first operand, and the contents of access register R_2 are used for accessing the second operand and the dictionaries and the symbol-translation table.

The operation starts at the left end of both operands and proceeds to the right. The operation is ended

when the end of either operand is reached or when a CPU-determined amount of data has been processed, whichever occurs first.

During a compression operation, the end of the first operand is considered to be reached when the number of unused bit positions remaining in the first-operand location is not sufficient to contain additional compressed data.

During an expansion operation, the end of the first-operand location is considered to be reached when either of the following two conditions is met:

1. The number of unused byte positions remaining in the first-operand location is not sufficient to contain all the characters that would result from expansion of the next index symbol.
2. Immediately when the number of unused byte positions is zero, that is, immediately when the expansion of an index symbol completely fills the first-operand location.

During an expansion operation, the end of the second-operand location is considered to be reached when the next index symbol does not reside entirely within the second-operand location. The second-operand location ends at the beginning of the byte designated by the sum of the address in general register R_2 and the length in general register $R_2 + 1$, regardless of the compressed-data bit number in bit positions 61-63 of general register 1.

If the operation is ended because the end of the second operand is reached, condition code 0 is set. If the operation is ended because the end of the first operand is reached, condition code 1 is set, except that condition code 0 is set if the end of the second operand is also reached. If the operation is ended because a CPU-determined amount of data has been processed, condition code 3 is set.

At the completion of the operation, the length in general register $R_1 + 1$ is decremented by the number of complete bytes stored at the first-operand location, and the address in general register R_1 is incremented by the same amount. During compression, a complete byte is considered to be stored only if all of its bit positions contain bits of compressed data. During compression when the first bit of compressed data stored is not in bit position 0 of a byte, the bits in the byte to the left of the first bit of compressed data remain unchanged. During compression if the last

byte stored does not completely contain compressed data, the bits in the byte to the right of the rightmost bit of compressed data in the byte either are unchanged or are set to zeros.

The length in general register $R_2 + 1$ is decremented by the number of complete bytes processed at the second-operand location, and the address in general register R_2 is incremented by the same amount. During expansion, a complete byte is considered to be processed only if all of its bits have been used to produce expanded data.

The leftmost bits which are not part of the address in general registers R_1 and R_2 may be set to zeros or may remain unchanged. However, in the 24-bit or 31-bit addressing mode, bits 0-31 of these registers and also of general registers $R_1 + 1$ and $R_2 + 1$ always remain unchanged.

When all of the following conditions are met, zero padding may be performed, as described below.

- The CMPSC-enhancement facility is installed.
- The zero-padding (ZP) control, bit 46 of general register 0, is one.
- Any of the following is true:
 - The end of the first operand is reached (CC1).
 - The end of the second operand is reached (CC0).
 - A CPU-determined amount of data has been processed (CC3).
- The last byte of compressed or expanded data does not coincide with the rightmost byte of a model-dependent integral storage boundary.
- The remaining length in general register $R_1 + 1$ is sufficient to store up to the model-dependent integral storage boundary.

It is model dependent whether zero padding is performed. When any of the conditions listed above are not met, zero padding is not performed.

Zero padding consists of storing zeros to the right of the last compressed or expanded byte, up to a model-dependent integral storage boundary. The address in general register R_1 and the length in general register $R_1 + 1$ are not updated to account for the

zero padding. PER storage-alteration events are recognized for stores that occur as a result of zero padding. The model-dependent integral storage boundary is no larger than 4,096.

When the R_1 and R_2 fields do not designate general register 0, the handling of general register 1 is as follows. The bit number of the bit following the last bit of compressed data processed is placed in bit positions 61-63 of general register 1, and bits 52-60 of the register and the leftmost bits which are not part of the address in the register may be set to zeros or may remain unchanged, except that when one or both of the original length values are so small that no compressed data can be processed, all bits in the register may remain unchanged. However, when bit 47 of general register 0 is one, bits 52-60 of general register 1 always remain unchanged. Also, in the 24-bit or 31-bit addressing mode, bits 0-31 of the register always remain unchanged.

If (a) the operands overlap one another, (b) the first operand overlaps the dictionaries or the symbol-translation table in storage in any way, or (c) either the R_1 or the R_2 field designates general register 0, the results are unpredictable.

When symbol translation is specified, the symbol-translation table consists of two-byte entries, and an entry contains an interchange symbol in the rightmost bit positions of the entry. The length of the interchange symbol is specified by bits 48-51 of general register 0. The left-hand bits that are not part of the interchange symbol in a symbol-translation-table entry must be zeros; otherwise, the results are unpredictable.

To translate an index symbol to an interchange symbol, the index symbol is multiplied by 2 and then added to the address of the beginning of the symbol-translation table to locate an entry in the table, and then the interchange symbol is obtained from the entry.

The execution of the instruction is interruptible. When an interruption occurs, other than one that follows termination, the contents of the registers designated by the R_1 and R_2 fields and of general register 1 are updated the same as upon normal completion of the instruction, so that the instruction, when reexecuted, resumes at the point of interruption. The condition code is unpredictable.

For operands longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the current location being processed. Access exceptions may be recognized for all locations in the dictionaries and symbol-translation table if those areas are specified to be used and even if the locations would not be used during the operation. Access exceptions are not recognized for an operand, the dictionaries, or the symbol-translation table if the R field associated with that operand is odd. Also, when the R_1 field is odd, PER storage-alteration events are not recognized, and no change bits are set.

If an access exception is due to be recognized for either of the operands or for a dictionary or the symbol-translation table, the result is that either the exception is recognized or condition code 3 is set. If condition code 3 is set, the exception will be recognized when the instruction is executed again to continue processing the same operands, assuming that the exception condition still exists.

During compression, regardless of whether the exception is recognized or condition code 3 is set, a nullifying access-exception condition or a suppressing DAT-protection exception condition is handled so that an index symbol is generated only if it is the one that would result if there were no access-exception condition.

During compression or expansion, regardless of whether the exception is recognized or condition code 3 is set, a nullifying or suppressing access-exception condition may result in data having been stored at the first-operand location at or to the right of the location designated by the final address in general register R_1 , which result is not true nullification or suppression. The amount of data stored depends on the reason for the access-exception condition. If the condition is due to a reference to a dictionary or the symbol-translation table, up to 4K bytes of data may have been stored at or to the right of the location designated by the final address. If the condition is due to a reference to the first or second operand, part of one index or interchange symbol, during compression, or part of one character symbol, during expansion, may have been stored at or to the right of the location designated by the final address. In all cases, the storing will be repeated when the instruction is executed again to continue processing the same operands.

If the end of the first operand is reached and an access exception is due to be recognized for the sec-

ond operand, it is unpredictable whether condition code 1 is set or the access exception is recognized.

During expansion when the expansion dictionary is not logically correct, unusual storing may occur as described in the section “Expansion Process” in Chapter 1 of *Enterprise Systems Architecture/390 Data Compression*, SA22-7208-01. The results of an access exception in this case may not be true nullification or suppression.

Special Conditions

During compression of each character symbol, either the characters in the symbol or the dictionary character entries (not sibling descriptors) representing characters of the symbol are counted, and a data exception is recognized if this count becomes too large. The count can reach at least 260 without the exception being recognized.

During compression, the number of child characters or sibling characters processed during the processing of each parent entry are counted, and a data exception is recognized if this count becomes too large. The count can reach at least 260 without the exception being recognized. That is, a parent must not have more than 260 children; otherwise, a data exception may be recognized.

During expansion of each character symbol, either the characters in the symbol or the dictionary entries representing characters of the symbol are counted, and a data exception is recognized if this count becomes too large. If the characters in the symbol are counted, the count can reach at least 260 without the exception being recognized. If the dictionary entries representing characters of the symbol are counted, the count can reach at least 127 without the exception being recognized.

Certain error conditions in the dictionaries cause a data exception to be recognized and the operation to be either suppressed or terminated. Some of these error conditions are described in the sections “Expansion Process” and “Results of Dictionary Errors” in Chapter 1 of SA22-7208-01. The others are described in Chapter 2 of SA22-7208-01.

Resulting Condition Code:

- 0 End of second operand reached
- 1 End of first operand reached and end of second operand not reached
- 2 --
- 3 CPU-determined amount of data processed

Program Exceptions:

- Access (fetch, operand 2, dictionaries, and symbol-translation table; store, operand 1)
- Data
- Specification

Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the operation. The program need not determine the amount of data processed.
2. During compression when a nullifying access exception is due to be recognized, an index symbol is generated only if it is the one that would result if there were no access-exception condition. The result of this is that compression of the same expanded data by means of one or more executions of the instruction and by using the same dictionary always results in the same compressed data. That is, (1) the best possible matches in the dictionary are always found for the characters in the second operand, or else the execution is ended by either setting CC3 or recognizing the exception, and (2) the results of compression are repeatable (although possibly by means of a different number of executions of the instruction) and predictable.

For example, if the next characters of the string being compressed are ABC, and dictionary entry A has a child B, which has a child C, the normal operation is to compress ABC as a single index symbol, but if the C of the string is in the first byte of an invalid page (a page-translation exception is due to be recognized), no index symbol is generated. More specifically, an index symbol corresponding to the character symbol AB is not generated because this is not the index symbol that would be generated if the access-exception condition did not exist.

In the above, “best possible match” refers only to the characters in the second operand. In the example above, if the next two characters of the

second operand (the string) are AB, and these are the last two characters of the second operand, the best possible match is on AB, even though there could be a match on ABC if the second operand included one more byte containing C.

Expansion is normally always repeatable. An index symbol is always expanded to exactly the character symbol it represents unless an exception that causes termination is recognized.

3. During expansion, if at least one unused byte position remains in the first operand location, COMPRESSION CALL may completely process the next index symbol in the second operand before it determines that the first-operand location does not have sufficient unused byte positions to contain the expanded data that would result from the next index symbol. If that next index symbol causes encountering of bad dictionary entries, the result can be either a data exception or condition code 1.

COMPRESSION CALL immediately sets condition code 1 when processing of an index symbol exactly fills the first-operand location, except that it sets condition code 0 if the end of the second-operand location also has been reached. Immediately setting condition code 1 has the advantage that data can be compressed using one dictionary and then followed immediately, possibly on a bit boundary, by a different type of data compressed using another dictionary. The compressed data can be successfully expanded if, during the expansion of the data compressed using the first dictionary, the length of the first-operand location is specified to be exactly the length of the expanded data that will be produced. Condition code 1 will then be set when the first-operand location is full, at which time the specification of the dictionary can be changed in order to expand the remainder of the compressed data using the second dictionary. If the definition allowed condition code 1 not to be set, it might be attempted to expand the next index symbol, which resulted from use of the second dictionary, by means of the first dictionary, and this might cause recognition of a data exception. For example, the next index symbol, which properly designates a character entry in the second dictionary, might designate the second half of a format-1 sibling descriptor in the first dictionary,

and that second half might begin with a character, such as 0 (F0 hex), that would appear to be an invalid partial symbol length in a character entry.

4. A nullifying access-exception condition due to a reference to a dictionary or the symbol-translation table may result in the storing of data at or to the right of the location designated by the final address in general register R_1 . This storing and the processing needed to produce the data stored will be repeated when COMPRESSION CALL is executed again to continue processing the same operands. The repeated processing will reduce the performance of the instruction execution, and it should be avoided by ensuring that the environment in which the program is executed is one in which page-translation-exception conditions for the dictionaries and symbol-translation table are infrequent.
5. Following is an example of how the compressed-data bit number (CBN) is used and set. In this example:
 - The operation is an expansion operation.
 - The CDSS in general register 0 is 0010 binary. Therefore, there are 1K entries in the expansion dictionary, and the length of an index symbol is 10 bits.
 - The second operand (compressed-data operand) begins at location 6000 hex and has a length of five bytes. The initial CBN is 7. Therefore, there are three index symbols to be expanded, and the final CBN will be 5.
 - The compressed data beginning at location 6000 hex is 0081FF9FF8 hex. Therefore, the three index symbols are 103, 3FC, and 3FF hex.
 - The first operand (expanded-data operand) begins at location 5000 hex and has a length of 64 bytes. The three index symbols are expanded to a total of 14 bytes of expanded data.

The following figure shows the initial and final contents of general registers R_1 , $R_1 + 1$, R_2 , and $R_2 + 1$, the contents of locations 6000-6004 hex in binary, and the way a cursor corresponding to

the CBN is advanced during the expansion operation.

Register	Initial Contents in Hex	Final Contents in Hex
R ₁	5000	500E
R ₁ +1	40	32
R ₂	6000	6004
R ₂ +1	5	1

Contents of Locations 6000-6004 Hex in Binary

00000000	10000001	11111111	10011111	11111000
↑		↑	↑	↑
Initial				Final
CBN				CBN
(7)				(5)

- The reason for allowing a parent to have no more than 260 children is as follows. The parent can contain five identical child characters. Then, 255 different sibling characters are possible — all of these must be different from the child characters and each other, or else they may be wasted (never matched against), depending on the implementation. Thus, every possible child is permitted.
- Symbol translation is for use by VTAM. VTAM will begin by doing compression by means of software and an adaptive dictionary. When the adaptive dictionary has matured such that the degree of compression becomes sufficiently good (crosses some threshold), VTAM will “freeze” (stop adapting) its dictionary, inform the other end of the session to freeze also, transform its adaptive dictionary to the dictionary form used by COMPRESSION CALL, and then use COMPRESSION CALL to continue on with the compression. The other end of the session can continue to use its frozen adaptive dictionary.

Following is clarification about the STT offset. Assume VTAM uses a 4K-entry adaptive dictionary, which is the largest size VTAM uses. All of the entries in this dictionary correspond to character symbols because there are no sibling descriptors in the VTAM dictionary. The VTAM

- dictionary cannot map one-to-one to a COMPRESSION CALL dictionary because the latter requires that some of the entries be sibling descriptors. Therefore, VTAM must have an 8K-entry dictionary for use in the basic compression operation. Only the first hundred or so entries in the second 4K of the 8K need to be used, and these entries compensate for (take the place of) the entries in the first 4K that must be sibling descriptors. The STT can and should, to save space, begin immediately after those hundred or so entries in the second 4K. In this example, the index symbols will be 13 bits but will be transformed to 12-bit interchange symbols.
- A program may place the dictionaries in pages that are managed by means of chaining fields at their beginnings. In this case, either the parts of a dictionary have to be moved to be compacted into contiguous locations or there have to be holes in the dictionaries. The definition of COMPRESSION CALL contains nothing explicitly to support holes. However, assuming there is at least one character that never appears in the expanded data, that character can be used as a child character in a parent entry or as a sibling character in a sibling descriptor to specify a child or children that will never be referenced, thus creating a hole.
 - The references to the operands, dictionaries, and symbol-translation table for COMPRESSION CALL may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
 - Setting the ZP control to one may provide improved performance on models in which the CMPSC-enhancement facility is installed.
 - Zero padding is not necessarily performed in every case where it is permissible.
 - Figure 7-183 on page 7-156 and Figure 7-184 on page 7-158 show possible forms (not the only possible forms) of the compression and expansion processes. The figures do not show testing for or the results of dictionary errors.

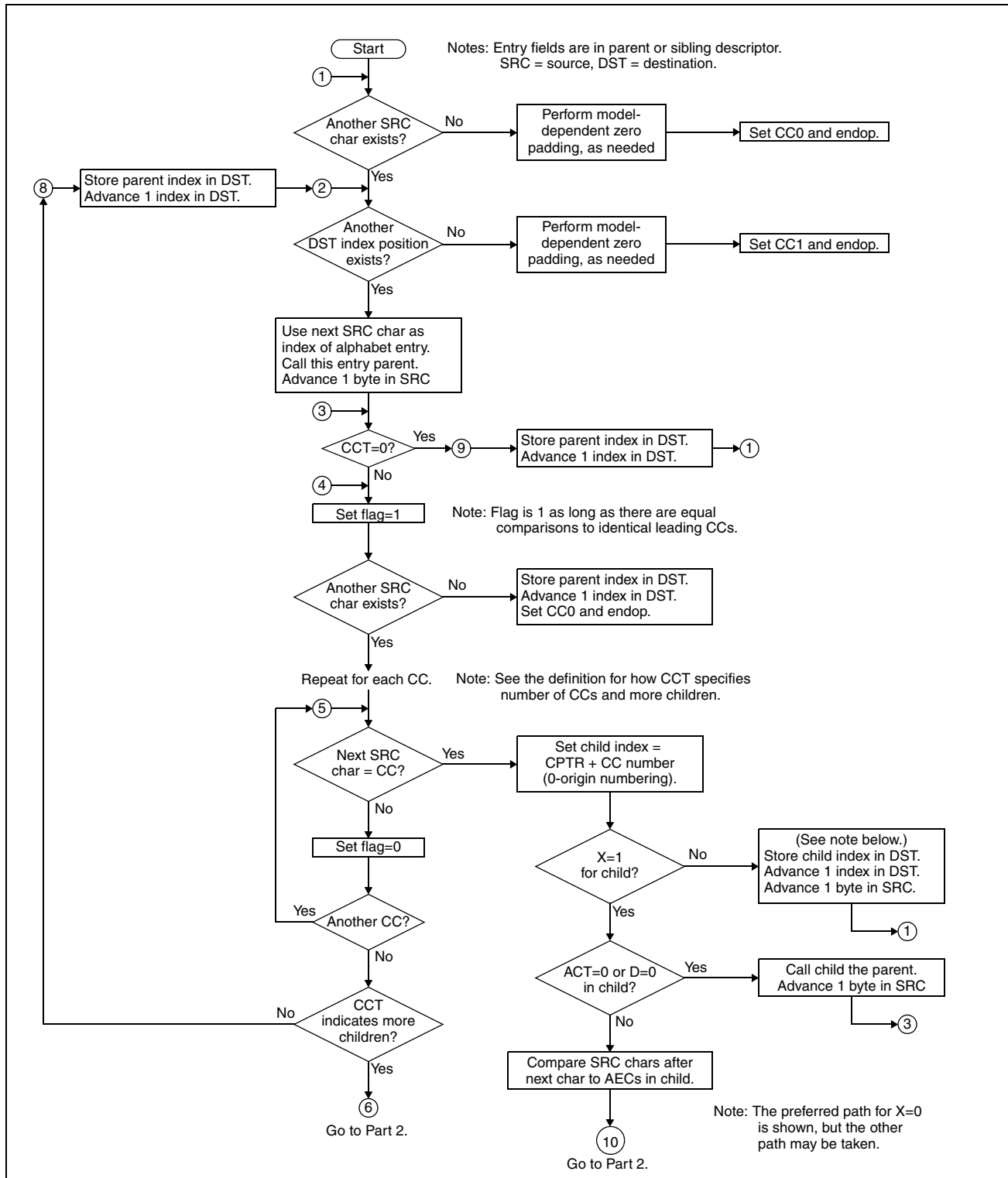


Figure 7-183. Compression Process (Part 1 of 2)

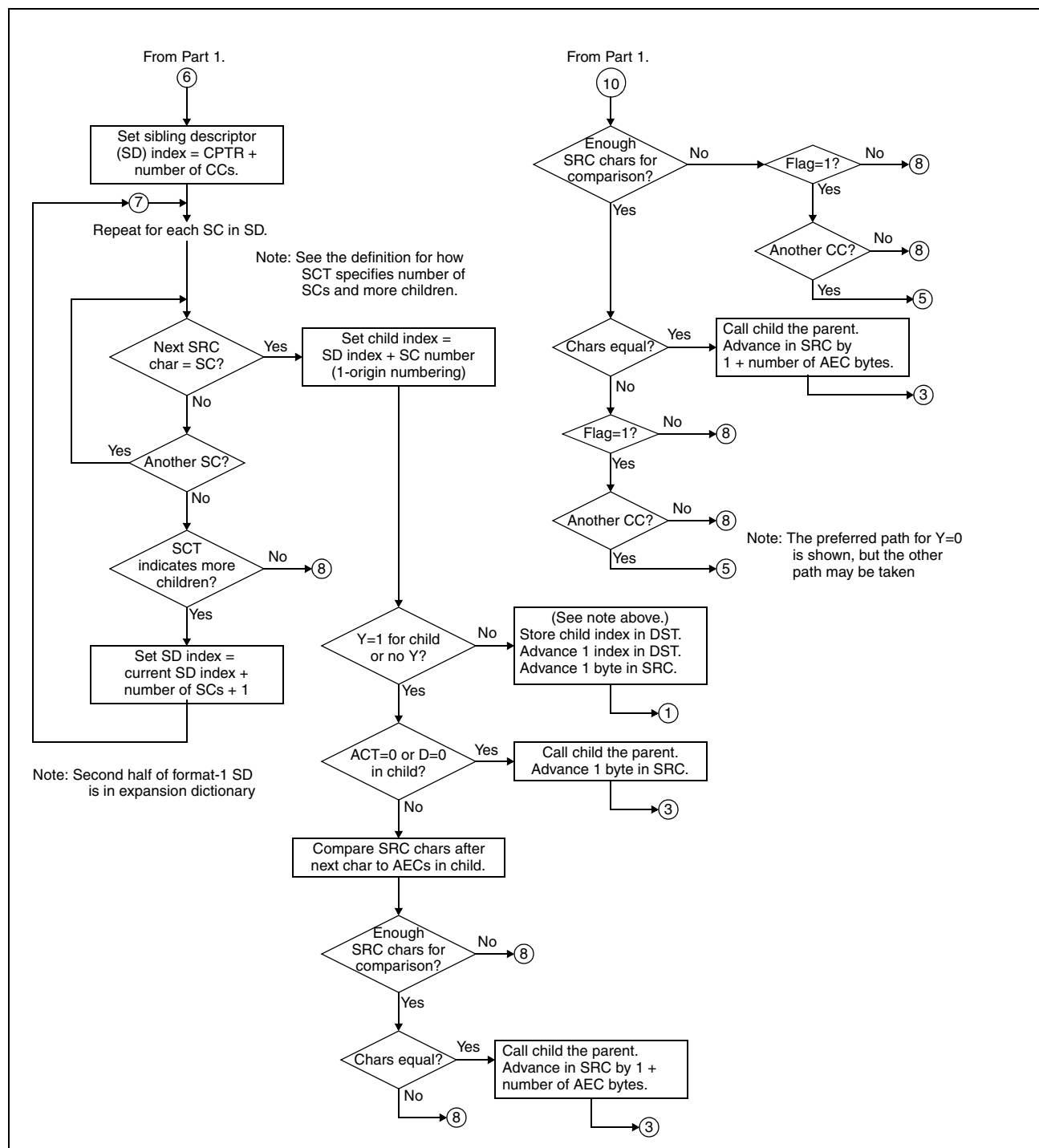


Figure 7-183. Compression Process (Part 2 of 2)

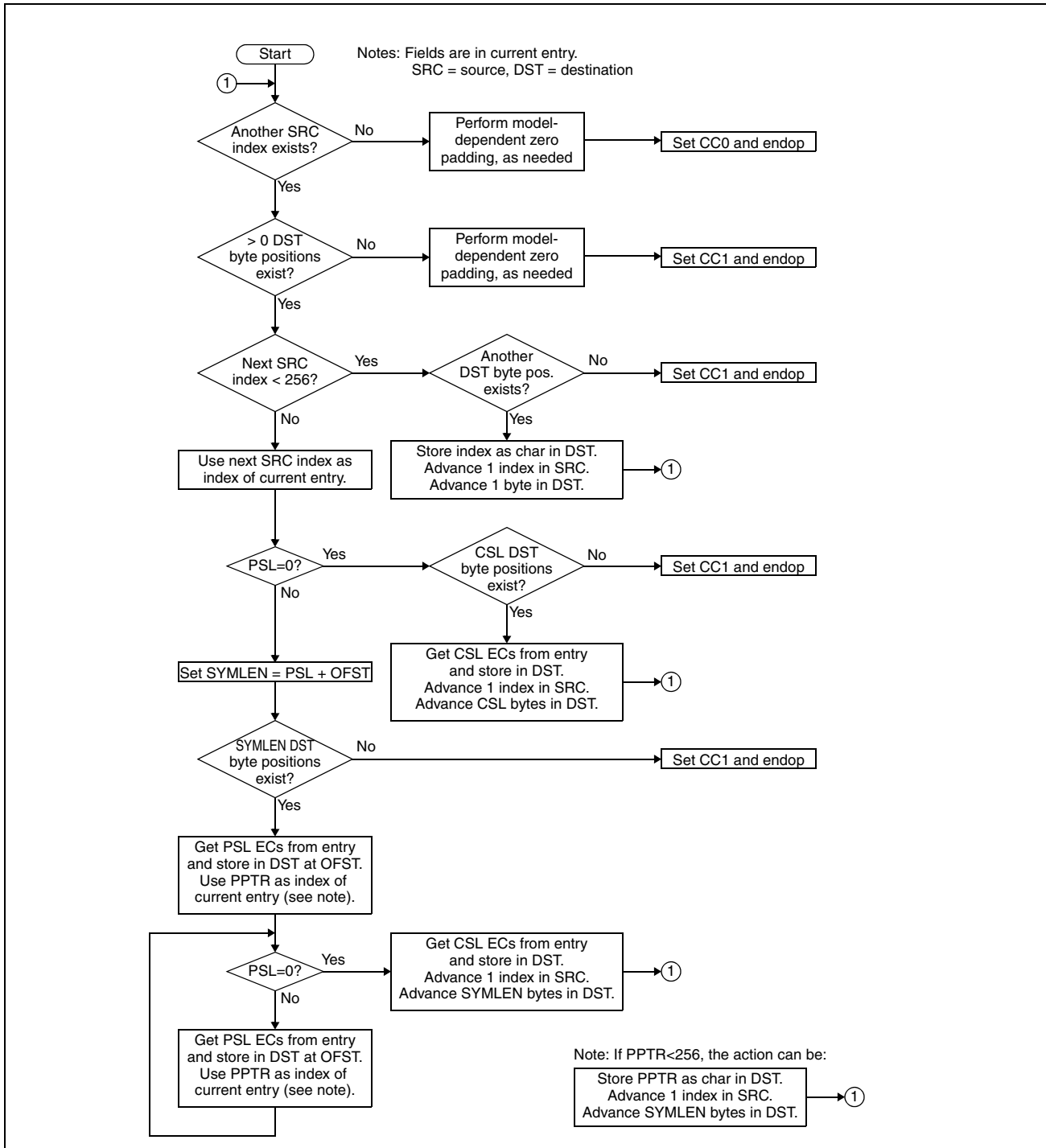
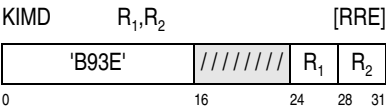
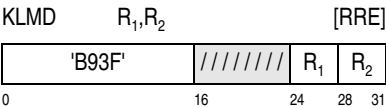


Figure 7-184. Expansion Process

COMPUTE INTERMEDIATE MESSAGE DIGEST



COMPUTE LAST MESSAGE DIGEST



A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction and the R₁ field are ignored.

Bit positions 57-63 of general register 0 contain the function code. Figure 7-185 and Figure 7-186 show the assigned function codes for COMPUTE INTERMEDIATE MESSAGE DIGEST and COMPUTE LAST MESSAGE DIGEST, respectively. All other function codes are unassigned. Bit 56 of general register 0 must be zero; otherwise, a specification exception is recognized. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

The function codes for COMPUTE INTERMEDIATE MESSAGE DIGEST are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	KIMD-Query	16	—
1	KIMD-SHA-1	20	64
2	KIMD-SHA-256	32	64
3	KIMD-SHA-512	64	128
65	KIMD-GHASH	32	16
Explanation: — Not applicable			

Figure 7-185. Function Codes for COMPUTE INTERMEDIATE MESSAGE DIGEST

The function codes for COMPUTE LAST MESSAGE DIGEST are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	KLMD-Query	16	—
1	KLMD-SHA-1	28	64
2	KLMD-SHA-256	40	64
3	KLMD-SHA-512	80	128
Explanation: — Not applicable			

Figure 7-186. Function Codes for COMPUTE LAST MESSAGE DIGEST

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers R₂ and R₂ + 1 are ignored for the query function.

For all other functions, the second operand is processed as specified by the function code using an initial chaining value in the parameter block, and the result replaces the chaining value. For COMPUTE LAST MESSAGE DIGEST, the operation also uses a message bit length in the parameter block. The operation proceeds until the end of the second-operand location is reached or a CPU-determined number of

bytes have been processed, whichever occurs first. The result is indicated in the condition code.

The R_2 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the second operand is specified by the contents of the R_2 general register. The number of bytes in the second-operand location is specified in general register $R_2 + 1$.

As part of the operation, the address in general register R_2 is incremented by the number of bytes processed from the second operand, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the address and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general register R_2 constitute the address of second operand, and the contents of bit positions 0-39 are ignored; bits 40-63 of the updated address replace the corresponding bits in general register R_2 , carries out of bit position 40 of the updated address are ignored, and the contents of bit positions 32-39 of general register R_2 are set to zeros. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register R_2 constitute the address of second operand, and the contents of bit positions 0-32 are ignored; bits 33-63 of the updated address replace the corresponding bits in general register R_2 , carries out of bit position 33 of the updated address are ignored, and the content of bit position 32 of general register R_2 is set to zero. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register R_2 constitute the address of second operand; bits 0-63 of the updated address replace the contents of general register R_2 and carries out of bit position 0 are ignored.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 32-63 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which specifies the number of bytes in the second operand; and the updated value replaces the contents of bit positions 32-63 of general register $R_2 + 1$. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register $R_2 + 1$ form a 64-bit unsigned binary integer which specifies the number of bytes in the second operand; and the updated value replaces the contents of general register $R_2 + 1$.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_2 and $R_2 + 1$, always remain unchanged.

Figure 7-187 on page 7-161 shows the contents of the general registers just described.

In the access-register mode, access registers 1 and R_2 specify the address spaces containing the parameter block and second operand, respectively.

The result is obtained as if processing starts at the left end of the second operand and proceeds to the right, block by block. The operation is ended when all source bytes in the second operand have been processed (called normal completion), or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually nonzero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless reoccurrence of this no-progress case.

When the chaining-value field overlaps any portion of the second operand, the result in the chaining-value field is unpredictable.

For COMPUTE INTERMEDIATE MESSAGE DIGEST, normal completion occurs when the number of bytes in the second operand as specified in general register $R_2 + 1$ have been processed. For COMPUTE LAST MESSAGE DIGEST, after all bytes in the second operand as specified in general register $R_2 + 1$ have been processed, the padding operation is performed, and then normal completion occurs.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in $R_2 + 1$ is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in $R_2 + 1$ is nonzero.

When the second-operand length is initially zero, the second operand is not accessed, general registers R_2 and $R_2 + 1$ are not changed, and condition code 0 is set. For COMPUTE INTERMEDIATE MESSAGE DIGEST, the parameter block is not accessed. However, for COMPUTE LAST MESSAGE DIGEST, the empty block ($L = 0$) case padding operation is per-

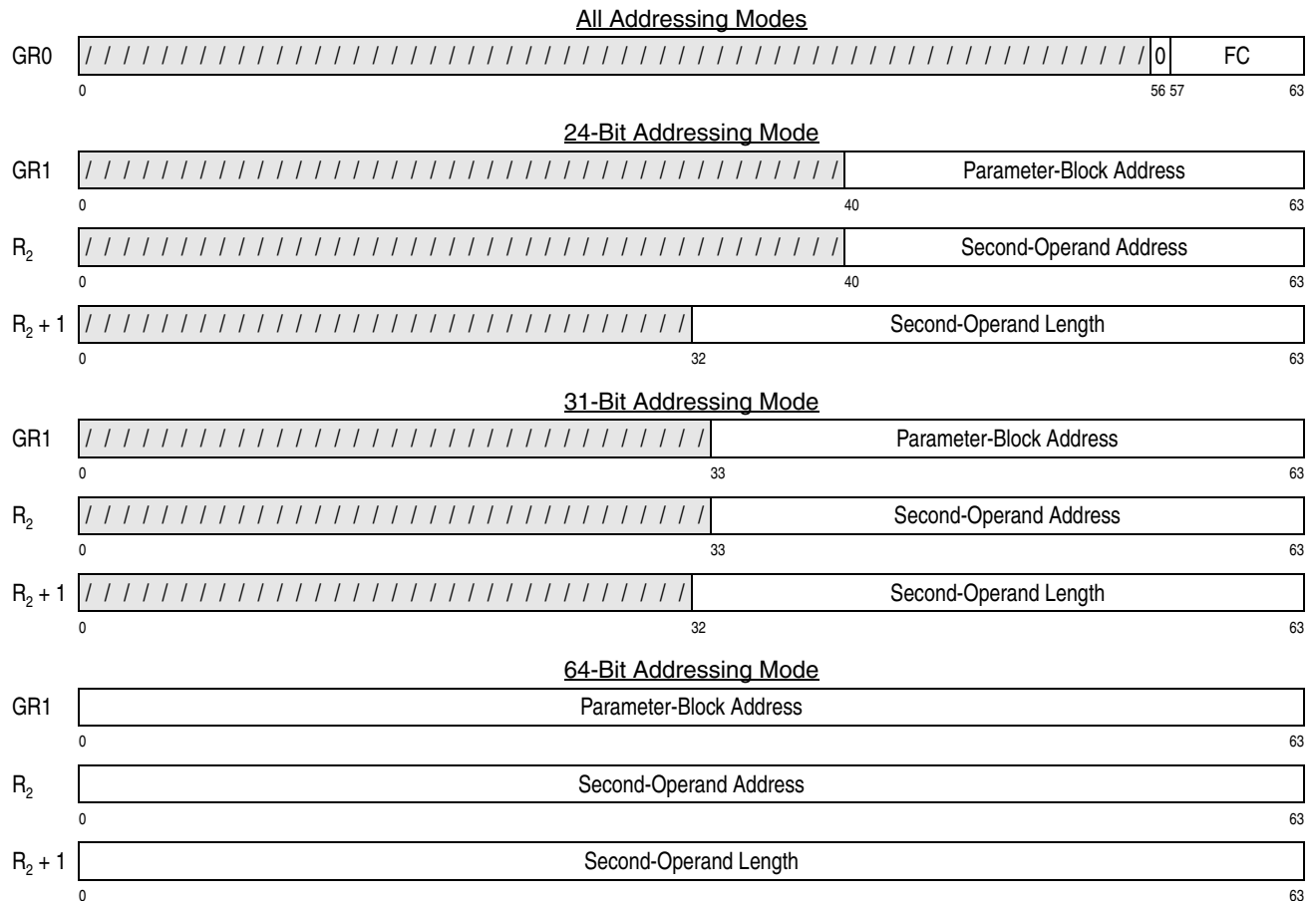


Figure 7-187. General Register Assignment for KIMD and KLMD

formed and the result is stored into the parameter block.

As observed by other CPUs and channel programs, references to the parameter block and storage operands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

Access exceptions may be reported for a larger portion of the second operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of the second operand nor for locations

more than 4K bytes beyond the current location being processed.

Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the COMPUTE INTERMEDIATE MESSAGE DIGEST and COMPUTE LAST MESSAGE DIGEST functions. Further description of the secure hash algorithm may be found in *Secure Hash Standard*, Federal Information Processing Standards publication 180-2, National Institute of Standards and Technology, Washington DC, August 1, 2002. Further description of the GCM (Galois/counter mode) multiplication over $GF(2^{128})$ may be found in the draft of NIST Special Publication 800-38D, dated June 2007.

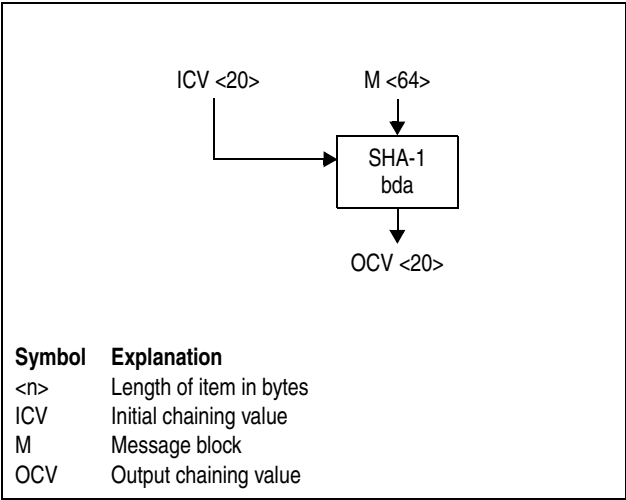


Figure 7-188. Symbol for SHA-1 Block Digest Algorithm

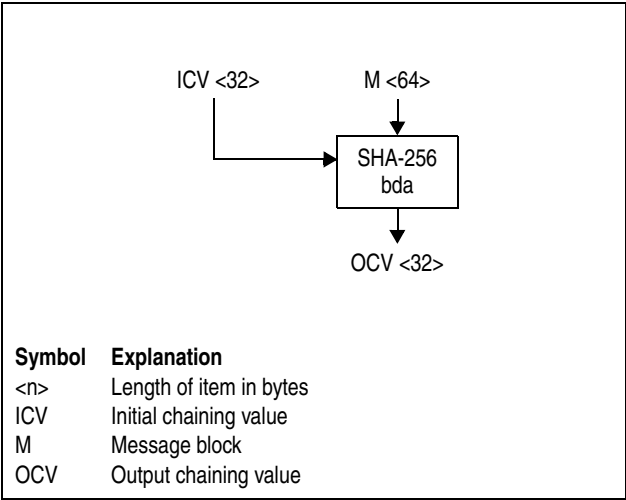


Figure 7-189. Symbol for SHA-256 Block Digest Algorithm

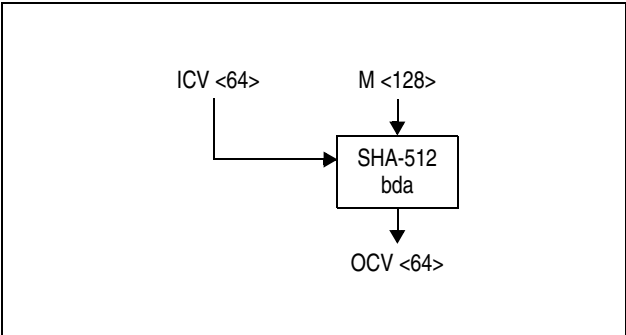


Figure 7-190. Symbol for SHA-512 Block Digest Algorithm

Symbol	Explanation
$<n>$	Length of item in bytes
ICV	Initial chaining value
M	Message block
OCV	Output chaining value

Figure 7-190. Symbol for SHA-512 Block Digest Algorithm

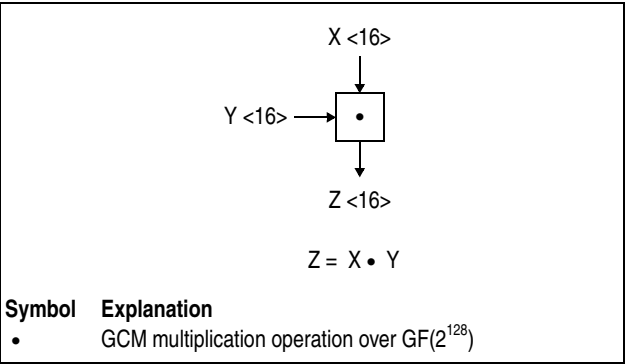


Figure 7-191. Symbol For GCM Multiplication Operation Over $GF(2^{128})$

KIMD-Query (KIMD Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the function has the following format:

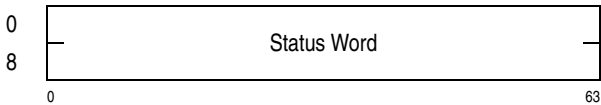


Figure 7-192. Parameter Block for KIMD-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the COMPUTE INTERMEDIATE MESSAGE DIGEST instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KIMD-Query function completes; condition code 3 is not applicable to this function.

KIMD-SHA-1 (KIMD Function Code 1)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the function has the following format:

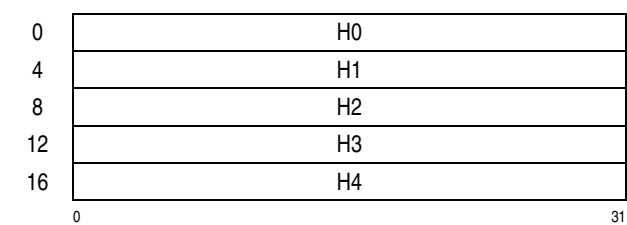


Figure 7-193. Parameter Block for KIMD-SHA-1

A 20-byte intermediate message digest is generated for the 64-byte message blocks in operand 2 using the SHA-1 block digest algorithm with the 20-byte chaining value in the parameter block. The generated intermediate message digest, also called the output chaining value (OCV), is stored in the chaining-value field of the parameter block. The operation is shown in Figure 7-194.

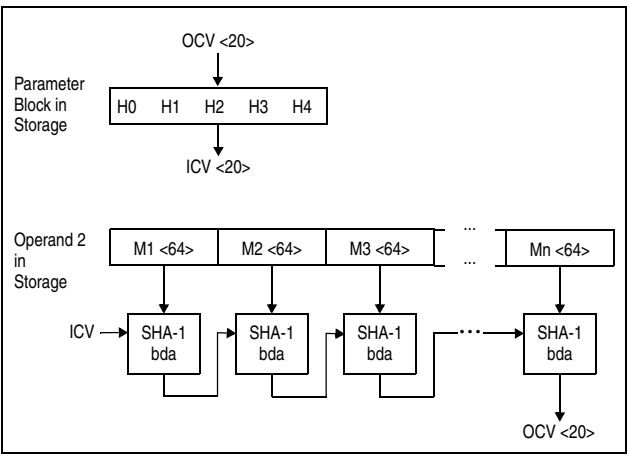


Figure 7-194. KIMD-SHA-1

KIMD-SHA-256 (KIMD Function Code 2)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the function has the following format:

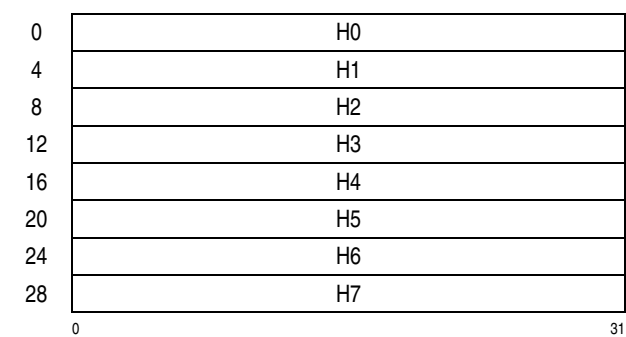


Figure 7-195. Parameter Block for KIMD-SHA-256

A 32-byte intermediate message digest is generated for the 64-byte message blocks in operand 2 using the SHA-256 block digest algorithm with the 32-byte chaining value in the parameter block. The generated intermediate message digest, also called the output chaining value (OCV), is stored in the chaining-value field of the parameter block. The operation is shown in Figure 7-196.

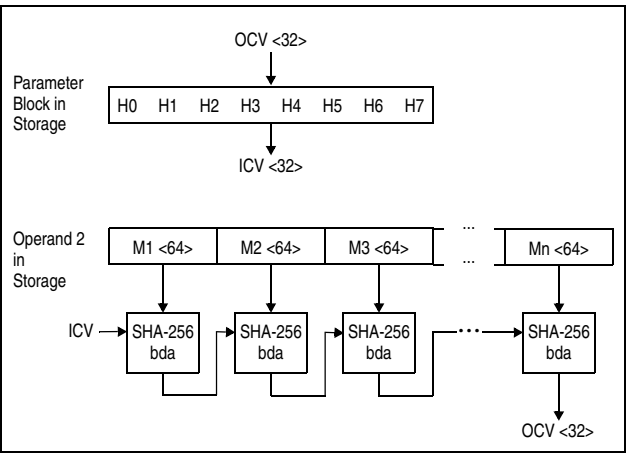


Figure 7-196. KIMD-SHA-256

KIMD-SHA-512 (KIMD Function Code 3)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the function has the following format:

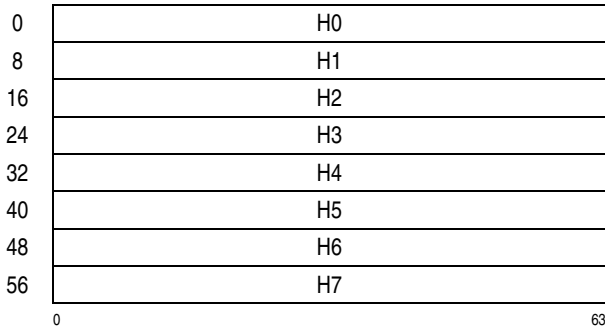


Figure 7-197. Parameter Block for KIMD-SHA-512

A 64-byte intermediate message digest is generated for the 128-byte message blocks in operand 2 using the SHA-512 block digest algorithm with the 64-byte chaining value in the parameter block. The generated intermediate message digest, also called the output chaining value (OCV), is stored in the chaining-value field of the parameter block. The operation is shown in Figure 7-196.

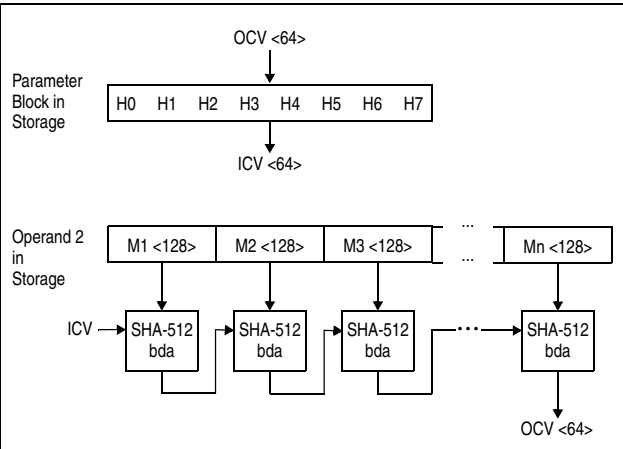


Figure 7-198. KIMD-SHA-512

KIMD-GHASH (Function Code 65)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the KIMD-GHASH function has the following format:

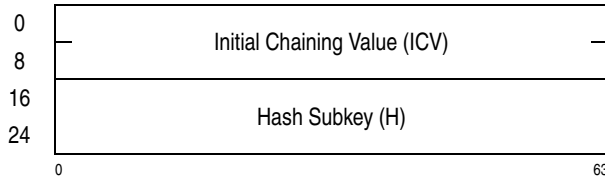


Figure 7-199. Parameter Block for KIMD-GHASH

The initial-chaining value is in byte offsets 0-15 of the parameter block, and the hash subkey is in byte offsets 16-31 of the parameter block.

The GHASH for the 16-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the hash subkey and the 128-bit chaining value.

The first message block is exclusive-ORed with the initial chaining value (ICV) in the parameter block. The result of the exclusive-OR operation and the hash subkey (H) are multiplied to produce the chaining value for the next message block. The process is repeated until all message blocks have been processed, or until a CPU-determined number of message blocks have been processed.

The GHASH, called the output chaining value (OCV), is stored into the Initial-chaining-value field of the parameter block. The operation is shown in Figure 7-200 on page 7-164.

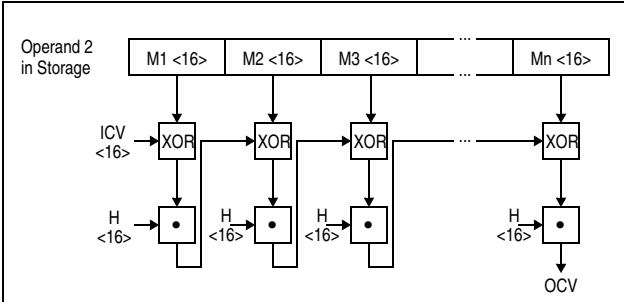


Figure 7-200. KIMD-GHASH

KLMD-Query (KLMD Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the function has the following format:

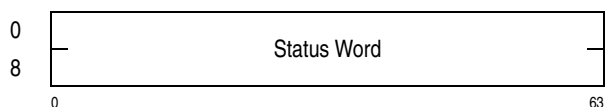


Figure 7-201. Parameter Block for KLMD-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the COMPUTE LAST MESSAGE DIGEST instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KLMD-Query function completes; condition code 3 is not applicable to this function.

KLMD-SHA-1 (KLMD Function Code 1)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the function has the following format:

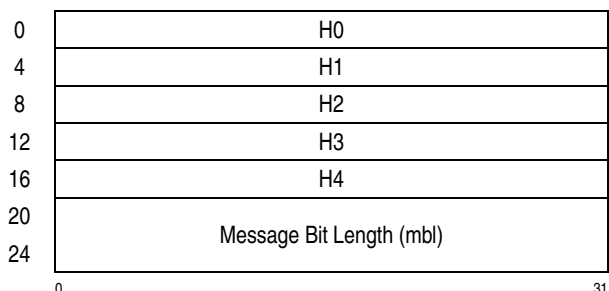


Figure 7-202. Parameter Block for KLMD-SHA-1

The message digest for the message (M) in operand 2 is generated using the SHA-1 algorithm with the chaining value and message-bit-length information in the parameter block.

If the length of the message in operand 2 is equal to or greater than 64 bytes, an intermediate message digest is generated for each 64-byte message block using the SHA-1 block digest algorithm with the 20-byte chaining value in the parameter block, and the generated intermediate message digest, also called the output chaining value (OCV), is stored into

the chaining-value field of the parameter block. This operation is shown in Figure 7-203 and repeats until the remaining message is less than 64 bytes.

If the length of the message or the remaining message is zero bytes, then the operation in Figure 7-204 is performed. If the length of the message or the remaining message is between one byte and 55 bytes inclusive, then the operation in Figure 7-205 is performed; if the length is between 56 bytes and 63 bytes inclusive, then the operation in Figure 7-206 is performed. The message digest, also called the output chaining value (OCV), is stored into the chaining-value field of the parameter block.

Additional Symbols Used in KLMD Functions

The following additional symbols are used in the description of the COMPUTE LAST MESSAGE DIGEST functions.

Symbol	Explanation for KLMD Function Figures
L	Byte length of operand 2 in storage.
p <n>	n padding bytes; leftmost byte is 80 hex; all other bytes are 00 hex.
z <n>	n padding bytes of zero.
mbl	an 8-byte or 16-byte value specifying the bit length of the total message.
q <64>	a padding block, consisting of 56 bytes of zero followed by an 8-byte mbl.
q <128>	a padding block, consisting of 112 bytes of zero followed by a 16-byte mbl.

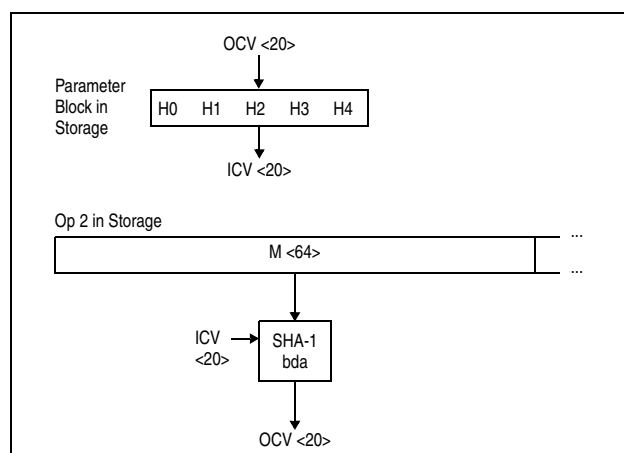


Figure 7-203. KLMD-SHA-1 Full Block ($L \geq 64$)

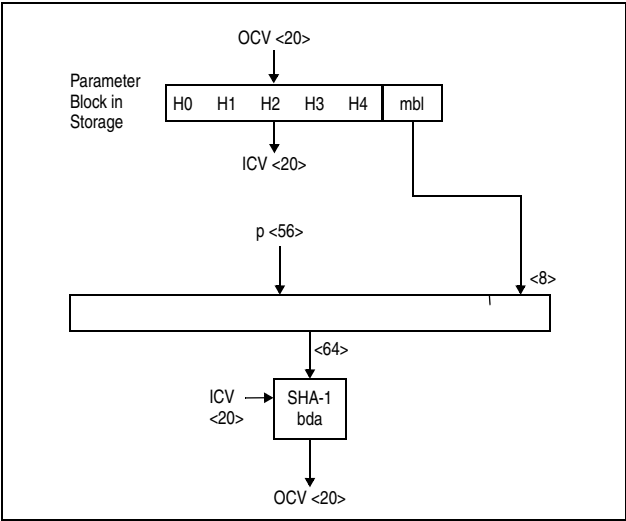


Figure 7-204. KLMD-SHA-1 Empty Block ($L = 0$)

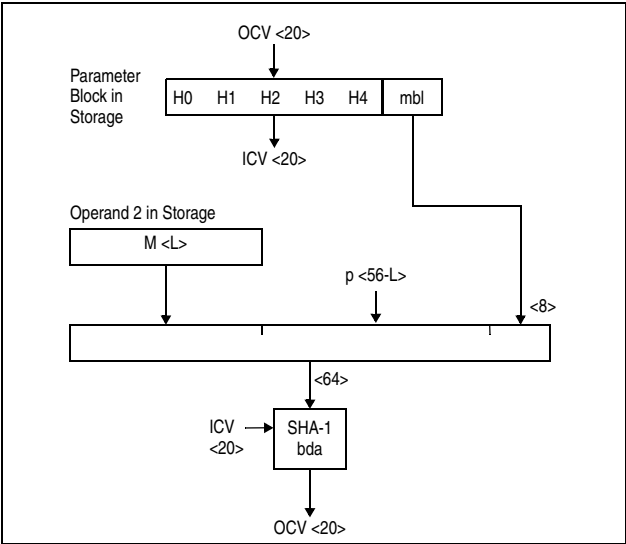


Figure 7-205. KLMD-SHA-1 Partial-Block Case 1 ($1 \leq L \leq 55$)

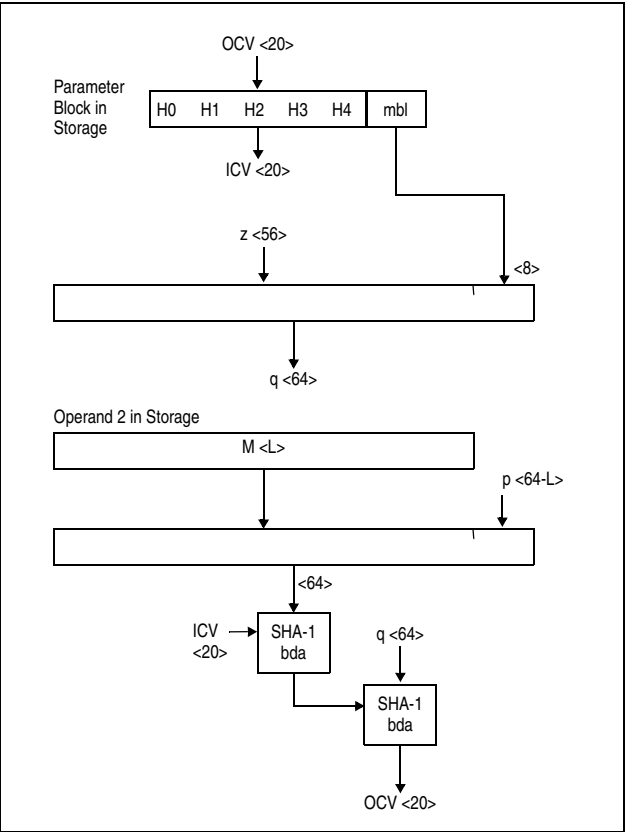


Figure 7-206. KLMD-SHA-1 Partial-Block Case 2 ($56 \leq L \leq 63$)

KLMD-SHA-256 (KLMD Function Code 2)
The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the function has the following format:

0	H0
4	H1
8	H2
12	H3
16	H4
20	H5
24	H6
28	H7
32	Message Bit Length (mbl)
36	

Figure 7-207. Parameter Block for KLMD-SHA-256

The message digest for the message (M) in operand 2 is generated using the SHA-256 algorithm with the chaining value and message-bit-length information in the parameter block.

If the message in operand 2 is equal to or greater than 64 bytes, an intermediate message digest is generated for each 64-byte message block using the SHA-256 block digest algorithm with the 32-byte chaining value in the parameter block, and the generated intermediate message digest, also called the output chaining value (OCV), is stored into the chaining-value field of the parameter block. This operation is shown in Figure 7-208 and repeats until the remaining message is less than 64 bytes.

If the length of the message or the remaining message is zero byte, then the operation in Figure 7-209 is performed. If the length of the message or the remaining message is between one byte and 55 bytes inclusive, then the operation in Figure 7-210 is performed; if the length is between 56 bytes and 63 bytes inclusive, then the operation in Figure 7-211 is performed. The message digest, also called the output chaining value (OCV), is stored into the chaining-value field of the parameter block.

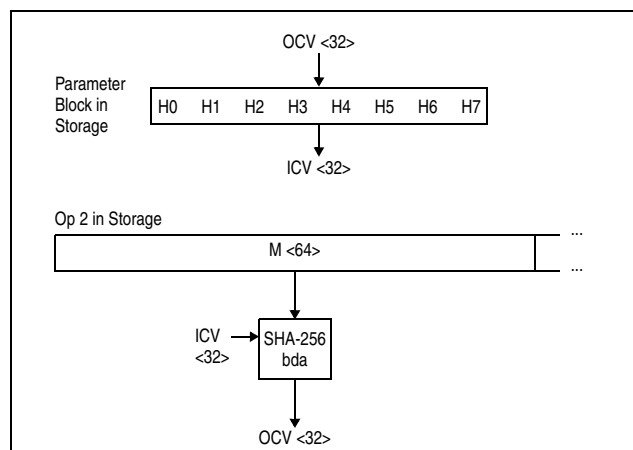


Figure 7-208. KLMD-SHA-256 Full Block ($L \geq 64$)

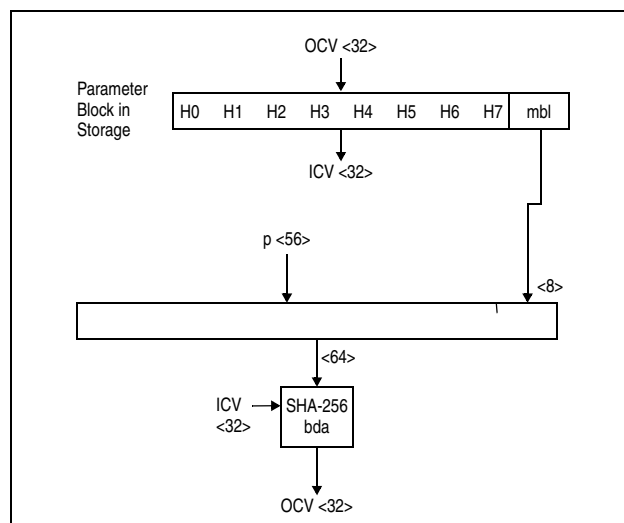


Figure 7-209. KLMD-SHA-256 Empty Block ($L = 0$)

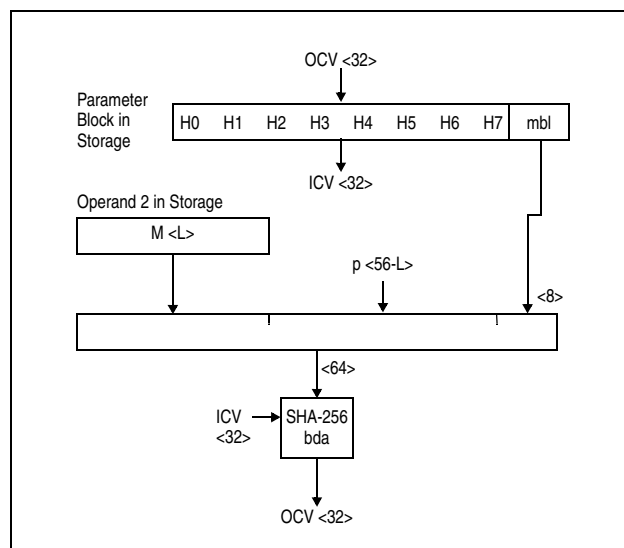
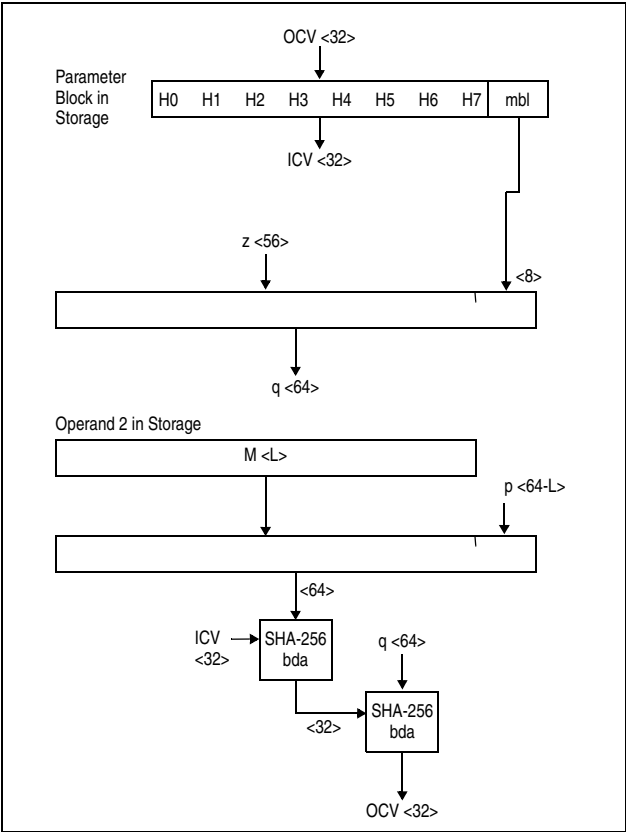


Figure 7-210. KLMD-SHA-256 Partial-Block Case 1 ($1 \leq L \leq 55$)



KLMD-SHA-512 (KLMD Function Code 3)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-187 on page 7-161.

The parameter block used for the function has the following format:

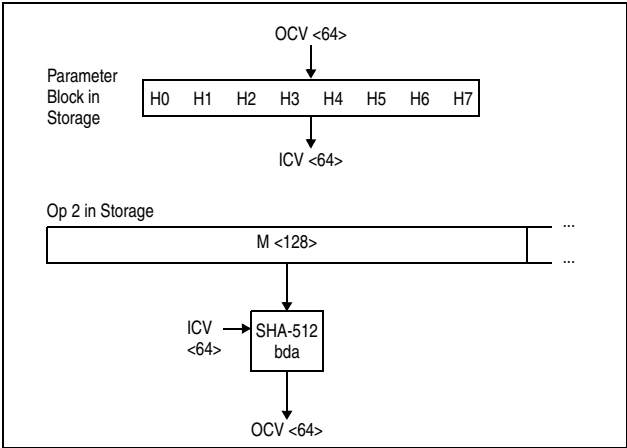
0	H0
8	H1
16	H2
24	H3
32	H4
40	H5
48	H6
56	H7
64	Message Bit Length (mbl)
72	

Figure 7-212. Parameter Block for KLMD-SHA-512

The message digest for the message (M) in operand 2 is generated using the SHA-512 algorithm with the chaining value and message-bit-length information in the parameter block.

If the message in operand 2 is equal to or greater than 128 bytes, an intermediate message digest is generated for each 128-byte message block using the SHA-512 block digest algorithm with the 64-byte chaining value in the parameter block, and the generated intermediate message digest, also called the output chaining value (OCV), is stored into the chaining-value field of the parameter block. This operation is shown in Figure 7-213 and repeats until the remaining message is less than 128 bytes.

If the length of the message or the remaining message is zero byte, then the operation in Figure 7-214 is performed. If the length of the message or the remaining message is between one byte and 111 bytes inclusive, then the operation in Figure 7-215 is performed; if the length is between 112 bytes and 127 bytes inclusive, then the operation in Figure 7-216 is performed. The message digest, also called the output chaining value (OCV), is stored into the chaining-value field of the parameter block.



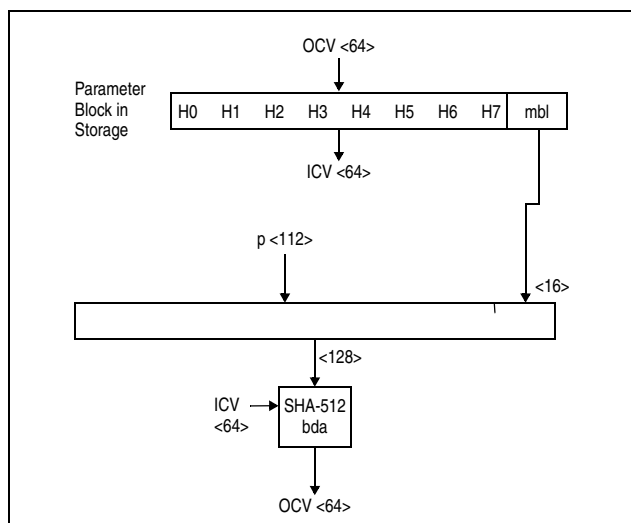


Figure 7-214. KLMD-SHA-512 Empty Block ($L = 0$)

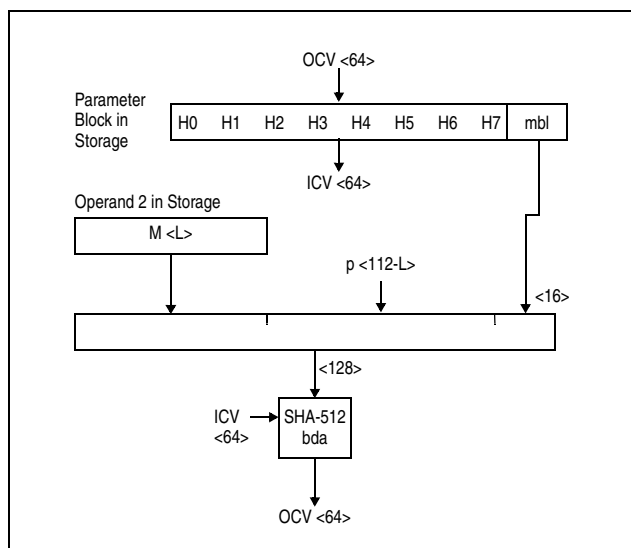


Figure 7-215. KLMD-SHA-512 Partial-Block Case 1 ($1 \leq L \leq 111$)

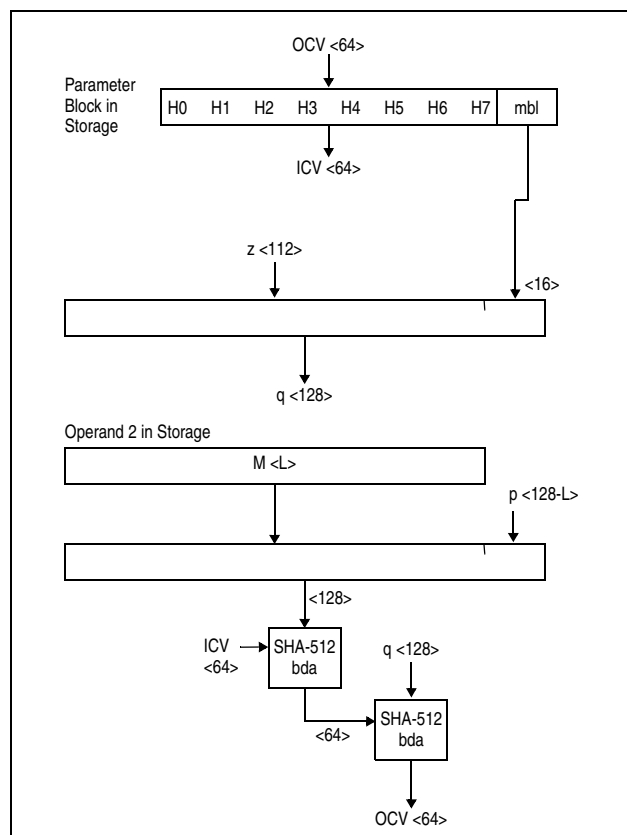


Figure 7-216. KLMD-SHA-512 Partial-Block Case 2 ($112 \leq L \leq 127$)

Special Conditions for KIMD and KLMD

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bit 56 of general register 0 is not zero.
2. Bits 57-63 of general register 0 specify an unassigned or uninstalled function code.
3. The R_2 field designates an odd-numbered register or general register 0.
4. For COMPUTE INTERMEDIATE MESSAGE DIGEST, the second-operand length is not a multiple of the data block size of the designated function (see Figure 7-185 on page 7-159 for COMPUTE INTERMEDIATE MESSAGE DIGEST functions). This specification-exception condition does not apply to the query function, nor does it apply to COMPUTE LAST MESSAGE DIGEST.

Resulting Condition Code:

- 0 Normal completion
- 1 --
- 2 --
- 3 Partial completion

Program Exceptions:

- Access (fetch, operand 2 and message bit length; fetch and store, chaining value)
- Operation (if the message-security assist is not installed)
- Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Operation exception.
8.	Specification exception due to invalid function code or invalid register number.
9.	Specification exception due to invalid operand length.
10.	Condition code 0 due to second-operand length originally zero.
11.	Access exceptions for an access to the parameter block or second operand.
12.	Condition code 0 due to normal completion (second-operand length originally nonzero, but stepped to zero).
13.	Condition code 3 due to partial completion (second-operand length still nonzero).

Figure 7-217. Priority of Execution: KIMD and KLMD

Programming Notes:

1. Bit 56 of general register 0 is reserved for future extension and should be set to zero.
2. When condition code 3 is set, the second operand address and length in general registers R₂ and R₂ + 1, respectively, and the chaining-value in the parameter block are usually updated such

that the program can simply branch back to the instruction to continue the operation.

For unusual situations, the CPU protects against endless reoccurrence for the no-progress case. Thus, the program can safely branch back to the instruction whenever condition code 3 is set with no exposure to an endless loop.

3. If the length of the second operand is nonzero initially and condition code 0 is set, the registers are updated in the same manner as for condition code 3; the chaining value in this case is such that additional operands can be processed as if they were part of the same chain.
4. The instructions COMPUTE INTERMEDIATE MESSAGE DIGEST and COMPUTE LAST MESSAGE DIGEST are designed to be used by a security service application programming interface (API). These APIs provide the program with means to compute the digest of messages of almost unlimited size, including those too large to fit in storage all at once. This is accomplished by permitting the program to pass the message to the API in parts. The following programming notes are described in terms of these APIs.
5. Before processing the first part of a message, the program must set the initial values for the chaining-value field. For SHA-1, the initial hexadecimal chaining values are listed as follows:

H0 = 6 7 4 5 2 3 0 1
H1 = E F C D A B 8 9
H2 = 9 8 B A D C F E
H3 = 1 0 3 2 5 4 7 6
H4 = C 3 D 2 E 1 F 0

For SHA-256, the initial hexadecimal chaining values are listed as follows:

H0 = 6 A 0 9 E 6 6 7
H1 = B B 6 7 A E 8 5
H2 = 3 C 6 E F 3 7 2
H3 = A 5 4 F F 5 3 A
H4 = 5 1 0 E 5 2 7 F
H5 = 9 B 0 5 6 8 8 C
H6 = 1 F 8 3 D 9 A B
H7 = 5 B E 0 C D 1 9

For SHA-512, the initial hexadecimal chaining value are listed as follows:

```
H0 = 6A09 E667 F3BC C908
H1 = BB67 AE85 84CA A73B
H2 = 3C6E F372 FE94 F82B
H3 = A54F F53A 5F1D 36F1
H4 = 510E 527F ADE6 82D1
H5 = 9B05 688C 2B3E 6C1F
H6 = 1F83 D9AB FB41 BD6B
H7 = 5BE0 CD19 137E 2179
```

6. When processing message parts other than the last, the program must process message parts in multiples of the data block size and use the COMPUTE INTERMEDIATE MESSAGE DIGEST instruction.
7. When processing the last message part, the program must compute the length of the original message in bits and place this length value in the message-bit-length field of the parameter block, and use the COMPUTE LAST MESSAGE DIGEST instruction.
8. For SHA functions, the COMPUTE LAST MESSAGE DIGEST instruction does not require the second operand to be a multiple of the data block size. It first processes complete blocks, and may set condition code 3 before processing all blocks. After processing all complete blocks, it then performs the padding operation including the remaining portion of the second operand. This may require one or two iterations of the designated block digest algorithm.
9. For SHA functions, the COMPUTE LAST MESSAGE DIGEST instruction provides the SHA padding for messages that are a multiple of eight bits in length. If a SHA function is to be applied to a bit string which is not a multiple of eight bits, the program must perform the SHA padding and use the COMPUTE INTERMEDIATE MESSAGE DIGEST instruction.
10. The SHA-224 algorithm is the same as the SHA-256 algorithm, except that the initial chaining values and the final message digest lengths are different. The program may obtain the SHA-224

message digest using the SHA-256 functions with the following two actions:

- a. The following initial hexadecimal chaining values for SHA-224 are used:

```
H0 = C105 9ED8
H1 = 367C D507
H2 = 3070 DD17
H3 = F70E 5939
H4 = FFC0 0B31
H5 = 6858 1511
H6 = 64F9 8FA7
H7 = BEFA 4FA4
```

- b. The 224-bit message digest is obtained by truncating the final message digest to its leftmost 224 bits.

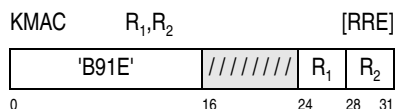
11. The SHA-384 algorithm is the same as the SHA-512 algorithm, except that the initial chaining values and the final message digest lengths are different. The program may obtain the SHA-384 message digest using the SHA-512 functions with the following two actions:

- a. The following initial hexadecimal chaining values for SHA-384 are used:

```
H0 = CBBB 9D5D C105 9ED8
H1 = 629A 292A 367C D507
H2 = 9159 015A 3070 DD17
H3 = 152F ECD8 F70E 5939
H4 = 6733 2667 FFC0 0B31
H5 = 8EB4 4A87 6858 1511
H6 = DB0C 2E0D 64F9 8FA7
H7 = 47B5 481D BEFA 4FA4
```

- b. The 384-bit message digest is obtained by truncating the final message digest to its leftmost 384 bits.

COMPUTE MESSAGE AUTHENTICATION CODE



A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction and the R_1 field are ignored.

Bit positions 57-63 of general register 0 contain the function code. Figure 7-218 shows the assigned function codes. All other function codes are unassigned. Bit 56 of general register 0 must be zero; otherwise, a specification exception is recognized. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

The function codes for COMPUTE MESSAGE AUTHENTICATION CODE are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	KMAC-Query	16	—
1	KMAC-DEA	16	8
2	KMAC-TDEA-128	24	8
3	KMAC-TDEA-192	32	8
9	KMAC-Encrypted-DEA	40	8
10	KMAC-Encrypted-TDEA-128	48	8
11	KMAC-Encrypted-TDEA-192	56	8
18	KMAC-AES-128	32	16
19	KMAC-AES-192	40	16
20	KMAC-AES-256	48	16
26	KMAC-Encrypted-AES-128	64	16
27	KMAC-Encrypted-AES-192	72	16
28	KMAC-Encrypted-AES-256	80	16
Explanation:			
—	Not applicable		

Figure 7-218. Function Codes for COMPUTE MESSAGE AUTHENTICATION CODE

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions. The contents of general registers R_2 and $R_2 + 1$ are ignored.

For all other functions, the second operand is processed as specified by the function code using an initial chaining value and the result replaces the initial chaining value. The operation also uses a cryptographic key. The operation proceeds until the end of the second-operand location is reached or a CPU-determined number of bytes have been processed, whichever occurs first. The result is indicated in the condition code.

The R_2 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the second operand is specified by the contents of the R_2 general register. The number of bytes in the second-operand location is specified in general register $R_2 + 1$.

As part of the operation, the address in general register R_2 is incremented by the number of bytes processed from the second operand, and the length in general register $R_2 + 1$ is decremented by the same number. The formation and updating of the address and length is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general register R_2 constitute the address of second operand, and the contents of bit positions 0-39 are ignored; bits 40-63 of the updated address replace the corresponding bits in general register R_2 , carries out of bit position 40 of the updated address are ignored and, the contents of bit positions 32-39 of general register R_2 are set to zeros. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register R_2 constitute the address of second operand, and the contents of bit positions 0-32 are ignored; bits 33-63 of the updated address replace the corresponding bits in general register R_2 , carries out of bit position 33 of the updated address are ignored, and the content of bit position 32 of general register R_2 is set to zero. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register R_2 constitute the address of second operand; bits 0-63 of the updated address replace the contents of general register R_2 and carries out of bit position 0 are ignored.

In both the 24-bit and the 31-bit addressing modes, the contents of bit positions 32-63 of general register $R_2 + 1$ form a 32-bit unsigned binary integer which specifies the number of bytes in the second operand; and the updated value replaces the contents of bit positions 32-63 of general register $R_2 + 1$. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register $R_2 + 1$ form a 64-bit unsigned binary integer which specifies the number of bytes in

the second operand; and the updated value replaces the contents of general register $R_2 + 1$.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_2 and $R_2 + 1$, always remain unchanged.

Figure 7-219 on page 7-173 shows the contents of the general registers just described.

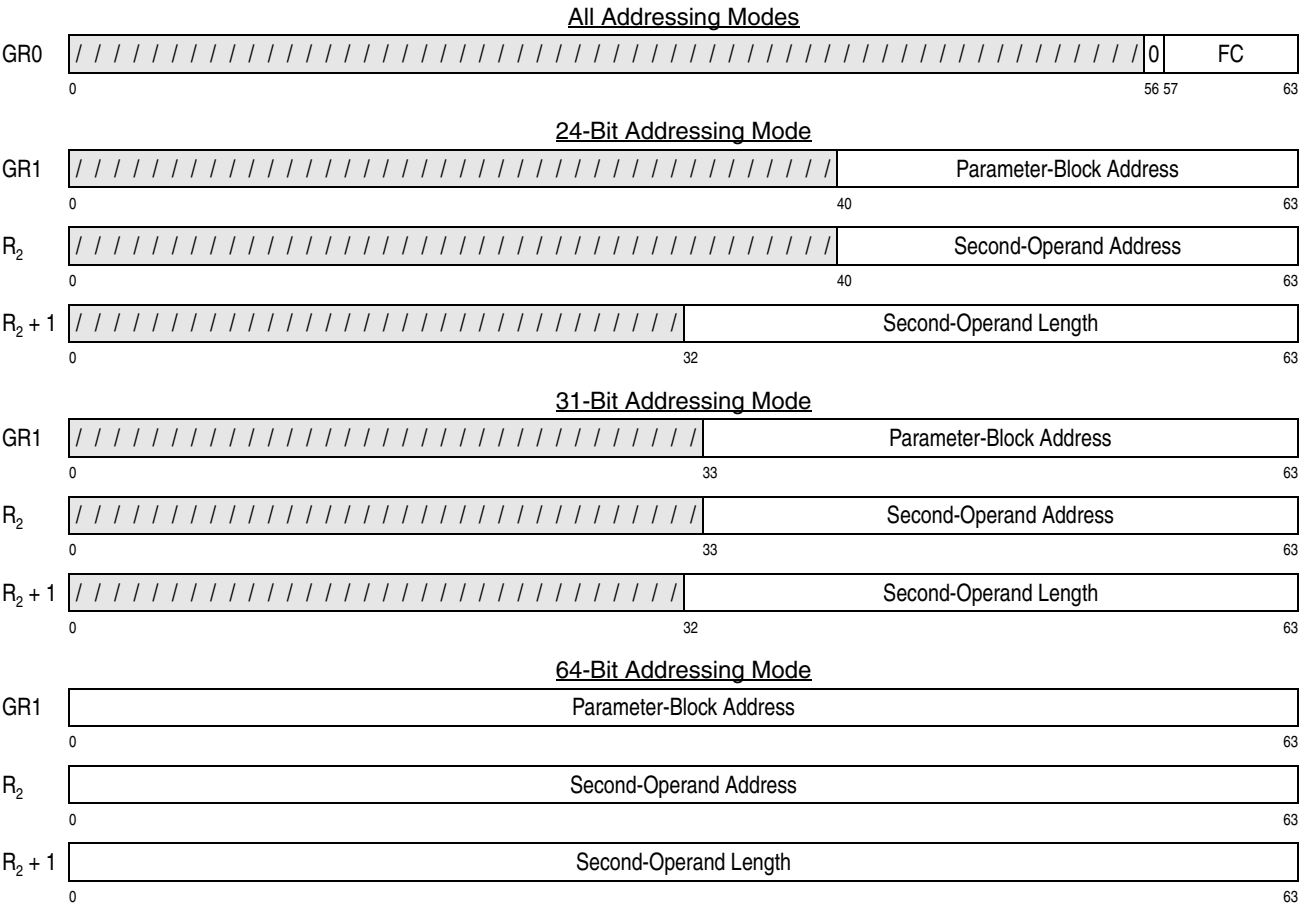


Figure 7-219. General Register Assignment for KMAC

In the access-register mode, access registers 1 and R_2 specify the address spaces containing the parameter block and second operand, respectively.

The result is obtained as if processing starts at the left end of the second operand and proceeds to the right, block by block. The operation is ended when all source bytes in the second operand have been processed (called normal completion), or when a CPU-determined number of blocks that is less than the length of the second operand have been processed (called partial completion). The CPU-determined

number of blocks depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of blocks is usually nonzero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless reoccurrence of this no-progress case.

When the initial-chaining-value field overlaps any portion of the second operand, the result in the chaining-value field is unpredictable.

Normal completion occurs when the number of bytes in the second operand as specified in general register $R_2 + 1$ have been processed.

When the operation ends due to normal completion, condition code 0 is set and the resulting value in $R_2 + 1$ is zero. When the operation ends due to partial completion, condition code 3 is set and the resulting value in $R_2 + 1$ is nonzero.

When the second-operand length is initially zero, the second operand and the parameter block are not accessed, general registers R_2 and $R_2 + 1$ are not changed, and condition code 0 is set.

As observed by other CPUs and channel programs, references to the parameter block and storage operands may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

Access exceptions may be reported for a larger portion of the second operand than is processed in a single execution of the instruction; however, access exceptions are not recognized for locations beyond the length of the second operand nor for locations more than 4K bytes beyond the current location being processed.

Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the COMPUTE MESSAGE AUTHENTICATION CODE functions. For data-encryption-algorithm (DEA) functions, the DEA-key-parity bit in each byte of the DEA key is ignored, and the operation proceeds normally, regardless of the DEA-key parity of the key. Further description of the data-encryption algorithm may be found in *Data Encryption Algorithm*, ANSI-X3.92.1981, American National Standard for Information Systems.

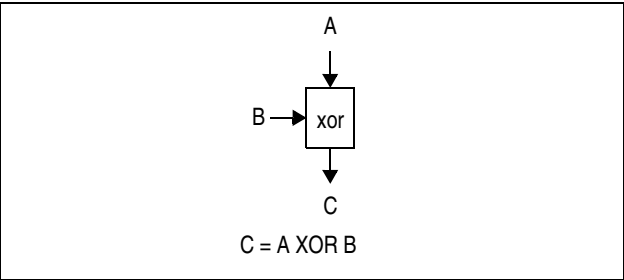


Figure 7-220. Symbol For Bit-Wise Exclusive OR

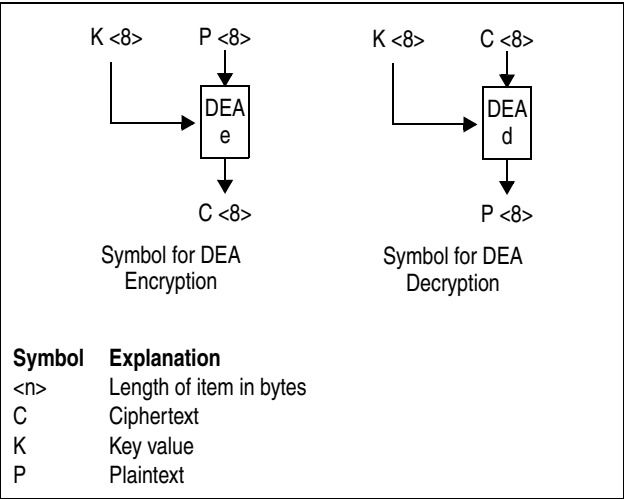


Figure 7-221. Symbols for DEA Encryption and Decryption

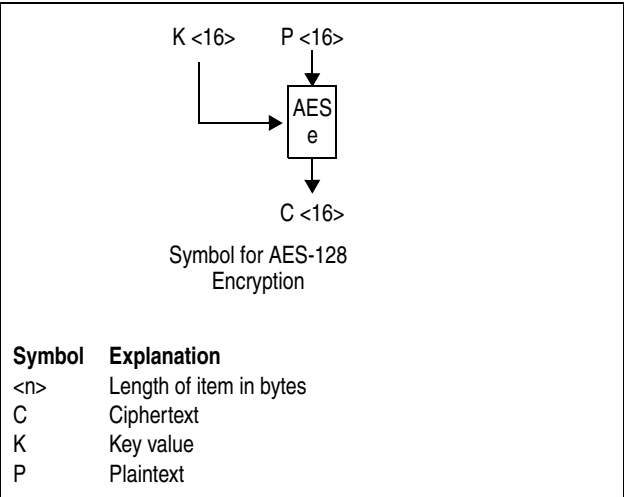


Figure 7-222. Symbols for AES-128 Encryption

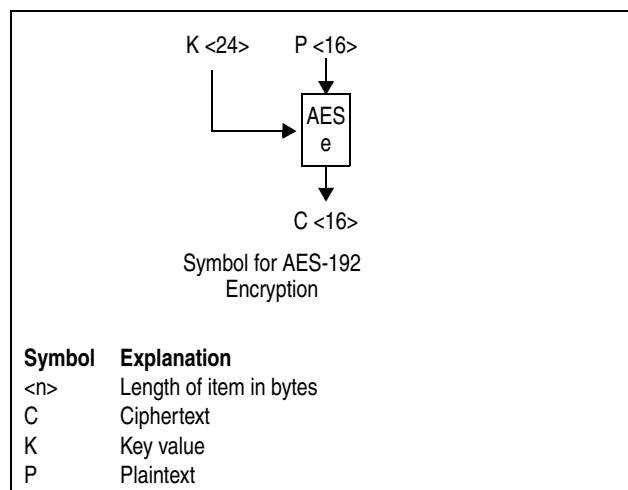


Figure 7-223. Symbols for AES-192 Encryption

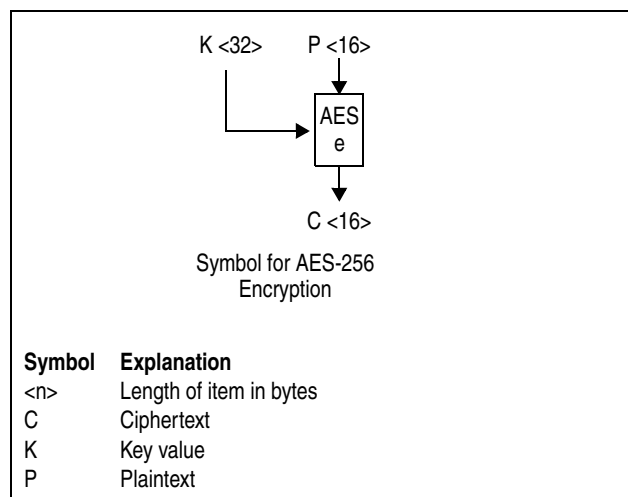


Figure 7-224. Symbols for AES-256 Encryption

KMAC-Query (Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-219 on page 7-173.

The parameter block used for the function has the following format:

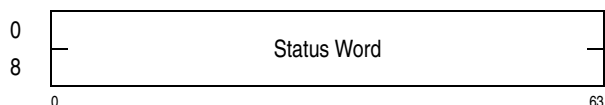


Figure 7-225. Parameter Block for KMAC-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the KMAC instruction.

When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Condition code 0 is set when execution of the KMAC-Query function completes; condition code 3 is not applicable to this function.

KMAC-DEA (Function Code 1)

KMAC-Encrypted-DEA (Function Code 9)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-219 on page 7-173.

The parameter block used for the KMAC-DEA function has the following format:

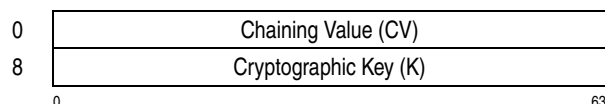


Figure 7-226. Parameter Block for KMAC-DEA

For the KMAC-DEA function, the initial chaining value is in byte offsets 0-7 of the parameter block and the cryptographic key is in byte offsets 8-15 of the parameter block.

The parameter block used for the KMAC-Encrypted-DEA function has the following format:

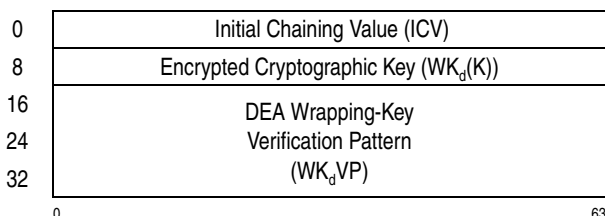


Figure 7-227. Parameter Block for KMAC-Encrypted-DEA

For the KMAC-Encrypted-DEA function, the contents of byte offsets 16-39 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the initial chaining value, and the contents of byte offsets 8-15 of the parameter block are deciphered using the DEA wrapping key to obtain the 64-bit cryptographic key. (See the section, "Protection of Cryptographic Key" on page 7-339, for details.)

The description in the following paragraph applies to both functions.

The message authentication code for the 8-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the DEA algorithm with the 64-bit cryptographic key and the 64-bit initial chaining value.

The message authentication code, also called the output chaining value (OCV), is stored in the initial-chaining-value field of the parameter block. The operation is shown in Figure 7-228.

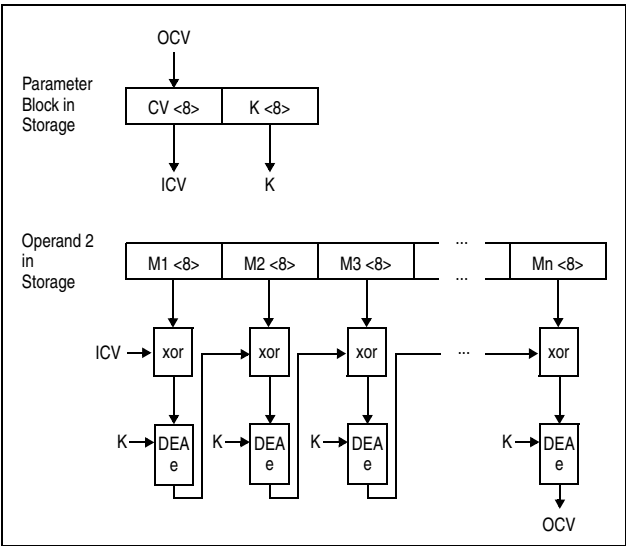


Figure 7-228. MAC Computation Using 64-bit DEA Key

KMAC-TDEA-128 (Function Code 2)

KMAC-Encrypted-TDEA-128 (Function Code 10)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-219 on page 7-173.

The parameter block used for the KMAC-TDEA-128 function has the following format:

0	Initial Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)

Figure 7-229. Parameter Block for KMAC-TDEA-128

For the KMAC-TDEA-128 function, the initial chaining value is in byte offsets 0-7 of the parameter block and the cryptographic key is in byte offsets 8-23 of the parameter block.

The parameter block used for the KMAC-Encrypted-TDEA-128 function has the following format:

0	Initial Chaining Value (ICV)
8	Encrypted Cryptographic Key (WK _e (K))
16	
24	DEA Wrapping-Key Verification Pattern (WK _d VP)
32	
40	

Figure 7-230. Parameter Block for KMAC-Encrypted-TDEA-128

For the KMAC-Encrypted-TDEA-128 function, the contents of byte offsets 24-47 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the initial chaining value, and the contents of byte offsets 8-23 of the parameter block are deciphered using the DEA wrapping key to obtain the 128-bit cryptographic key, K. (See the section, “Protection of Cryptographic Key” on page 7-339, for details.)

The description in the following paragraph applies to both functions.

The message authentication code for the 8-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the TDEA algorithm with the two 64-bit cryptographic keys and the 64-bit initial chaining value.

The message authentication code, also called the output chaining value (OCV), is stored in the initial-

chaining-value field of the parameter block. The operation is shown in Figure 7-231.

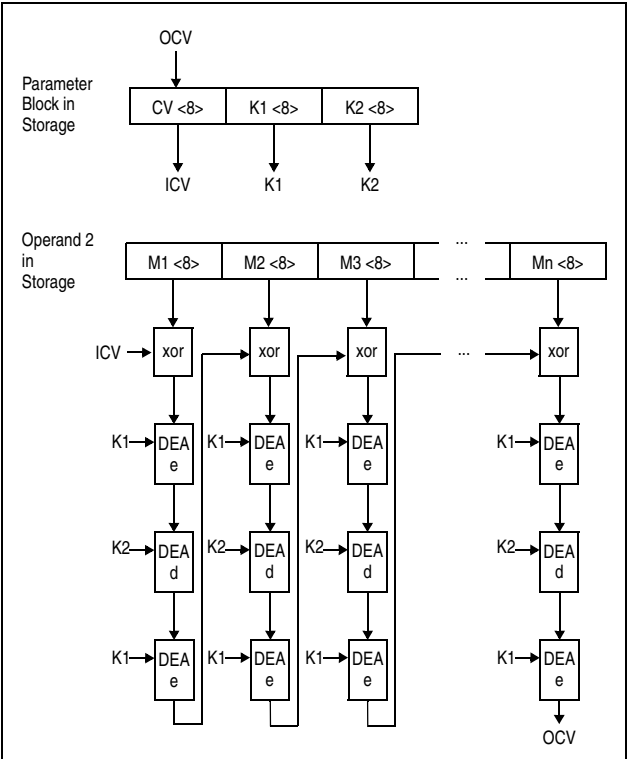


Figure 7-231. MAC Computation Using 128-bit TDEA Key

KMAC-TDEA-192 (Function Code 3)

KMAC-Encrypted-TDEA-192 (Function Code 11)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-219 on page 7-173.

The parameter block used for the KMAC-TDEA-192 function has the following format:

0	Initial Chaining Value (CV)
8	Cryptographic Key 1 (K1)
16	Cryptographic Key 2 (K2)
24	Cryptographic Key 3 (K3)

Figure 7-232. Parameter Block for KMAC-TDEA-192

For the KMAC-TDEA-192 function, the initial chaining value is in byte offsets 0-7 of the parameter block and the cryptographic key is in byte offsets 8-31 of the parameter block.

The parameter block used for the KMAC-Encrypted-TDEA-192 function has the following format:

0	Initial Chaining Value (ICV)
8	
16	Encrypted Cryptographic Key ($WK_e(K)$)
24	
32	DEA Wrapping-Key Verification Pattern ($WK_d(VP)$)
40	
48	
63	

Figure 7-233. Parameter Block for KMAC-Encrypted-TDEA-192

For the KMAC-Encrypted-TDEA-192 function, the contents of byte offsets 32-55 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-7 of the parameter block contain the initial chaining value, and the contents of byte offsets 8-31 of the parameter block are deciphered using the DEA wrapping key to obtain the 192-bit cryptographic key, K. (See the section, "Protection of Cryptographic Key" on page 7-339, for details.)

The description in the following paragraph applies to both functions.

The message authentication code for the 8-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the TDEA algorithm with the three 64-bit cryptographic keys and the 64-bit chaining value.

The message authentication code, also called the output chaining value (OCV), is stored in the initial-

chaining-value field of the parameter block. The operation is shown in Figure 7-234.

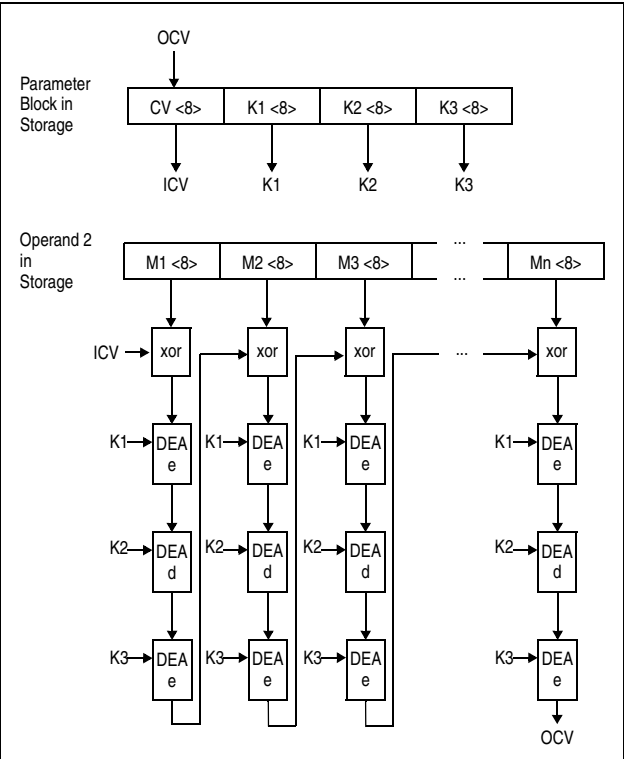


Figure 7-234. MAC Computation Using 192-bit TDEA Key

KMAC-AES-128 (Function Code 18)

KMAC-Encrypted-AES-128 (Function Code 26)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-219 on page 7-173.

The parameter block used for the KMAC-AES-128 function has the following format:

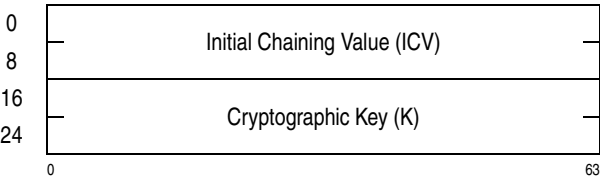


Figure 7-235. Parameter Block for KMAC-AES-128

For the KMAC-AES-128 function, the initial-chaining value is in byte offsets 0-15 of the parameter block and the cryptographic key is in byte offsets 16-31 of the parameter block.

The parameter block used for the KMAC-Encrypted-AES-128 function has the following format:

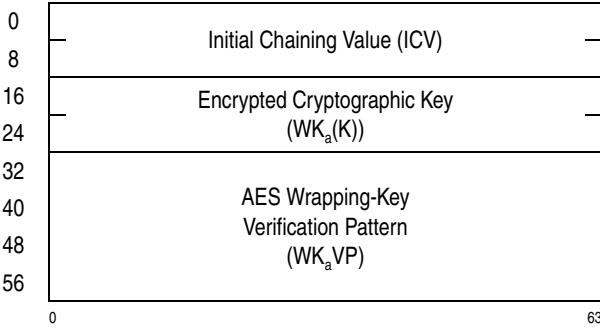


Figure 7-236. Parameter Block for KMAC-Encrypted-AES-128

For the KMAC-Encrypted-AES-128 function, the contents of byte offsets 32-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block contain the initial chaining value, and the contents of byte offsets 16-31 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The description in the following paragraph applies to both functions.

The message authentication code for the 16-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the AES algorithm with the 128-bit cryptographic key and the 128-bit chaining value.

The message authentication code, also called the output chaining value (OCV), is stored in the initial-chaining-value field of the parameter block. The operation is shown in Figure 7-237.

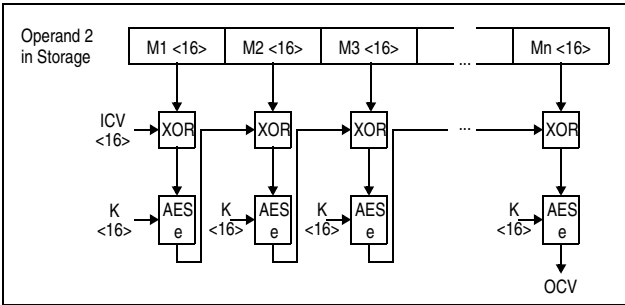


Figure 7-237. KMAC-AES-128

KMAC-AES-192 (Function Code 19)

KMAC-Encrypted-AES-192 (Function Code 27)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-219 on page 7-173.

The parameter block used for the KMAC-AES-192 function has the following format:

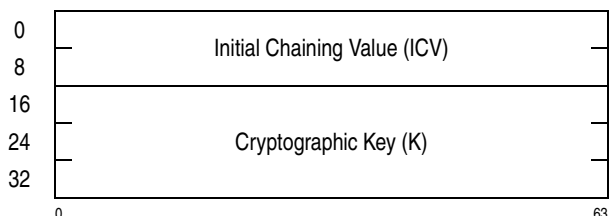


Figure 7-238. Parameter Block for KMAC-AES-192

For the KMAC-AES-192 function, the initial-chaining value is in byte offsets 0-15 of the parameter block and the cryptographic key is in byte offsets 16-39 of the parameter block.

The parameter block used for the KMAC-Encrypted-AES-192 function has the following format:

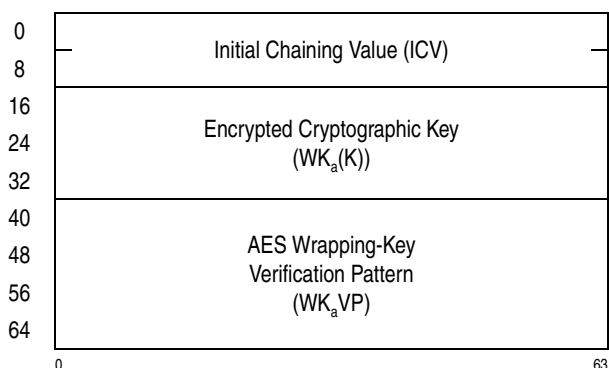


Figure 7-239. Parameter Block for KMAC-Encrypted-AES-192

For the KMAC-Encrypted-AES-192 function, the contents of byte offsets 40-71 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block contain the initial chaining value, and the contents of byte offsets 16-39 of the parameter block are deciphered using the AES wrapping key to obtain the 192-bit cryptographic key, K. (See the section “Pro-

tection of Cryptographic Key” on page 7-339 for details.)

The description in the following paragraph applies to both functions.

The message authentication code for the 16-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the AES algorithm with the 192-bit cryptographic key and the 128-bit chaining value.

The message authentication code, also called the output chaining value (OCV), is stored in the initial-chaining-value field of the parameter block. The operation is shown in Figure 7-240.

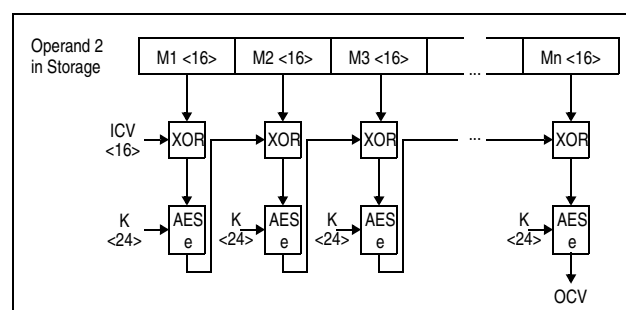


Figure 7-240. KMAC-AES-192

KMAC-AES-256 (Function Code 20)

KMAC-Encrypted-AES-256 (Function Code 28)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-219 on page 7-173.

The parameter block used for the KMAC-AES-256 function has the following format:

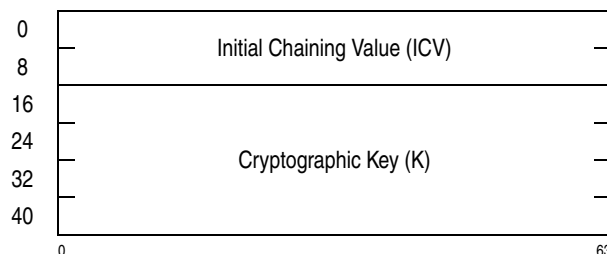


Figure 7-241. Parameter Block for KMAC-AES-256

For the KMAC-AES-256 function, the initial-chaining value is in byte offsets 0-15 of the parameter block and the cryptographic key is in byte offsets 16-47 of the parameter block.

The parameter block used for the KMAC-Encrypted-AES-256 function has the following format:

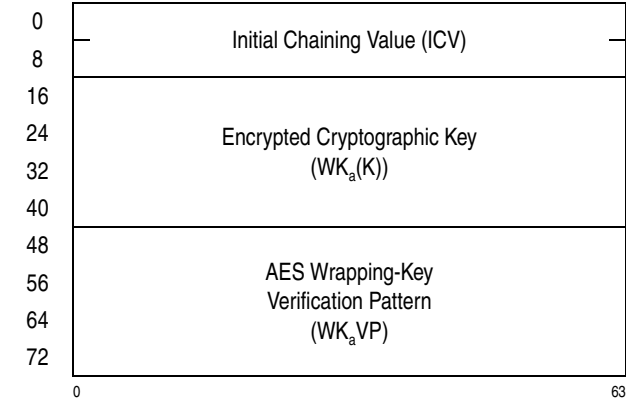


Figure 7-242. Parameter Block for KMAC-Encrypted-AES-256

For the KMAC-Encrypted-AES-256 function, the contents of byte offsets 48-79 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 0-15 of the parameter block contain the initial chaining value, and the contents of byte offsets 16-47 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The description in the following paragraph applies to both functions.

The message authentication code for the 16-byte message blocks (M1, M2, ..., Mn) in operand 2 is computed using the DEA algorithm with the 256-bit cryptographic key and the 128-bit chaining value.

The message authentication code, also called the output chaining value (OCV), is stored in the initial-

chaining-value field of the parameter block. The operation is shown in Figure 7-243.

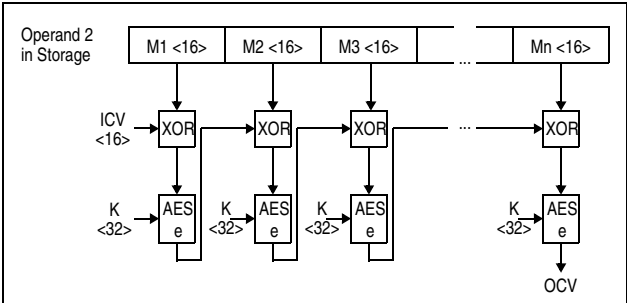


Figure 7-243. KMAC-AES-256

Special Conditions

A specification exception is recognized and no other action is taken if any of the following occurs:

- 1. Bit 56 of general register 0 is not zero.
- 2. Bits 57-63 of general register 0 specify an unsigned or uninstalled function code.
- 3. The R₂ field designates an odd-numbered register or general register 0.
- 4. The second-operand length is not a multiple of the data block size of the designated function (see Figure 7-218 on page 7-172 to determine the data block size for COMPUTE MESSAGE AUTHENTICATION CODE functions).

Resulting Condition Code:

- 0 Normal completion
- 1 Verification pattern mismatch
- 2 --
- 3 Partial completion

Program Exceptions:

- Access (fetch, operand 2, cryptographic key; fetch and store, chaining value)
- Operation (if the message-security assist is not installed)

• Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Operation exception.
8.	Specification exception due to invalid function code or invalid register number.
9.	Specification exception due to invalid operand length.
10.	Condition code 0 due to second-operand length originally zero.
11.	Access exceptions for an access to the parameter block or second operand.
12.	Condition code 1 due to verification pattern mismatch.
13.	Condition code 0 due to normal completion (second-operand length originally nonzero, but stepped to zero).
14.	Condition code 3 due to partial completion (second-operand length still nonzero).

Figure 7-244. Priority of Execution: KMAC

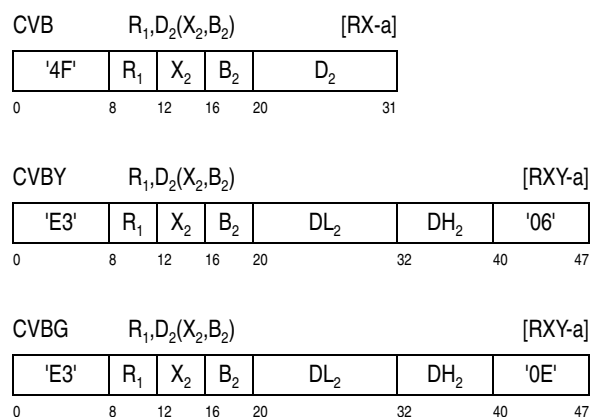
Programming Notes:

- When condition code 3 is set, the second operand address and length in general registers R_2 and $R_2 + 1$, respectively, and the initial-chaining-value in the parameter block are usually updated such that the program can simply branch back to the instruction to continue the operation. For unusual situations, the CPU protects against endless reoccurrence for the no-progress case. Thus, the program can safely branch back to the instruction whenever condition code 3 is set with no exposure to an endless loop.
- If the length of the second operand is nonzero initially and condition code 0 is set, the registers are updated in the same manner as for condition code 3; the initial chaining value in this case is such that additional operands can be processed as if they were part of the same chain.
- Before processing the first part of a message, the program must set the initial values for the initial-chaining-value field. To comply with ANSI

X9.9 or X9.19, the initial chaining value shall be set to all binary zeros. To comply with ANSI X9.9, ANSI X9.19, or NIST Special Publication 800-38D, the initial chaining value shall be set to all binary zeros.

- For the KMAC-GHASH function, the hash subkey, H, is the result of encrypting a 128-bit zero using the AES encryption algorithm with a 128-bit, 192-bit, or 256-bit cryptographic key.
-

CONVERT TO BINARY



The second operand is changed from decimal to binary, and the result is placed at the first-operand location.

For CONVERT TO BINARY (CVB, CVBY), the second operand occupies eight bytes in storage, and, for CONVERT TO BINARY (CVBG), the second operand occupies sixteen bytes in storage. The second operand has the format of signed-packed-decimal data, as described in Chapter 8, "Decimal Instructions." It is checked for valid sign and digit codes, and a decimal-operand data exception is recognized when an invalid code is detected.

For CONVERT TO BINARY (CVB, CVBY), the result of the conversion is a 32-bit signed binary integer, which is placed in bit positions 32-63 of general register R_1 . Bits 0-31 of the register remain unchanged. The maximum positive number that can be converted and still be contained in 32 bit positions is 2,147,483,647; the maximum negative number (the negative number with the greatest absolute value) that can be converted is -2,147,483,648.

For any decimal number outside this range, the operation is completed by placing the 32 rightmost bits of the binary result in the register, and a fixed-point-divide exception is recognized.

For CONVERT TO BINARY (CVBG), the result of the conversion is a 64-bit signed binary integer, which is placed in bit positions 0-63 of general register R₁. The maximum positive number that can be converted and still be contained in a 64-bit register is 9,223,372,036,854,775,807; the maximum negative number (the negative number with the greatest absolute value) that can be converted is -9,223,372,036,854,775,808. For any decimal number outside this range, a fixed-point-divide exception is recognized, and the operation is suppressed.

The displacement for CVB is treated as a 12-bit unsigned binary integer. The displacement for CVBY and CVBG is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

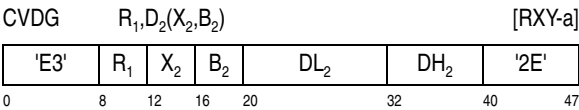
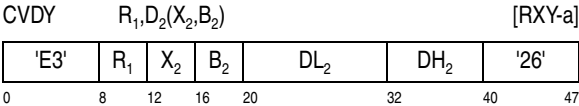
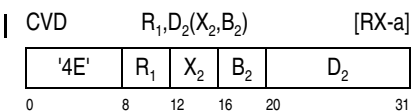
Program Exceptions:

- Access (fetch, operand 2)
- Data
- Fixed-point divide
- Operation (CVBY, if the long-displacement facility is not installed)

Programming Notes:

1. An example of the use of the CONVERT TO BINARY instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When the second operand is negative, the result is in two's-complement notation.
3. The storage-operand references for CONVERT TO BINARY may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)

CONVERT TO DECIMAL



The first operand is changed from binary to decimal, and the result is stored at the second-operand location.

For CONVERT TO DECIMAL (CVD, CVDY), the first operand is treated as a 32-bit signed binary integer, and the result occupies eight bytes in storage. For CONVERT TO DECIMAL (CVDG), the first operand is treated as a 64-bit signed binary integer, and the result occupies sixteen bytes in storage.

The result is in the format for signed-packed-decimal data, as described in Chapter 8, "Decimal Instructions." The rightmost four bits of the result represent the sign. A positive sign is encoded as 1100; a negative sign is encoded as 1101.

The displacement for CVD is treated as a 12-bit unsigned binary integer. The displacement for CVDY and CVDG is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (CVDY, if the long-displacement facility is not installed)

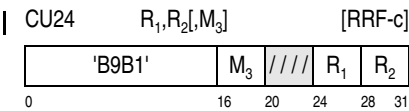
Programming Notes:

1. An example of the use of the CONVERT TO DECIMAL instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. For CVD and CVDY, the number to be converted is a 32-bit signed binary integer obtained from a general register. Since 15 decimal digits are available for the result, and the decimal equivalent of 31 bits requires at most 10 decimal digits, an overflow cannot occur. Similarly, for CVDG, 31 decimal digits are available, the decimal equivalent of 63 bits requires at most 20 decimal digits, an overflow cannot occur.

lent of 63 bits is at most 19 digits, and an overflow cannot occur.

- 3. The storage-operand references for CONVERT TO DECIMAL may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

CONVERT UTF-16 TO UTF-32



The two-byte UTF-16 (Unicode) characters of the second operand are converted to UTF-32 characters and placed at the first-operand location. The UTF-32 characters are four bytes. The operation proceeds until the end of the first or second operand is reached, a CPU-determined number of characters have been converted, or an invalid Unicode character is encountered, whichever occurs first. The result is indicated in the condition code.

The R₁ and R₂ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and the second operand is designated by the contents of general registers R₁ and R₂, respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the contents of bit positions 32-63 of general registers R₁ + 1 and R₂ + 1, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the entire contents of general registers R₁ + 1 and R₂ + 1, respectively, and those contents are treated as 64-bit unsigned binary integers.

The handling of the addresses in general registers R₁ and R₂ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R₁ and R₂ constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-245 on page 7-184.

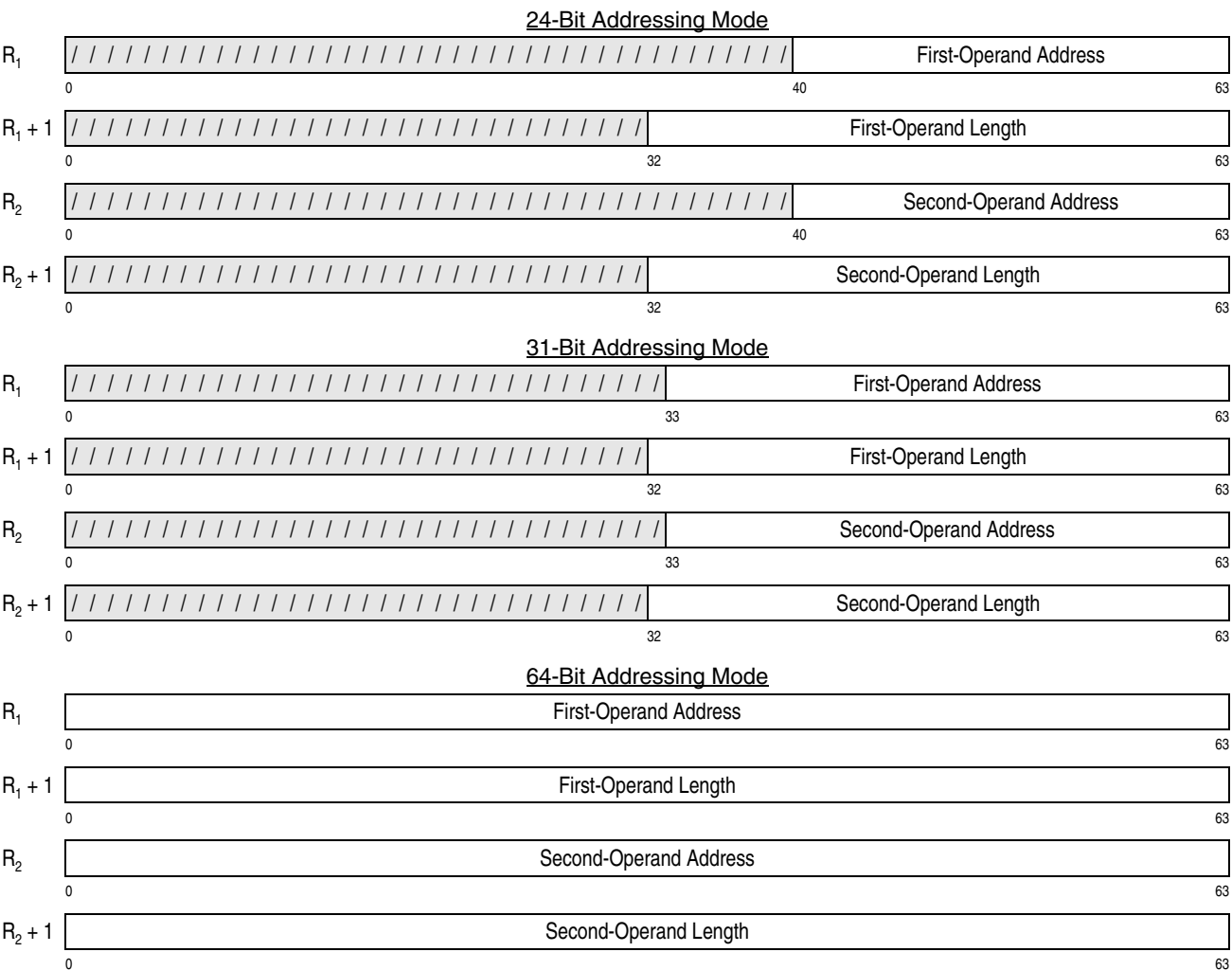
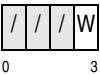


Figure 7-245. Register Contents for CONVERT UTF-16 TO UTF-32

When the ETF3-enhancement facility is installed, the M_3 field has the following format:



The bits of the M_3 field are defined as follows:

- Unassigned:** Bits 0-2 are unassigned and should contain zeros; otherwise, the program may not operate compatibly in the future.
- Enhanced Well-Formedness-Checking (W):** The W bit, bit 3 of the M_3 field, controls whether enhanced well-formedness checking of the Unicode characters is performed. When the W bit is

zero, enhanced checking is not performed. When the W bit is one, enhanced checking is performed, as described below.

When the ETF3-enhancement facility is not installed, the M_3 field is ignored.

The characters of the second operand are selected one by one for conversion, proceeding left to right. The characters resulting from a conversion are placed at the first-operand location, proceeding left to right. The operation proceeds until the first-operand or second-operand location is exhausted or a CPU-determined number of second-operand characters have been converted.

To show the method of converting a UTF-16 character to a UTF-32 character, the bits of a Unicode character are identified by letters as follows:

Unicode Character Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	a b c d e f g h	i j k l m n o p

In the case of a Unicode surrogate pair, which is a character pair consisting of a character called a high surrogate followed by a character called a low surrogate, the bits are identified by letters as follows:

Unicode High Surrogate Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	1 1 0 1 1 0 a b	c d e f g h i j

Unicode Low Surrogate Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	1 1 0 1 1 1 k l	m n o p q r s t

Any Unicode character in the range 0000 to D7FF and DC00 to FFFF hex is converted to a four-byte UTF-32 character as follows:

Unicode Character	a b c d e f g h	i j k l m n o p		
UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	a b c d e f g h	i j k l m n o p

Any Unicode surrogate pair starting with a high surrogate in the range D800 to DBFF hex is converted to a four-byte UTF-32 character as follows:

Unicode Characters	1 1 0 1 1 0 a b	c d e f g h i j	1 1 0 1 1 1 k l	m n o p q r s t
UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 u v w x y	e f g h i j k l	m n o p q r s t
where uvwxy = abcd + 1				

When the ETF3-enhancement facility is not installed, or when the W bit of the M₃ field is zero, the first six bits of the Unicode low surrogate are ignored. When the ETF3-enhancement facility is installed, and the W bit is one, the first six bits of the Unicode low surrogate must contain 110111 binary; otherwise, the Unicode low surrogate is invalid, and condition code 2 is set.

The second-operand location is considered exhausted when it does not contain at least two remaining bytes or at least four remaining bytes when the first two bytes are a Unicode high surro-

gate. The first-operand location is considered exhausted when it does not contain at least four remaining bytes.

When the second-operand location is exhausted, condition code 0 is set. When the first-operand location is exhausted, condition code 1 is set, except that condition code 0 is set if the second-operand location also is exhausted. When a CPU-determined number of characters have been converted, condition code 3 is set.

When the conditions for setting condition codes 1 and 2 are both met, condition code 2 is set.

When the operation is completed, the contents of general register R₂ + 1 are decremented by the number of bytes converted, and the contents of general register R₂ are incremented by the same number. Also, the contents of general register R₁ + 1 are decremented by the number of bytes placed at the first-operand location, and the contents of general register R₁ are incremented by the same number. When general registers R₁ and R₂ are updated in the 24-bit or 31-bit addressing mode, bits 32-39 of them, in the 24-bit mode, or bit 32, in the 31-bit mode, may be set to zeros or may remain unchanged.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R₁, R₁ + 1, R₂, and R₂ + 1, always remain unchanged.

When condition code 2 is set, general register R2 contains the address of the invalid Unicode high surrogate that precedes the invalid low surrogate.

When condition code 3 is set, the registers have been updated so that the instruction, when reexecuted, resumes at the next byte locations to be processed.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the R₁ register is the same register as the R₂ register, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portions of the operands to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

- 0 Entire second operand processed
- 1 End of first operand reached
- 2 Invalid Unicode low surrogate
- 3 CPU-determined number of characters converted

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 3 is not installed)
- Specification

Programming Notes:

- When condition code 3 is set, the program can simply branch back to the instruction to continue the conversion. The program need not determine the number of first-operand or second-operand bytes that were processed.
- The storage-operand references of CONVERT UTF-16 TO UTF-32 may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
- The CONVERT UTF-16 TO UTF-32 instruction supports UTF-16 and UTF-32 characters only in the big-endian encoding.

CONVERT UTF-16 TO UTF-8
 CONVERT UNICODE TO UTF-8



The two-byte Unicode characters of the second operand are converted to UTF-8 characters and placed at the first-operand location. The UTF-8 characters are one, two, three, or four bytes, depending on the Unicode characters that are converted. The operation proceeds until the end of the first or second operand is reached or a CPU-determined number of characters have been converted, whichever occurs first. The result is indicated in the condition code.

The R₁ and R₂ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and the second operand is designated by the contents of general registers R₁ and R₂, respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the contents of bit positions 32-63 of general registers R₁ + 1 and R₂ + 1, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the entire contents of general registers R₁ + 1 and R₂ + 1, respectively, and those contents are treated as 64-bit unsigned binary integers.

The handling of the addresses in general registers R₁ and R₂ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R₁ and R₂ constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-246 on page 7-187.

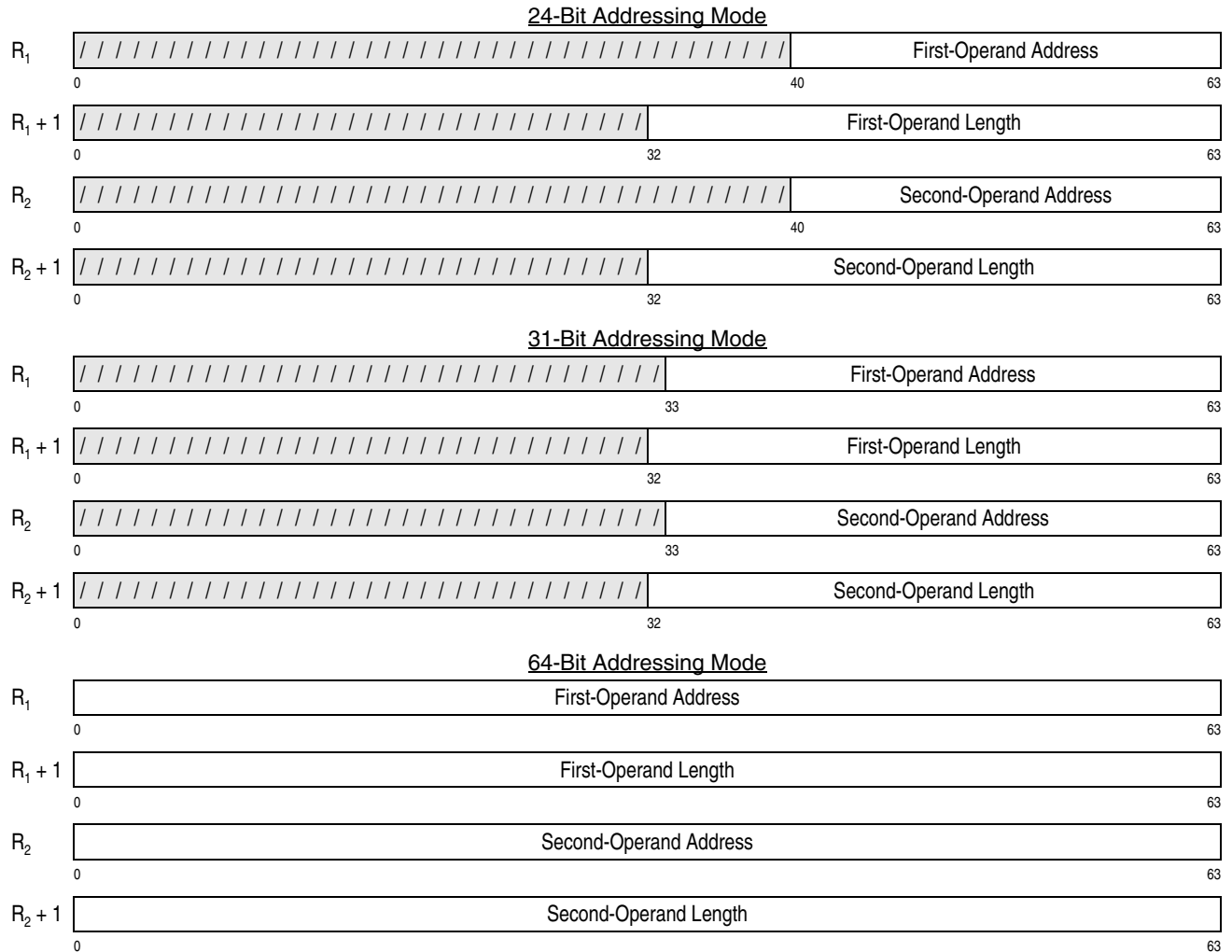
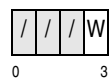


Figure 7-246. Register Contents for CONVERT UTF-16 TO UTF-8

When the ETF3-enhancement facility is installed, the M_3 field has the following format:



The bits of the M_3 field are defined as follows:

- **Unassigned:** Bits 0-2 are unassigned and should contain zeros; otherwise, the program may not operate compatibly in the future.
- **Enhanced Well-Formedness-Checking (W):** The W bit, bit 3 of the M_3 field, controls whether enhanced well-formedness checking of the Unicode characters is performed. When the W bit is

zero, enhanced checking is not performed. When the W bit is one, enhanced checking is performed, as described below.

When the ETF3-enhancement facility is not installed, the M_3 field is ignored.

The characters of the second operand are selected one by one for conversion, proceeding left to right. The bytes resulting from a conversion are placed at the first-operand location, proceeding left to right. The operation proceeds until the first-operand or second-operand location is exhausted, a CPU-determined number of second-operand characters have been converted, or an invalid Unicode character is encountered in the second operand.

To show the method of converting a Unicode character to a UTF-8 character, the bits of a Unicode character are identified by letters as follows:

Unicode Character Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	a b c d e f g h	i j k l m n o p

In the case of a Unicode surrogate pair, which is a character pair consisting of a character called a high surrogate followed by a character called a low surrogate, the bits are identified by letters as follows:

Unicode High Surrogate Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	1 1 0 1 1 0 a b	c d e f g h i j

Unicode Low Surrogate Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	1 1 0 1 1 1 k l	m n o p q r s t

Any Unicode character in the range 0000 to 007F hex is converted to a one-byte UTF-8 character as follows:

Unicode Character	0 0 0 0 0 0 0 0	0 j k l m n o p
UTF-8 Character	0 j k l m n o p	

Any Unicode character in the range 0080 to 07FF hex is converted to a two-byte UTF-8 character as follows:

Unicode Character	0 0 0 0 0 f g h	i j k l m n o p
UTF-8 Character	1 1 0 f g h i j	1 0 k l m n o p

Any Unicode character in the range 0800 to D7FF and DC00 to FFFF hex is converted to a three-byte UTF-8 character as follows:

Unicode Character	a b c d e f g h	i j k l m n o p	
UTF-8 Character	1 1 1 0 a b c d	1 0 e f g h i j	1 0 k l m n o p

Any Unicode surrogate pair starting with a high surrogate in the range D800 to DBFF hex is converted to a four-byte UTF-8 character as follows:

Unicode Characters	1 1 0 1 1 0 a b	c d e f g h i j	1 1 0 1 1 1 k l	m n o p q r s t
UTF-8 Character	1 1 1 1 0 u v w	1 0 x y e f g h	1 0 i j k l m n	1 0 o p q r s t
where $uvwx = abcd + 1$				

When the ETF3-enhancement facility is not installed, or when the W bit of the M₃ field is zero, the first six bits of the Unicode low surrogate are ignored. When the ETF3-enhancement facility is installed, and the W bit is one, the first six bits of the Unicode low surrogate must contain 110111 binary; otherwise, the Unicode low surrogate is invalid, and condition code 2 is set.

The second-operand location is considered exhausted when it does not contain at least two remaining bytes or at least four remaining bytes when the first two bytes are a Unicode high surrogate. The first-operand location is considered exhausted when it does not contain at least the one, two, three, or four remaining bytes required to contain the UTF-8 character resulting from the conversion of the next second-operand character or surrogate pair.

When the second-operand location is exhausted, condition code 0 is set. When the first-operand location is exhausted, condition code 1 is set, except that condition code 0 is set if the second-operand location also is exhausted. When a CPU-determined number of characters have been converted, condition code 3 is set.

When the conditions for setting condition codes 1 and 2 are both met, condition code 2 is set.

When the operation is completed, the contents of general register R₂ + 1 are decremented by the number of bytes converted, and the contents of general register R₂ are incremented by the same number. Also, the contents of general register R₁ + 1 are decremented by the number of bytes placed at the first-operand location, and the contents of general register R₁ are incremented by the same number. When general registers R₁ and R₂ are updated in the 24-bit or 31-bit addressing mode, bits 32-39 of them, in the 24-bit mode, or bit 32, in the 31-bit mode, may be set to zeros or may remain unchanged.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, R_2 , and $R_2 + 1$, always remain unchanged.

When condition code 2 is set, general register R_2 contains the address of the invalid Unicode high surrogate that precedes the invalid low surrogate.

When condition code 3 is set, the registers have been updated so that the instruction, when reexecuted, resumes at the next byte locations to be processed.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the R_1 register is the same register as the R_2 register, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portions of the operands to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

- 0 Entire second operand processed
- 1 End of first operand reached
- 2 Invalid Unicode low surrogate
- 3 CPU-determined number of characters converted

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Specification

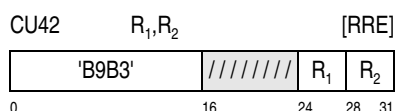
Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the conversion. The program need not determine

the number of first-operand or second-operand bytes that were processed.

2. The storage-operand references of CONVERT UNICODE TO UTF-8 may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
3. The CONVERT UTF-16 TO UTF-8 (CONVERT UNICODE TO UTF-8) instruction supports UTF-16 characters only in the big-endian encoding.

CONVERT UTF-32 TO UTF-16



The four-byte UTF-32 characters of the second operand are converted to two-byte UTF-16 characters and placed at the first-operand location. The operation proceeds until the end of the first or second operand is reached, a CPU-determined number of characters have been converted, or an invalid UTF-32 character is encountered, whichever occurs first. The result is indicated in the condition code.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and the second operand is designated by the contents of general registers R_1 and R_2 , respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the contents of bit positions 32-63 of general registers $R_1 + 1$ and $R_2 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the entire contents of general registers $R_1 + 1$ and $R_2 + 1$, respectively, and those contents are treated as 64-bit unsigned binary integers.

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the address, and the contents of bit positions

0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-247 on page 7-190.

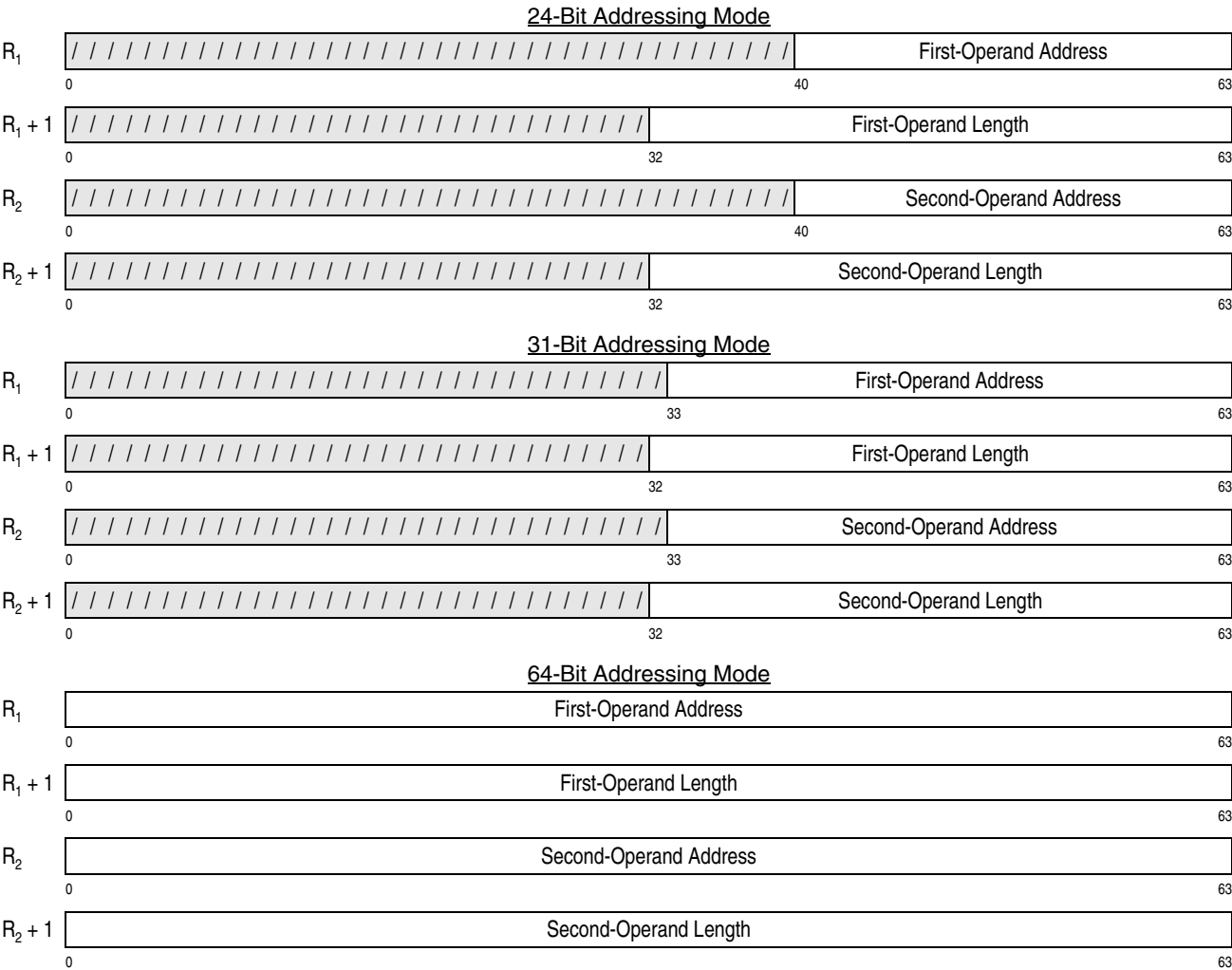


Figure 7-247. Register Contents for CONVERT UTF-32 TO UTF-16

The characters of the second operand are selected one by one for conversion, proceeding left to right. The bytes resulting from a conversion are placed at the first-operand location, proceeding left to right. The operation proceeds until the first-operand or second-operand location is exhausted, a CPU-determined number of second-operand characters have been converted, or an invalid UTF-32 character is encountered in the second operand.

To show the method of converting a UTF-32 character to a UTF-16 character, the bits of a UTF-32 character in the range 00000000 to 0000D7FF and

0000DC00 to 0000FFFF hex are identified by letters as follows:

UTF-32 Character Bit Numbers	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Identifying Bit Letters	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

UTF-32 Character Bit Numbers	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Identifying Bit Letters	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p

The bits of a UTF-32 character in the range 00010000 to 0010FFFF hex are identified by letters as follows:

UTF-32 Character		
Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	0 0 0 0 0 0 0 0	0 0 0 u v w x y

UTF-32 Character		
Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	e f g h i j k l	m n o p q r s t

The bits of a UTF-16 character in the range 0000 to D7FF and DC00 to FFFF hex are identified by letters as follows:

UTF-16 Character		
Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	a b c d e f g h	i j k l m n o p

In the case of a UTF-16 surrogate pair, which is a character pair consisting of a character called a high surrogate followed by a character called a low surrogate, the bits are identified by letters as follows:

UTF-16 High Surrogate Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	1 1 0 1 1 0 a b	c d e f g h i j

UTF-16 Low Surrogate Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	1 1 0 1 1 1 k l	m n o p q r s t

When the contents of the UTF-32 characters are in the range 00000000 to 0000D7FF and 0000DC00 to 0000FFFF hex, the character is converted to a two-byte UTF-32 character as follows:

UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	a b c d e f g h	i j k l m n o p
UTF-16 Character	a b c d e f g h	i j k l m n o p		

When the contents of the a UTF-32 character are in the range 00010000 to 0010FFFF hex, the character

is converted to two two-byte UTF-16 characters (a surrogate pair) as follows:

UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 u v w x y	e f g h i j k l	m n o p q r s t
UTF-16 Characters	1 1 0 1 1 0 a b	c d e f g h i j	1 1 0 1 1 1 k l	m n o p q r s t
where zabcd = uvwxy - 1				

The high order bit (z) produced by the subtract operation should be zero but is ignored.

The second-operand location is considered exhausted when it does not contain at least four remaining bytes. The first-operand location is considered exhausted when it does not contain at least two remaining bytes or at least four remaining bytes in the case when a UTF-32 character in the range 00010000 to 0010FFFF hex is converted.

When the second-operand location is exhausted, condition code 0 is set. When the first-operand location is exhausted, condition code 1 is set, except that condition code 0 is set if the second-operand location also is exhausted. When a CPU-determined number of characters have been processed, condition code 3 is set.

When the contents of the next UTF-32 character is in the range 0000D800 to 0000DBFF or 00110000 to FFFFFFFF hex, the character is invalid, and condition code 2 is set.

When the conditions for setting condition codes 1 and 2 are both met, condition code 2 is set.

When the operation is completed, the contents of general register $R_2 + 1$ are decremented by the number of bytes converted, and the contents of general register R_2 are incremented by the same number. Also, the contents of general register $R_1 + 1$ are decremented by the number of bytes placed at the first-operand location, and the contents of general register R_1 are incremented by the same number. When general registers R_1 and R_2 are updated in the 24-bit or 31-bit addressing mode, bits 32-39 of them, in the 24-bit mode, or bit 32, in the 31-bit mode, may be set to zeros or may remain unchanged.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, R_2 , and $R_2 + 1$, always remain unchanged.

When condition code 2 is set, general register R_2 contains the address of the invalid UTF-32 character.

When condition code 3 is set, the registers have been updated so that the instruction, when reexecuted, resumes at the next byte locations to be processed.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the R_1 register is the same register as the R_2 register, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portions of the operands to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

- 0 Entire second operand processed
- 1 End of first operand reached
- 2 Invalid UTF-32 character
- 3 CPU-determined number of characters processed

Program Exceptions:

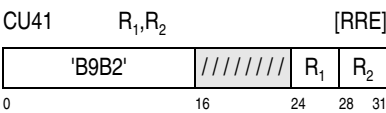
- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 3 is not installed)
- Specification

Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the conversion. The program need not determine the number of first-operand or second-operand bytes that were processed.

2. The storage-operand references of CONVERT UTF-32 TO UTF-16 may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
3. The CONVERT UTF-32 TO UTF-16 instruction supports UTF-16 and UTF-32 characters only in the big-endian encoding.

CONVERT UTF-32 TO UTF-8



The four-byte UTF-32 characters of the second operand are converted to UTF-8 characters and placed at the first-operand location. The UTF-8 characters are one, two, three, or four bytes, depending on the Unicode characters that are converted. The operation proceeds until the end of the first or second operand is reached or a CPU-determined number of characters have been converted, whichever occurs first. The result is indicated in the condition code.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and the second operand is designated by the contents of general registers R_1 and R_2 , respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the contents of bit positions 32-63 of general registers $R_1 + 1$ and $R_2 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the entire contents of general registers $R_1 + 1$ and $R_2 + 1$, respectively, and those contents are treated as 64-bit unsigned binary integers.

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the

contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-248 on page 7-193.

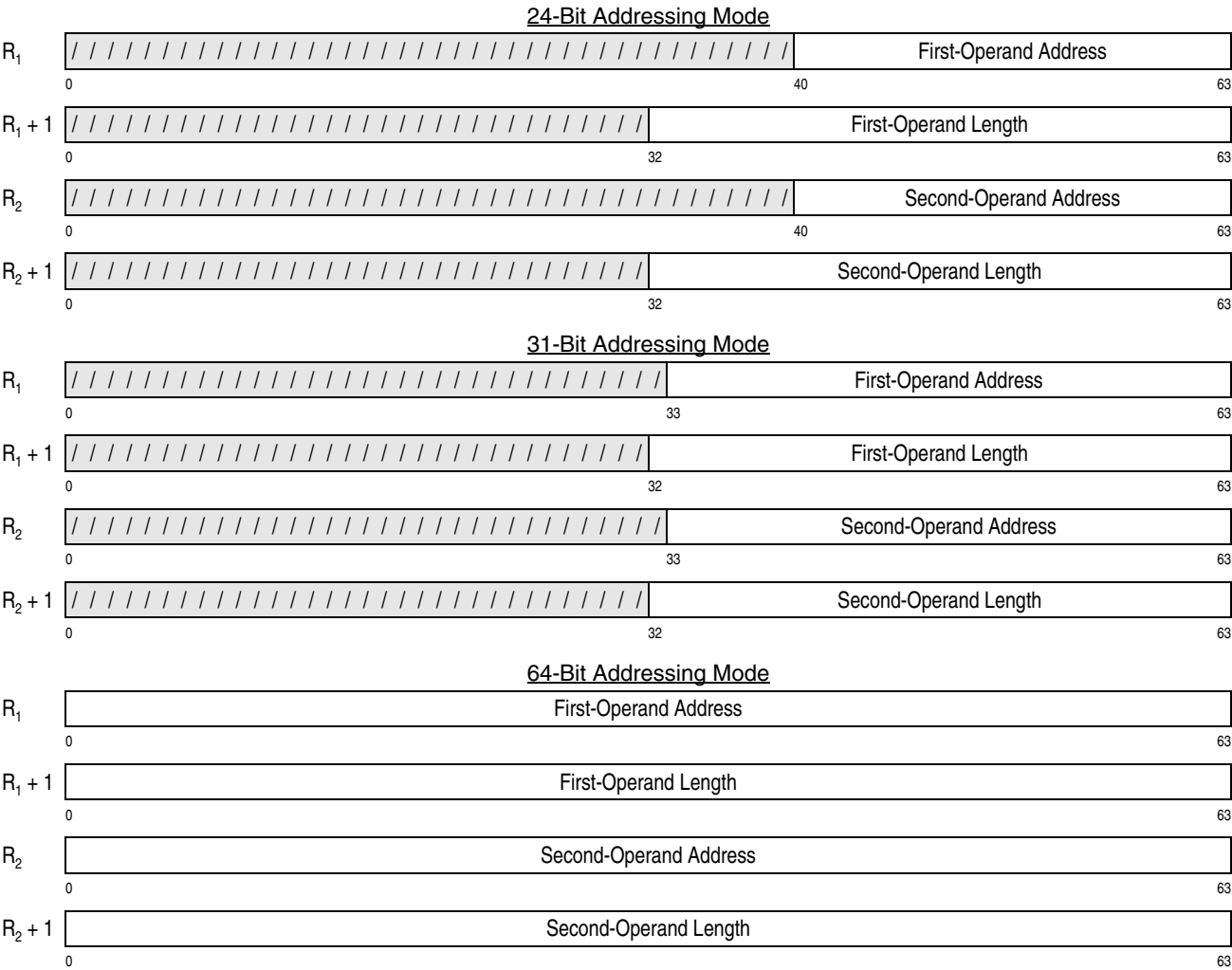


Figure 7-248. Register Contents for CONVERT UTF-32 TO UTF-8

The characters of the second operand are selected one by one for conversion, proceeding left to right. The bytes resulting from a conversion are placed at the first-operand location, proceeding left to right. The operation proceeds until the first-operand or second-operand location is exhausted or a CPU-determined number of second-operand characters have been converted.

To show the method of converting a UTF-32 character to a UTF-8 character, the bits of a UTF-32 character in the range 00000000 to 0000D7FF and

0000DC00 to 0000FFFF hex are identified by letters as follows:

UTF-32 Character Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

UTF-32 Character Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	a b c d e f g h	i j k l m n o p

The bits of a UTF-32 character in the range 00010000 to 0010FFFF hex (UTF-16 surrogate pair) are identified by letters as follows:

UTF-32 Character Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	0 0 0 0 0 0 0 0	0 0 0 u v w x y

UTF-32 Character Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	e f g h i j k l	m n o p q r s t

Any UTF-32 character in the range 00000000 to 0000007F hex is converted to a one-byte UTF-8 character as follows:

UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 j k l m n o p
UTF-8 Character	0 j k l m n o p			

Any UTF-32 character in the range 00000080 to 000007FF hex is converted to a two-byte UTF-8 character as follows:

UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 f g h	i j k l m n o p
UTF-8 Character	1 1 0 f g h i j	1 0 k l m n o p		

Any UTF-32 character in the range 00000800 to 0000D7FF and 0000DC00 to 0000FFFF hex is converted to a three-byte UTF-8 character as follows:

UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	a b c d e f g h	i j k l m n o p
UTF-8 Character	1 1 1 0 a b c d	1 0 e f g h i j	1 0 k l m n o p	

Any UTF-32 character in the range 00010000 to 0010FFFF hex is converted to a four-byte UTF-8 character as follows:

UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 u v w x y	e f g h i j k l	m n o p q r s t
UTF-8 Character	1 1 1 1 0 u v w	1 0 x y e f g h	1 0 i j k l m n	1 0 o p q r s t

The second-operand location is considered exhausted when it does not contain at least four remaining bytes. The first-operand location is considered exhausted when it does not contain at least the

one, two, three, or four remaining bytes required to contain the UTF-8 character resulting from the conversion of the next second-operand character or surrogate pair.

When the second-operand location is exhausted, condition code 0 is set. When the first-operand location is exhausted, condition code 1 is set, except that condition code 0 is set if the second-operand location also is exhausted. When a CPU-determined number of characters have been converted, condition code 3 is set.

When the UTF-32 character is in the range 0000D800 and 0000DBFF or 00110000 to FFFFFFFF hex, the character is invalid, and condition code 2 is set.

When the conditions for setting condition codes 1 and 2 are both met, condition code 2 is set.

When the operation is completed, the contents of general register $R_2 + 1$ are decremented by the number of bytes converted, and the contents of general register R_2 are incremented by the same number. Also, the contents of general register $R_1 + 1$ are decremented by the number of bytes placed at the first-operand location, and the contents of general register R_1 are incremented by the same number. When general registers R_1 and R_2 are updated in the 24-bit or 31-bit addressing mode, bits 32-39 of them, in the 24-bit mode, or bit 32, in the 31-bit mode, may be set to zeros or may remain unchanged.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, R_2 , and $R_2 + 1$, always remain unchanged.

When condition code 2 is set, general register R_2 contains the address of the invalid UTF-32 character.

When condition code 3 is set, the registers have been updated so that the instruction, when reexecuted, resumes at the next byte locations to be processed.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the R_1 register is the same register as the R_2 register, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portions of the operands to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

- 0 Entire second operand processed
- 1 End of first operand reached
- 2 Invalid UTF-32 character.
- 3 CPU-determined number of characters converted

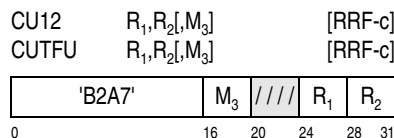
Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 3 is not installed)
- Specification

Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the conversion. The program need not determine the number of first-operand or second-operand bytes that were processed.
2. The storage-operand references of CONVERT UTF-32 TO UTF-8 may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
3. The CONVERT UTF-32 TO UTF-8 instruction supports UTF-32 characters only in the big-endian encoding.

CONVERT UTF-8 TO UTF-16 CONVERT UTF-8 TO UNICODE



The one-, two-, three-, or four-byte UTF-8 characters of the second operand are converted to two-byte Unicode characters and placed at the first-operand location. The operation proceeds until the end of the first or second operand is reached, a CPU-determined number of characters have been converted, or an invalid UTF-8 character is encountered, whichever occurs first. The result is indicated in the condition code.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and the second operand is designated by the contents of general registers R_1 and R_2 , respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the contents of bit positions 32-63 of general registers $R_1 + 1$ and $R_2 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the entire contents of general registers $R_1 + 1$ and $R_2 + 1$, respectively, and those contents are treated as 64-bit unsigned binary integers.

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-249 on page 7-196.

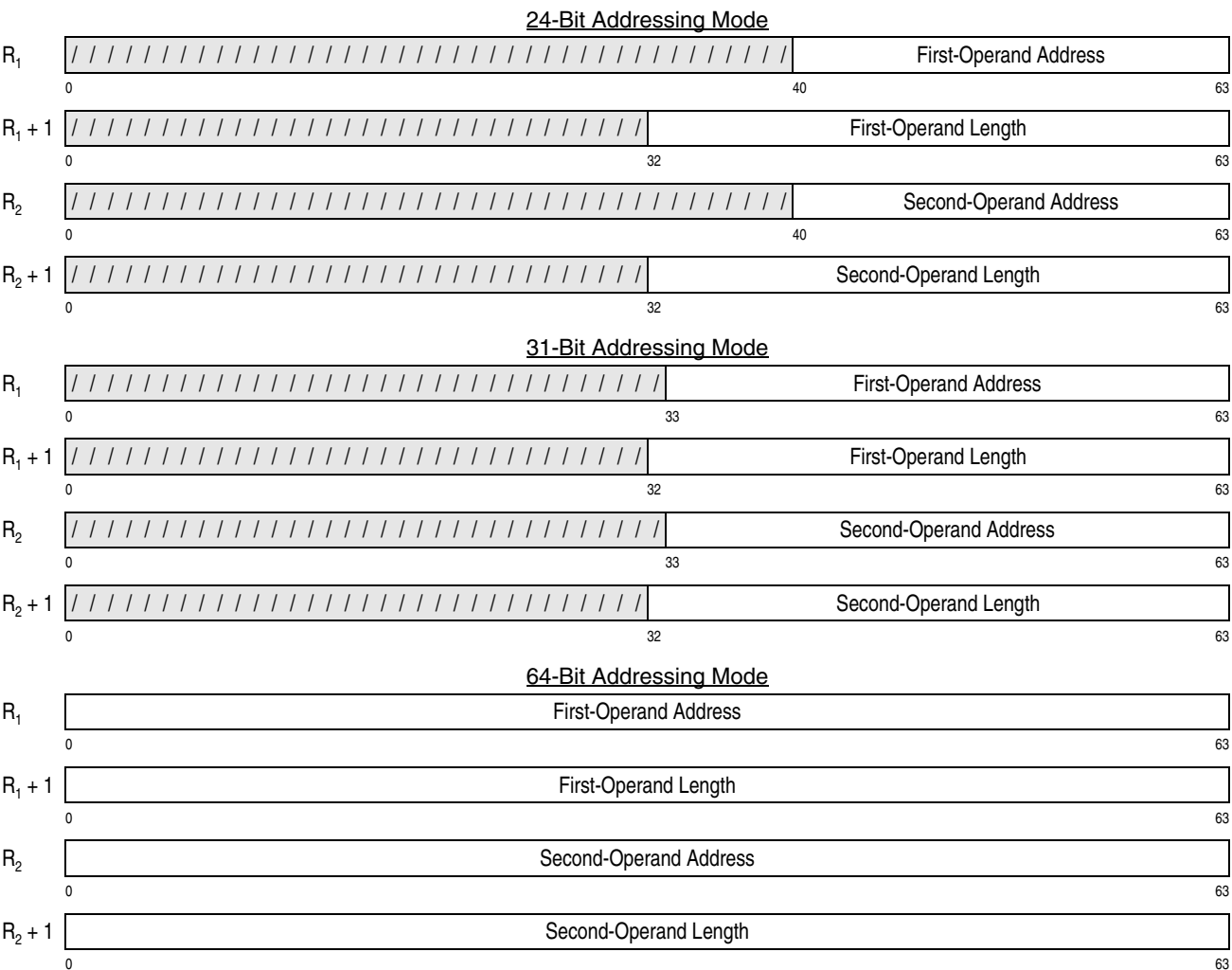
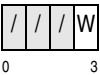


Figure 7-249. Register Contents for CONVERT UTF-8 TO UTF-16

When the ETF3-enhancement facility is installed, the M_3 field has the following format:



The bits of the M_3 field are defined as follows:

- Unassigned:** Bits 0-2 are unassigned and should contain zeros; otherwise, the program may not operate compatibly in the future.
- Enhanced Well-Formedness-Checking (W):** The W bit, bit 3 of the M_3 field, controls whether enhanced well-formedness checking of the UTF-8 characters is performed. When the W bit is

zero, enhanced checking is not performed. When the W bit is one, enhanced checking is performed, as described below.

When the ETF3-enhancement facility is not installed, the M_3 field is ignored.

The characters of the second operand are selected one by one for conversion, proceeding left to right. The bytes resulting from a conversion are placed at the first-operand location, proceeding left to right. The operation proceeds until the first-operand or second-operand location is exhausted, a CPU-determined number of second-operand characters have been converted, or an invalid UTF-8 character is encountered in the second operand.

To show the method of converting a UTF-8 character to a Unicode character, the bits of a Unicode character are identified by letters as follows:

Unicode Character Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	a b c d e f g h	i j k l m n o p

In the case of a Unicode surrogate pair, which is a character pair consisting of a character called a high surrogate followed by a character called a low surrogate, the bits are identified by letters as follows:

Unicode High Surrogate Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	1 1 0 1 1 0 a b	c d e f g h i j

Unicode Low Surrogate Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	1 1 0 1 1 1 k l	m n o p q r s t

Conversion of a UTF-8 character to a Unicode character is as follows:

1. When the contents of the first byte of a UTF-8 character are in the range 00 to 7F hex, the character is a one-byte character, and it is converted to a two-byte Unicode character as follows:

UTF-8 Character	0 j k l m n o p
Unicode Character	0 0 0 0 0 0 0 0 j k l m n o p

2. When the contents of the first byte of the UTF-8 character are in the range 80 to BF hex, the character is invalid. When the ETF3-enhancement facility is installed, the W bit of the M_3 field is one, and the contents of the first byte of the UTF-8 character are in the range C0 to C1 hex, the character is also invalid.
3. When the ETF3-enhancement facility is not installed or the W bit of the M_3 field is zero, and the contents of the first byte of the UTF-8 character are in the range of C0 to DF hex; or when the ETF3-enhancement facility is installed, the W bit is one, and the contents of the first byte of the UTF-8 character are in the range of C2 to DF hex; the character is a two-byte character, and it

is converted to a two-byte Unicode character as follows:

UTF-8 Character	1 1 0 f g h i j	1 0 k l m n o p
Unicode Character	0 0 0 0 0 f g h	i j k l m n o p

When the ETF3-enhancement facility is not installed or when the W bit of the M_3 field is zero, the first two bits in the second byte of the UTF-8 character are ignored. When the ETF3-enhancement facility is installed and the W bit of the M_3 field is one, the contents of the second byte of the UTF-8 character must be in the range 80 to BF; otherwise the character is invalid.

4. When the contents of the first byte of a UTF-8 character are in the range E0 to EF hex, the character is a three-byte character, and it is converted to a two-byte Unicode character as follows:

UTF-8 Character	1 1 1 0 a b c d	1 0 e f g h i j	1 0 k l m n o p
Unicode Character	a b c d e f g h	i j k l m n o p	

When the ETF3-enhancement facility is not installed or when the W bit of the M_3 field is zero, the first two bits in the second and third bytes of the UTF-8 character are ignored.

When the ETF3-enhancement facility is installed and the W bit of the M_3 field is one, the contents of the second and third bytes of the UTF-8 character must be as follows:

- When the first byte is E0 hex, the second and third bytes must be in the ranges A0 to BF and 80 to BF, respectively.
- When the first byte is in the range E1 to EC hex or EE to EF, the second and third bytes must both be in the range 80 to BF hex.
- When the first byte is ED hex, the second and third bytes must be in the ranges 80 to 9F and 80 to BF, respectively.

Otherwise, the character is invalid.

5. When the ETF3-enhancement facility is not installed or the W bit of the M_3 field is zero, and the contents of the first byte of the UTF-8 character are in the range of F0 to F7 hex; or when the ETF3-enhancement facility is installed, the W bit is one, and the contents of the first byte of the

UTF-8 character are in the range of F0 to F4 hex; the character is a four-byte character, and it is converted to two two-byte Unicode characters (a surrogate pair) as follows:

UTF-8 Character	1 1 1 1 0 u v w	1 0 x y e f g h	1 0 i j k l m n	1 0 o p q r s t
Unicode Characters	1 1 0 1 1 0 a b	c d e f g h i j	1 1 0 1 1 1 k l	m n o p q r s t
where $zabcd = uvwxy - 1$				

When the ETF3-enhancement facility is not installed or when the W bit of the M₃ field is zero, the first two bits in the second, third, and fourth bytes of the UTF-8 character are ignored, and the high order bit (z) produced by the subtract operation should be zero but is ignored.

When the ETF3-enhancement facility is installed and the W bit of the M₃ field is one, the contents of the second, third, and fourth bytes of the UTF-8 character must be as follows:

- When the first byte is F0 hex, the second, third, and fourth bytes must be in the ranges 90 to BF, 80 to BF, and 80 to BF, respectively.
- When the first byte is in the range F1 to F3, the second, third, and fourth bytes must all be in the range 80 to BF hex.
- When the first byte is F4 hex, the second, third, and fourth bytes must be in the ranges 80 to 8F, 80 to BF, and 80 to BF, respectively.

Otherwise, the character is invalid.

6. When the ETF3-enhancement facility is installed, the W bit of the M₃ field is one, and the contents of the first byte of the UTF-8 character are in the range of F5 to F7 hex, the character is invalid.
7. When the contents of the first byte of the UTF-8 character are in the range of F8-FF, the character is invalid.

If an invalid character is encountered, condition code 2 is set, and general register R₂ contains the address of the first byte of the invalid UTF-8 character.

The second-operand location is considered exhausted when it does not contain at least one remaining byte or when it does not contain at least the two, three, or four remaining bytes required to contain the two-, three-, or four-byte UTF-8 character indicated by the contents of the first remaining byte. The first-operand location is considered exhausted

when it does not contain at least two remaining bytes or at least four remaining bytes in the case when a four byte UTF-8 character is to be converted.

When the second-operand location is exhausted, condition code 0 is set. When the first-operand location is exhausted, condition code 1 is set, except that condition code 0 is set if the second-operand location also is exhausted. When a CPU-determined number of characters have been processed, condition code 3 is set.

When the contents of the first byte of the next UTF-8 character are in the range 80 to BF hex or F8 to FF hex, the character is invalid, and condition code 2 is set.

When the conditions for setting condition codes 1 and 2 are both met, condition code 2 is set.

When the operation is completed, the contents of general register R₂ + 1 are decremented by the number of bytes converted, and the contents of general register R₂ are incremented by the same number. Also, the contents of general register R₁ + 1 are decremented by the number of bytes placed at the first-operand location, and the contents of general register R₁ are incremented by the same number. When general registers R₁ and R₂ are updated in the 24-bit or 31-bit addressing mode, bits 32-39 of them, in the 24-bit mode, or bit 32, in the 31-bit mode, may be set to zeros or may remain unchanged.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R₁, R₁ + 1, R₂, and R₂ + 1, always remain unchanged.

When condition code 2 is set, general register R₂ contains the address of the invalid UTF-8 character.

When condition code 3 is set, the registers have been updated so that the instruction, when reexecuted, resumes at the next byte locations to be processed.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the R₁ register is the same register as the R₂ register, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portions of the operands to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

- 0 Entire second operand processed
- 1 End of first operand reached
- 2 Invalid UTF-8 character
- 3 CPU-determined number of characters processed

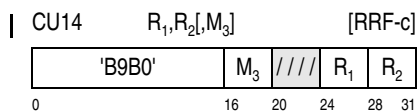
Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Specification

Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the conversion. The program need not determine the number of first-operand or second-operand bytes that were processed.
2. When the ETF3-enhancement facility is not installed, or when the W bit of the M_3 operand is zero, bits 0 and 1 of the continuation bytes of multiple-byte UTF-8 characters are not checked in order to improve the performance of the conversion. Therefore, invalid continuation bytes are not detected.
3. The storage-operand references of CONVERT UTF-8 TO UTF-16 may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
4. The CONVERT UTF-8 TO UTF-16 (CONVERT UTF-8 TO UNICODE) instruction supports UTF-16 characters only in the big-endian encoding.

CONVERT UTF-8 TO UTF-32



The one-, two-, three-, or four-byte UTF-8 characters of the second operand are converted to four-byte UTF-32 characters and placed at the first-operand location. The operation proceeds until the end of the first or second operand is reached, a CPU-determined number of characters have been converted, or an invalid UTF-8 character is encountered, whichever occurs first. The result is indicated in the condition code.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and the second operand is designated by the contents of general registers R_1 and R_2 , respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the contents of bit positions 32-63 of general registers $R_1 + 1$ and $R_2 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and second-operand locations is specified by the entire contents of general registers $R_1 + 1$ and $R_2 + 1$, respectively, and those contents are treated as 64-bit unsigned binary integers.

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-250 on page 7-200.

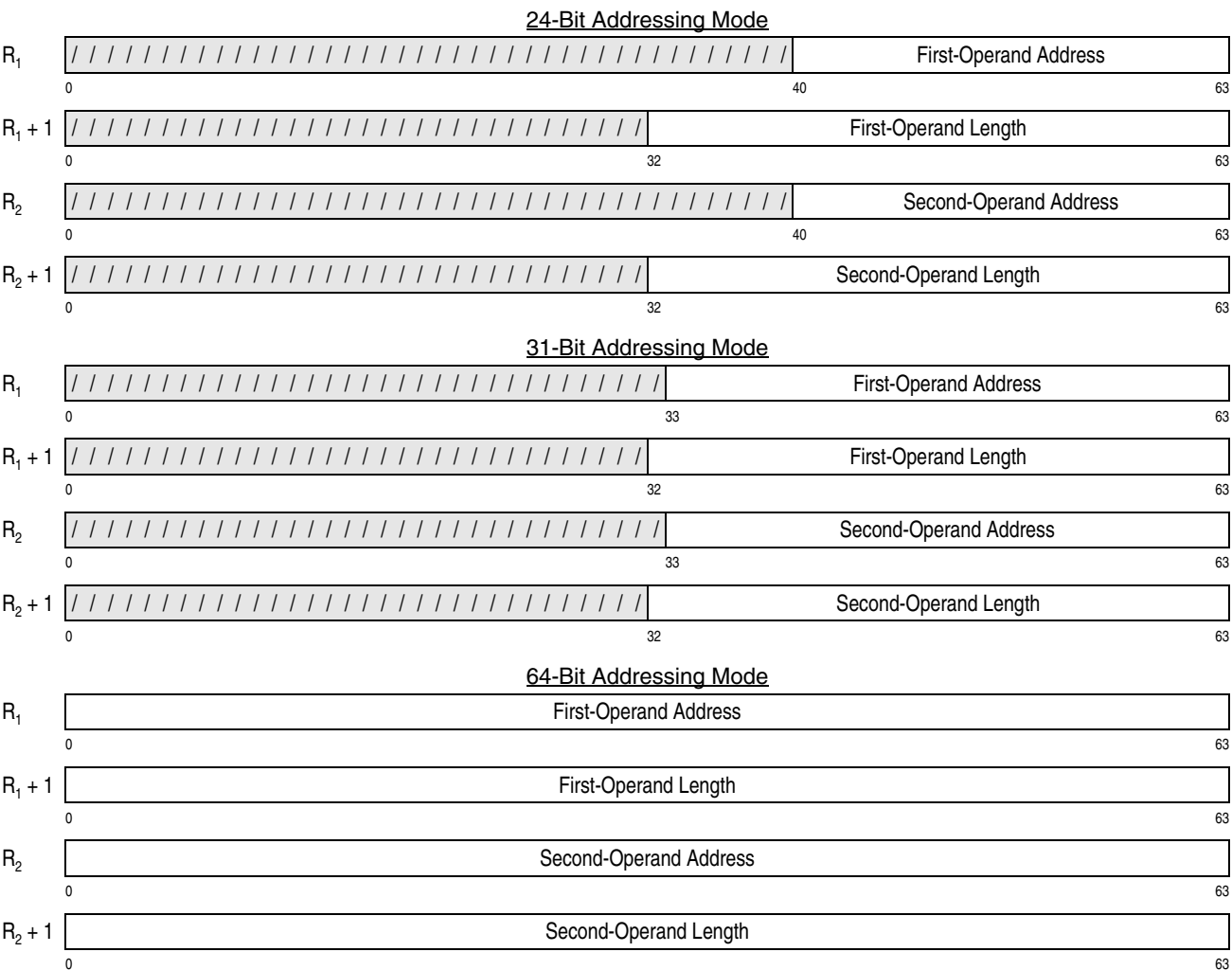
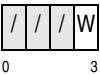


Figure 7-250. Register Contents for CONVERT UTF-8 TO UTF32

When the ETF3-enhancement facility is installed, the M_3 field has the following format:



The bits of the M_3 field are defined as follows:

- Unassigned:** Bits 0-2 are unassigned and should contain zeros; otherwise, the program may not operate compatibly in the future.
- Enhanced Well-Formedness-Checking (W):** The W bit, bit 3 of the M_3 field, controls whether enhanced well-formedness checking of the UTF-8 characters is performed. When the W bit is zero, enhanced checking is not performed. When

the W bit is one, enhanced checking is performed, as described below.

When the ETF3-enhancement facility is not installed, the M_3 field is ignored.

The characters of the second operand are selected one by one for conversion, proceeding left to right. The bytes resulting from a conversion are placed at the first-operand location, proceeding left to right. The operation proceeds until the first-operand or second-operand location is exhausted, a CPU-determined number of second-operand characters have been converted, or an invalid UTF-8 character is encountered in the second operand.

To show the method of converting a UTF-8 character to a UTF-32 character, the bits of a UTF-32 character

in the range 00000000 0000D7FF and 0000DC00 to 0000FFFF hex are identified by letters as follows:

UTF-32 Character		
Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

UTF-32 Character		
Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	a b c d e f g h	i j k l m n o p

The bits of a UTF-32 character in the range 00010000 0010FFFF hex (UTF-16 surrogate pair) are identified by letters as follows:

UTF-32 Character		
Bit Numbers	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
Identifying Bit Letters	0 0 0 0 0 0 0 0	0 0 0 u v w x y

UTF-32 Character		
Bit Numbers	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31
Identifying Bit Letters	e f g h i j k l	m n o p q r s t

Conversion of a UTF-8 character to a UTF-32 character is as follows:

1. When the contents of the first byte of a UTF-8 character are in the range 00 to 7F hex, the character is a one-byte character, and it is converted to a four-byte UTF-32 character as follows:

UTF-8 Character	0 j k l m n o p			
UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 j k l m n o p

2. When the contents of the first byte of the UTF-8 character are in the range 80 to BF hex, the character is invalid. When the ETF3-enhancement facility is installed, the W bit of the M_3 field is one, and the contents of the first byte of the UTF-8 character are in the range C0 to C1 hex, the character is also invalid.
3. When the ETF3-enhancement facility is not installed or the W bit of the M_3 field is zero, and the contents of the first byte of the UTF-8 character are in the range of C0 to DF hex; or when the ETF3-enhancement facility is installed, the W bit is one, and the contents of the first byte of the

UTF-8 character are in the range of C2 to DF hex; the character is a two-byte character, and it is converted to a four-byte UTF-32 character as follows:

UTF-8 Character	1 1 0 f g h i j	1 0 k l m n o p			
UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 f g h	i j k l m n o p	

When the ETF3-enhancement facility is not installed or when the W bit of the M_3 field is zero, the first two bits in the second byte of the UTF-8 character are ignored. When the ETF3-enhancement facility is installed and the W bit of the M_3 field is one, the contents of the second byte of the UTF-8 character must be in the range 80 to BF; otherwise the character is invalid.

4. When the contents of the first byte of a UTF-8 character are in the range E0 to EF hex, the character is a three-byte character, and it is converted to a four-byte UTF-32 character as follows:

UTF-8 Character	1 1 1 0 a b c d	1 0 e f g h i j	1 0 k l m n o p	
UTF-32 Character	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	a b c d e f g h	i j k l m n o p

When the ETF3-enhancement facility is not installed or when the W bit of the M_3 field is zero, the first two bits in the second and third bytes of the UTF-8 character are ignored.

When the ETF3-enhancement facility is installed and the W bit of the M_3 field is one, the contents of the second and third bytes of the UTF-8 character must be as follows:

- When the first byte is E0 hex, the second and third bytes must be in the ranges A0 to BF and 80 to BF, respectively.
- When the first byte is in the range E1 to EC hex or EE to EF, the second and third bytes must both be in the range 80 to BF hex.
- When the first byte is ED hex, the second and third bytes must be in the ranges 80 to 9F and 80 to BF, respectively.

Otherwise, the character is invalid.

5. When the ETF3-enhancement facility is not installed or the W bit of the M_3 field is zero, and the contents of the first byte of the UTF-8 character are in the range of F0 to F7 hex; or when the

ETF3-enhancement facility is installed, the W bit is one, and the contents of the first byte of the UTF-8 character are in the range of F0 to F4 hex; the character is a four-byte character, and it is converted to a four-byte UTF-32 character (a surrogate pair) as follows:

UTF-8 Character	11110uvw	10xyefgh	10ijklmn	10opqrst
UTF-32 Character	00000000	000uvwxxy	efghijkl	mnoqrst

When the ETF3-enhancement facility is not installed or when the W bit of the M_3 field is zero, the first two bits in the second, third, and fourth bytes of the UTF-8 character are ignored.

When the ETF3-enhancement facility is installed and the W bit of the M_3 field is one, the contents of the second, third, and fourth bytes of the UTF-8 character must be as follows:

- When the first byte is F0 hex, the second, third, and fourth bytes must be in the ranges 90 to BF, 80 to BF, and 80 to BF, respectively.
- When the first byte is in the range F1 to F3, the second, third, and fourth bytes must all be in the range 80 to BF hex.
- When the first byte is F4 hex, the second, third, and fourth bytes must be in the ranges 80 to 8F, 80 to BF, and 80 to BF, respectively.

Otherwise, the character is invalid.

6. When the ETF3-enhancement facility is installed, the W bit of the M_3 field is one, and the contents of the first byte of the UTF-8 character are in the range of F5 to F7 hex, the character is invalid.
7. When the contents of the first byte of the UTF-8 character are in the range of F8-FF, the character is invalid.

If an invalid character is encountered, condition code 2 is set, and general register R_2 contains the address of the first byte of the invalid UTF-8 character.

The second-operand location is considered exhausted when it does not contain at least one remaining byte or when it does not contain at least the two, three, or four remaining bytes required to contain the two-, three-, or four-byte UTF-8 character indicated by the contents of the first remaining byte. The first-operand location is considered exhausted

when it does not contain at least four remaining bytes.

When the second-operand location is exhausted, condition code 0 is set. When the first-operand location is exhausted, condition code 1 is set, except that condition code 0 is set if the second-operand location also is exhausted. When a CPU-determined number of characters have been processed, condition code 3 is set.

When the contents of the first byte of the next UTF-8 character are in the range 80 to BF hex or F8 to FF hex, the character is invalid, and condition code 2 is set.

When the conditions for setting condition codes 1 and 2 are both met, condition code 2 is set.

When the operation is completed, the contents of general register $R_2 + 1$ are decremented by the number of bytes converted, and the contents of general register R_2 are incremented by the same number. Also, the contents of general register $R_1 + 1$ are decremented by the number of bytes placed at the first-operand location, and the contents of general register R_1 are incremented by the same number. When general registers R_1 and R_2 are updated in the 24-bit or 31-bit addressing mode, bits 32-39 of them, in the 24-bit mode, or bit 32, in the 31-bit mode, may be set to zeros or may remain unchanged.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, R_2 , and $R_2 + 1$, always remain unchanged.

When condition code 2 is set, general register R_2 contains the address of the invalid UTF-8 character.

When condition code 3 is set, the registers have been updated so that the instruction, when reexecuted, resumes at the next byte locations to be processed.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the R_1 register is the same register as the R_2 register, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portions of the operands to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

- 0 Entire second operand processed
- 1 End of first operand reached
- 2 Invalid UTF-8 character
- 3 CPU-determined number of characters processed

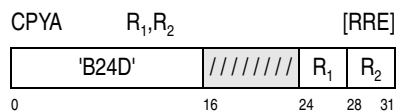
Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 3 is not installed)
- Specification

Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the conversion. The program need not determine the number of first-operand or second-operand bytes that were processed.
2. When the ETF3-enhancement facility is not installed, or when the W bit of the M₃ operand is zero, bits 0 and 1 of the continuation bytes of multiple-byte UTF-8 characters are not checked in order to improve the performance of the conversion. Therefore, invalid continuation bytes are not detected.
3. The storage-operand references of CONVERT UTF-8 TO UTF-32 may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
4. The CONVERT UTF-8 TO UTF-32 instruction supports UTF-32 characters only in the big-endian encoding.

COPY ACCESS



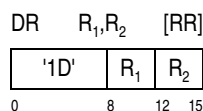
The contents of access register R₂ are placed in access register R₁.

Condition Code: The code remains unchanged.

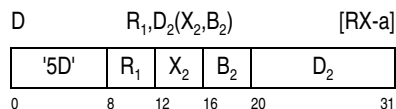
Program Exceptions: None.

DIVIDE

Register-and-register format:



Register-and-storage format:



The 64-bit first operand (the dividend) is divided by the 32-bit second operand (the divisor), and the 32-bit remainder and quotient are placed at the first-operand location.

The R₁ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The dividend is treated as a 64-bit signed binary integer. The leftmost 32 bits of the dividend are in bit positions 32-63 of general register R₁, and the rightmost 32 bits are in bit positions 32-63 of general register R₁ + 1.

The divisor, remainder, and quotient are treated as 32-bit signed binary integers. For DIVIDE (DR), the divisor is in bit positions 32-63 of general register R₂. The remainder is placed in bit positions 32-63 of general register R₁, and the quotient is placed in bit positions 32-63 of general register R₁ + 1. Bits 0-31 of the registers remain unchanged.

The sign of the quotient is determined by the rules of algebra, and the remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive.

When the divisor is zero, or when the magnitudes of the dividend and divisor are such that the quotient cannot be expressed by a 32-bit signed binary integer, a fixed-point-divide exception is recognized. This includes the case of division of zero by zero.

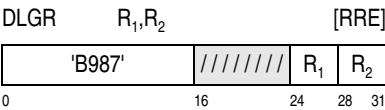
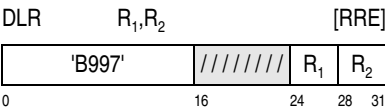
Condition Code: The code remains unchanged.

Program Exceptions:

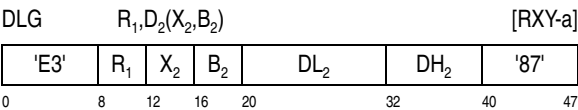
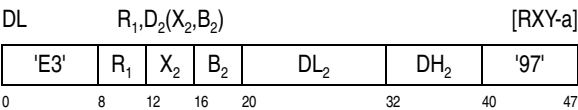
- Access (fetch, operand 2 of D only)
- Fixed-point divide
- Specification

DIVIDE LOGICAL

Register-and-register formats:



Register-and-storage formats:



The 64-bit or 128-bit first operand (the dividend) is divided by the 32-bit or 64-bit second operand (the divisor), and the 32-bit or 64-bit remainder and quotient are placed at the first-operand location.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

For DIVIDE LOGICAL (DLR, DL), the dividend is treated as a 64-bit unsigned binary integer. The leftmost 32 bits of the dividend are in bit positions 32-63 of general register R_1 , and the rightmost 32 bits are in bit positions 32-63 of general register $R_1 + 1$.

The divisor, remainder, and quotient are treated as 32-bit unsigned binary integers. For DIVIDE LOGICAL (DLR), the divisor is in bit positions 32-63 of general register R_2 . The remainder is placed in bit positions 32-63 of general register R_1 , and the quotient is placed in bit positions 32-63 of general register $R_1 + 1$. Bits 0-31 of the registers remain unchanged.

For DIVIDE LOGICAL (DLGR, DLG), the dividend is treated as a 128-bit unsigned binary integer. The leftmost 64 bits of the dividend are in general register R_1 , and the rightmost 64 bits are in general register $R_1 + 1$. The divisor, remainder, and quotient are treated as 64-bit unsigned binary integers. The remainder is placed in general register R_1 , and the quotient is placed in general register $R_1 + 1$.

When the divisor is zero, or when the magnitudes of the dividend and divisor are such that the quotient cannot be expressed as a 32-bit unsigned binary integer for DIVIDE LOGICAL (DLR, DL), or a 64-bit unsigned binary integer for DIVIDE LOGICAL (DLGR, DLG), a fixed-point-divide exception is recognized. This includes the case of division of zero by zero.

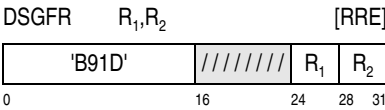
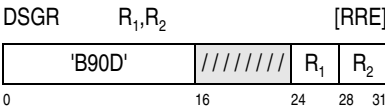
Condition Code: The code remains unchanged.

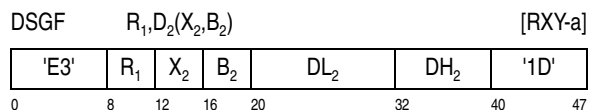
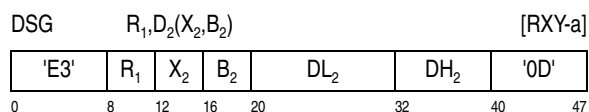
Program Exceptions:

- Access (fetch, operand 2 of DL or DLG only)
- Fixed-point divide
- Specification

DIVIDE SINGLE

Register-and-register formats:



Register-and-storage formats:

The 64-bit contents of general register $R_1 + 1$ (the dividend) are divided by the 64-bit or 32-bit second operand (the divisor), the 64-bit remainder is placed in general register R_1 , and the 64-bit quotient is placed in general register $R_1 + 1$.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The dividend, quotient, and remainder are treated as 64-bit signed binary integers. For DIVIDE SINGLE (DSGR, DSG), the divisor is treated as a 64-bit signed binary integer. For DIVIDE SINGLE (DSGFR, DSGF), the divisor is treated as a 32-bit signed binary integer. For DSGFR, the divisor is in bit positions 32-63 of general register R_2 .

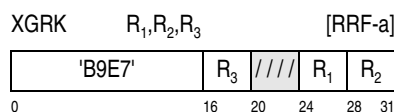
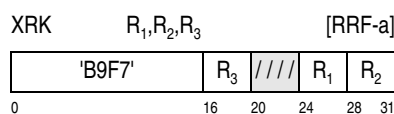
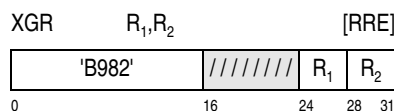
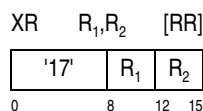
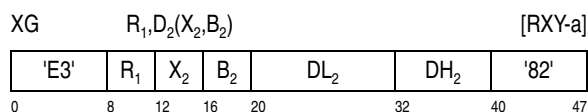
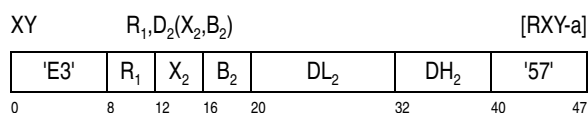
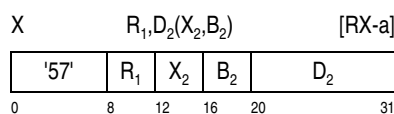
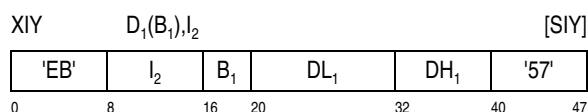
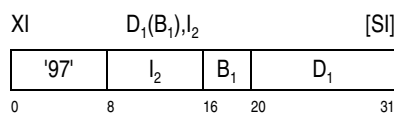
The sign of the quotient is determined by the rules of algebra, and the remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive.

When the divisor is zero, or when the magnitudes of the dividend and divisor are such that the quotient cannot be expressed by a 64-bit signed binary integer, a fixed-point-divide exception is recognized. This includes the case of division of zero by zero.

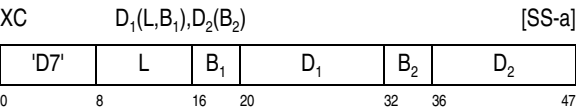
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of DSG and DSGF only)
- Fixed-point divide
- Specification

EXCLUSIVE OR**Register-and-register formats:****Register-and-storage formats:****Storage-and-immediate formats:**

Storage-and-storage format:



For X, XC, XG, XGR, XI, XIY, XR, and XY, the EXCLUSIVE OR of the first and second operands is placed at the first-operand location. For XGRK and XRK, the EXCLUSIVE OR of the second and third operands is placed at the first-operand location.

The connective EXCLUSIVE OR is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the bits in the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

For EXCLUSIVE OR (XC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

For EXCLUSIVE OR (XI, XIY), the first operand is one byte in length, and only one byte is stored.

For EXCLUSIVE OR (X, XR, XRK, and XY), the operands are 32 bits, and for EXCLUSIVE OR (XG, XGR, and XGRK), they are 64 bits.

The displacements for X, XI, and both operands of XC are treated as 12-bit unsigned binary integers. The displacement for XY, XIY, and XG is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 Result zero
- 1 Result not zero
- 2 --
- 3 --

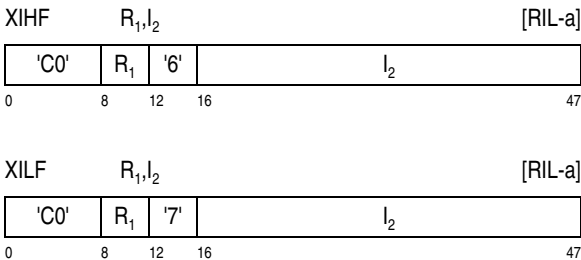
Program Exceptions:

- Access (fetch, operand 2, X, XY, XG, and XC; fetch and store, operand 1, XI, XIY, and XC)
- Operation (XY and XIY, if the long-displacement facility is not installed; XGRK and XRK, if the distinct-operands facility is not installed)

Programming Notes:

1. An example of the use of the EXCLUSIVE OR instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. EXCLUSIVE OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary switches.
3. A field EXCLUSIVE-ORed with itself becomes all zeros.
4. For EXCLUSIVE OR (XR or XGR), the sequence A EXCLUSIVE-OR B, B EXCLUSIVE-OR A, A EXCLUSIVE-OR B results in the exchange of the contents of A and B without the use of an additional general register.
5. Accesses to the first operand of EXCLUSIVE OR (XI) and EXCLUSIVE OR (XC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, EXCLUSIVE OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in "Multiprogramming and Multiprocessing Examples" in Appendix A, "Number Representation and Instruction-Use Examples."

EXCLUSIVE OR IMMEDIATE



The second operand is EXCLUSIVE ORed with bits of the first operand, and the result replaces those bits of the first operand. The remainder of the first operand remains unchanged.

For each instruction, the bits of the first operand that are EXCLUSIVE ORed with the second operand and then replaced are as follows:

Instruction	Bits Exclusive ORed and Replaced
XIHF	0-31
XILF	32-63

The connective EXCLUSIVE OR is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the bits in the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

The condition code is set based on the result of the 32-bit EXCLUSIVE OR operation.

Resulting Condition Code:

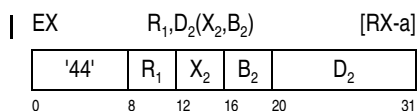
- 0 Result is zero
- 1 Result is not zero
- 2 --
- 3 --

Program Exceptions:

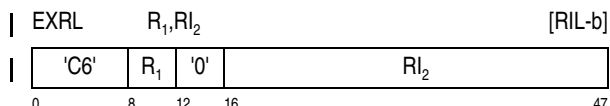
- Operation, if the extended-immediate facility is not installed)

Programming Note: The setting of the condition code is based only on the bits that are exclusive ORed and replaced.

EXECUTE



EXECUTE RELATIVE LONG



The single instruction at the second-operand address is modified by the contents of general register R_1 , and the resulting instruction, called the target instruction, is executed.

When the R_1 field is not zero, bits 8-15 of the instruction designated by the second-operand address are ORed with bits 56-63 of general register R_1 . The ORing does not change either the contents of general register R_1 or the instruction in storage, and it is effective only for the interpretation of the instruction to be executed. When the R_1 field is zero, no ORing takes place.

The target instruction may be two, four, or six bytes in length. The execution and exception handling of the target instruction are exactly as if the target instruction were obtained in normal sequential operation, except for the instruction address and the instruction-length code.

The instruction address in the current PSW is increased by the length of the execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG). This updated address and the instruction-length code of the execute-type instruction are used, for example, as part of the link information when the target instruction is BRANCH AND LINK. When the target instruction is a successful branching instruction, the instruction address in the current PSW is replaced by the branch address specified by the target instruction.

When the target instruction is in turn an execute-type instruction, an execute exception is recognized.

The effective address of EXECUTE must be even; otherwise, a specification exception is recognized.

When the target instruction is two or three halfwords in length but can be executed without fetching its second or third halfword, it is unpredictable whether access exceptions are recognized for the unused halfwords. Access exceptions are not recognized for the second-operand address when the address is odd.

The second-operand address of an execute-type instruction is an instruction address rather than a logical address; thus, the target instruction is fetched from the primary address space when in the primary-space, secondary-space, or access-register mode.

For EXECUTE RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the target instruction in storage.

Condition Code: The code may be set by the target instruction.

Program Exceptions:

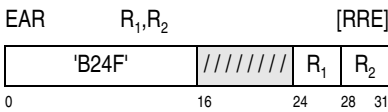
- Access (fetch, target instruction)
- Execute
- Operation (EXRL, when the execute-extensions facility is not installed)
- Specification (EX)

Programming Notes:

1. An example of the use of the EXECUTE instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. The ORing of eight bits from the general register with the designated instruction permits the indirect specification of the length, index, mask, immediate-data, register, or extended-op-code field.
3. The fetching of the target instruction is considered to be an instruction fetch for purposes of program-event recording and for purposes of reporting access exceptions.
4. An access or specification exception may be caused by execute-type instructions or by the target instruction, except that EXECUTE RELATIVE LONG cannot cause a specification exception due to a misaligned target operand.
5. When an interruptible instruction is made the target of an execute-type instruction, the program normally should not designate any register updated by the interruptible instruction as the R_1 , X_2 , or B_2 register for EXECUTE or R_1 register for EXECUTE RELATIVE LONG. Otherwise, on resumption of execution after an interruption, or if the instruction is refetched without an interruption, the updated values of these registers will be used in the execution of the execute-type instruction. Similarly, the program should normally not let the destination field in storage of an interruptible instruction include the location of an execute-type instruction, since the new contents of the location may be interpreted when resuming execution.
6. Exceptions conditions that occur during the execution of EXECUTE or EXECUTE RELATIVE

LONG are recognized before execution conditions for the target instruction.

EXTRACT ACCESS



The contents of access register R_2 are placed in bit positions 32-63 of general register R_1 . Bits 0-31 of general register R_1 remain unchanged.

Condition Code: The code remains unchanged.

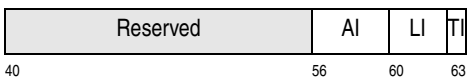
Program Exceptions: None.

EXTRACT CACHE ATTRIBUTE



Information regarding the specified attribute of the storage subsystem is placed into the first-operand location. The first operand is 64 bits.

The second-operand address is not used to address data; rather, the rightmost 24 bits of the address are treated as a code specifying which attribute is returned in general register R_1 . The rightmost 24 bits of the second-operand address are defined as follows:



Attribute Indication (AI): Bit positions 56-59 of the second-operand address contain a 4-bit unsigned integer indicating the cache attribute to be extracted, as follows:

- 0 Extract topology summary
- 1 Extract line size of the cache, in bytes
- 2 Extract total size of the cache, in bytes
- 3 Extract set-associativity level of the cache
- 4-15 Reserved

Level Indication (LI): Bit positions 60-62 of the second-operand address contain a 3-bit unsigned integer indicating the level of the cache for which the cache attribute is to be extracted, with 0 indicating the first-level cache, 1 indicating the second-level cache and so forth. If a cache level is not implemented on a model, its corresponding level indication is reserved.

Type Indication (TI): Bit 63 of the second-operand address indicates the type of cache for which the cache attribute is to be extracted, with 0 indicating the data cache and 1 indicating the instruction cache. When a cache level has a unified data and instruction cache, the same result is returned regardless of the type indication.

When the attribute indication is zero, the level and type indications are ignored.

Bits 0-39 of the second-operand address are ignored. Bits 40-55 of the second-operand address are reserved and should contain zeros. If a reserved bit position in the second-operand address contains one, or if a reserved attribute indication or level indication are specified, bits 0-63 of general register R_1 are set to ones.

The contents of general register R_3 are ignored, however the R_3 field should specify register 0; otherwise, the program may not operate compatibly in the future.

When the attribute indication is zero, a summary of each level of cache is returned in general register R_1 . Each summary field is eight bits, where bits 0-7 of the register contain the summary for the first-level cache, bits 8-15 contain the summary for the second-level cache, and so forth. The contents of an eight-bit summary field are as follows:

Bits	Meaning
0-3	Reserved, stored as zeros
4-5	Cache scope, as follows:
00	Cache does not exist at this level
01	Cache is private to the CPU
10	Cache may be shared by multiple CPUs
11	Reserved
6-7	When bit positions 4-5 contain a nonzero value, bit positions 6-7 contain the cache type, as follows:

Bits	Meaning
00	Separate instruction and data caches exist at this level
01	Only an instruction cache exists at this level
10	Only a data cache exists at this level
11	A unified instruction and data cache exists at this level

When bit positions 4-5 contain zeros, bit positions 6-7 also contain zeros.

Condition Code: The code remains unchanged.

Program Exceptions:

- Operation (if the general-instructions-extension facility is not installed)

Programming Notes:

1. Knowledge of the cache line size is useful when determining the placement of PREFETCH DATA and PREFETCH DATA RELATIVE LONG instructions.
2. The value placed in the first-operand location is model dependent and may differ significantly from one model to another. For example, the size of the data cache on one model may be smaller or larger than the size of the data cache on another model. To ensure compatible operation across multiple models, the program should not rely on any particular value being returned in the first-operand location.
3. When the attribute indication is 0000 binary, bit positions 4-5 of each 8-bit field returned in general register R_1 contain a cache-scope indication. When the cache-scope indication is 10 binary, the cache may be shared by more than one CPU in the configuration. The System Library publication for the model may provide further explanation of cache topology.
4. When the attribute indication is 0000 binary, the summary indications are returned in contiguous 8-bit fields in general register R_1 . Thus if the program scans the resulting summary-indication fields from left to right and encounters an 8-bit field containing all zeros, no further meaningful fields exist.

EXTRACT CPU TIME



The value of the current CPU timer is subtracted from the first operand, and the difference is placed in general register 0. The second operand is placed unchanged in general register 1. The eight bytes at the third operand location replace the contents of general register R₃.

The first and second operands and the results in general registers 0, 1, and R₃ are treated as 64-bit unsigned binary integers.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general register R₃ constitute the address of the third operand, and the contents of bit

positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the register constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address. In the access-register mode, the contents of access register R₃ are used for accessing the third operand.

All operands are fetched before any general register is modified. If the R₃ field designates general register 0, it is unpredictable whether the result placed in the register is the first operand minus the CPU timer or the third operand. If the R₃ field designates general register 1, it is unpredictable whether the result placed in the register is the second operand or the third operand.

The contents of the registers just described are shown in Figure 7-251.

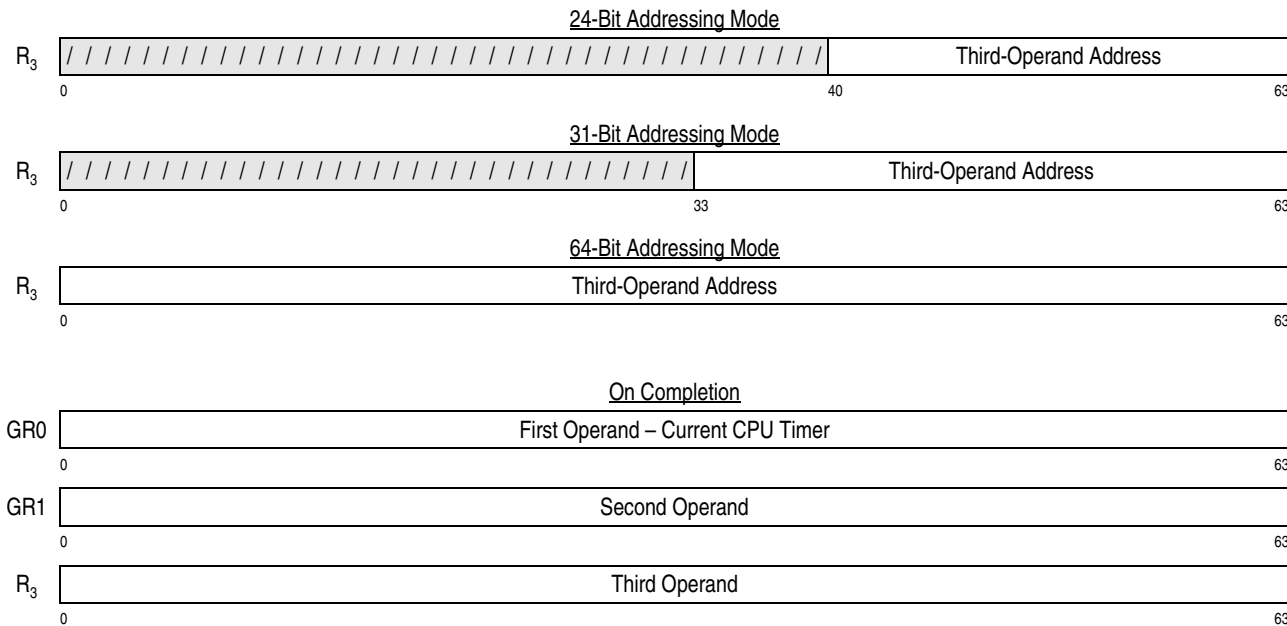


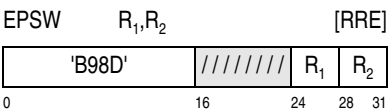
Figure 7-251. General Register Assignment for EXTRACT CPU TIME

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operands 1, 2, and 3)
- Operation (the extract-CPU-time facility is not installed)

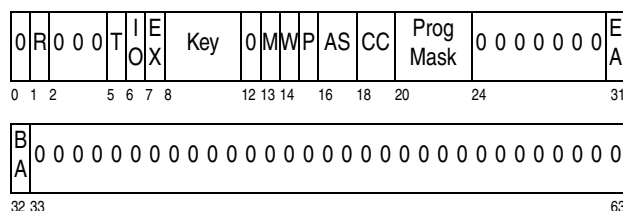
EXTRACT PSW



Bits 0-31 of the current PSW are placed in bit positions 32-63 of the first operand, and bits 0-31 of the

operand remain unchanged. Subsequently, bits 32-63 of the current PSW are placed in bit positions 32-63 of the second operand, and bits 0-31 of the operand remain unchanged. The action associated with the second operand is not performed if the R_2 field is zero.

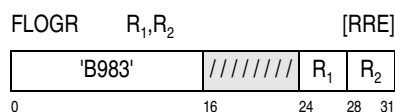
Bits 0-63 of the 128-bit PSW have the following format:



Condition Code: The code remains unchanged.

Program Exceptions: None.

FIND LEFTMOST ONE



Bits 0-63 of general register R_2 are scanned left to right for the leftmost one bit. A 64-bit binary integer designating the bit position of the leftmost one bit, or 64 if there is no one bit, is placed in general register R_1 . When a one bit is found in the second operand, the original contents of general register R_2 , with the leftmost one bit set to zero, are placed in general register $R_1 + 1$. When a one bit is not found, the contents of general register $R_1 + 1$ are set to zeros.

The condition code indicates whether or not a one bit was found in the second operand.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

Resulting Condition Code:

- | | |
|---|------------------|
| 0 | No one bit found |
| 1 | -- |
| 2 | One bit found |
| 3 | -- |

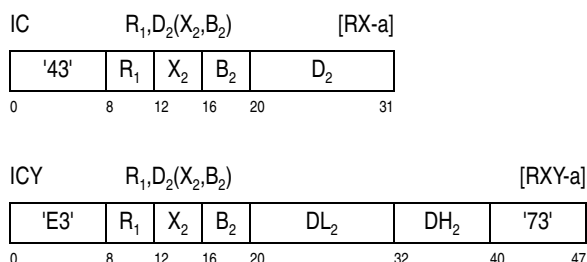
Program Exceptions:

- Operation (if the extended-immediate facility is not installed)
- Specification

Programming Notes:

1. An example of the use of the FIND LEFTMOST ONE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When the R_1 and R_2 fields designate the same register, the original contents of general register R_2 are replaced by the resulting bit-position value, or 64 if no one bit was found.
3. When the R_2 field designates general register $R_1 + 1$, the leftmost one bit (if any) of general register R_2 is set to zero; if no one bit is found, the entire register is set to zero.
4. When the R_2 field designates neither the even nor the odd registers designated by the R_1 field, then general register R_2 is not modified.

INSERT CHARACTER



The byte at the second-operand location is inserted into bit positions 56-63 of general register R_1 . The remaining bits in the register remain unchanged.

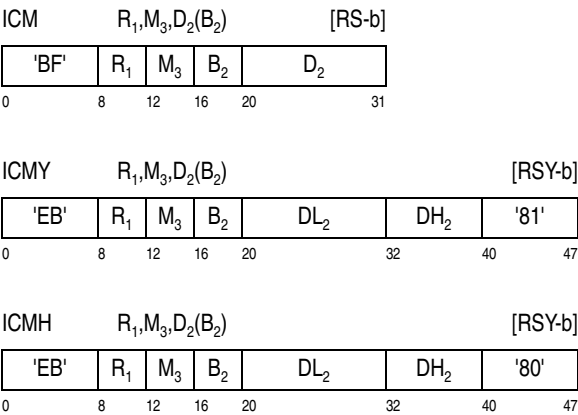
The displacement for IC is treated as a 12-bit unsigned binary integer. The displacement for ICY is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Operation (ICY, if the long-displacement facility is not installed)

INSERT CHARACTERS UNDER MASK



Bytes from contiguous locations beginning at the second-operand address are inserted into general register R_1 under control of a mask.

The contents of the M_3 field are used as a mask. These four bits, left to right, correspond one for one with four bytes, left to right, of general register R_1 . For INSERT CHARACTERS UNDER MASK (ICM, ICMY), the four bytes to which the mask bits correspond are in the low-order half, bit positions 32-63 of general register R_1 . For INSERT CHARACTERS UNDER MASK (ICMH), the four bytes are in the high-order half, bit positions 0-31, of the register. The byte positions corresponding to ones in the mask are filled, left to right, with bytes from successive storage locations beginning at the second-operand address. When the mask is not zero, the length of the second operand is equal to the number of ones in the mask. The bytes in the general register corresponding to zeros in the mask remain unchanged. For ICM and ICMY, bits 0-31 of the register remain unchanged, and, for ICMH, bits 32-63 remain unchanged.

The resulting condition code is based on the mask and on the value of the bits inserted. When the mask is zero or when all inserted bits are zeros, the condition code is set to 0. When the inserted bits are not all zeros, the code is set according to the leftmost bit of the storage operand: if this bit is one, the code is set to 1; if this bit is zero, the code is set to 2.

When the mask is not zero, exceptions associated with storage-operand access are recognized only for the number of bytes specified by the mask. When the mask is zero, access exceptions are recognized for one byte at the second-operand address.

The displacement for ICM is treated as a 12-bit unsigned binary integer. The displacement for ICMY and ICMH is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 All inserted bits zeros, or mask bits all zeros
- 1 Leftmost inserted bit one
- 2 Leftmost inserted bit zero, and not all inserted bits zeros
- 3 --

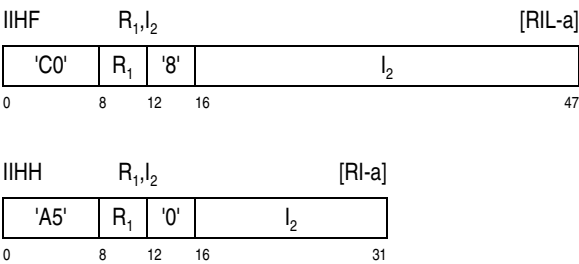
Program Exceptions:

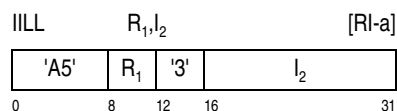
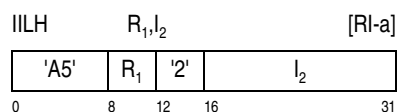
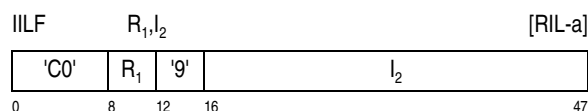
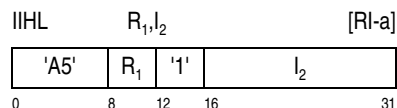
- Access (fetch, operand 2)
- Operation (ICMY, if the long-displacement facility is not installed)

Programming Notes:

1. Examples of the use of the INSERT CHARACTERS UNDER MASK instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The condition code for INSERT CHARACTERS UNDER MASK (ICM, ICMY only) is defined such that, when the mask is 1111, the instruction causes the same condition code to be set as for LOAD AND TEST (LTR only) Thus, the instruction may be used as a storage-to-register load-and-test operation.
3. INSERT CHARACTERS UNDER MASK (ICM, ICMY) with a mask of 1111 or 0001 performs a function similar to that of a LOAD (L) or INSERT CHARACTER (IC) instruction, respectively, with the exception of the condition-code setting. However, the performance of INSERT CHARACTERS UNDER MASK may be slower.

INSERT IMMEDIATE





The second operand is placed in bit positions of the first operand. The remainder of the first operand remains unchanged.

For each instruction, the bit positions of the first operand and that are loaded with the second operand are as follows:

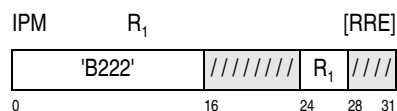
Instruction	Bit Positions Loaded
IIHF	0-31
IIHH	0-15
IIHL	16-31
IILF	32-63
IILH	32-47
IILL	48-63

Condition Code: The code remains unchanged.

Program Exceptions:

- Operation (IIHF and IILF, if the extended-immediate facility is not installed)

INSERT PROGRAM MASK



LOAD IMMEDIATE

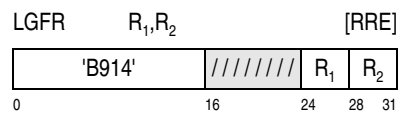
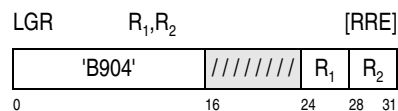
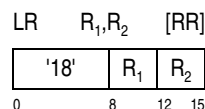
The condition code and program mask from the current PSW are inserted into bit positions 34 and 35 and 36-39, respectively, of general register R_1 . Bits 32 and 33 of the register are set to zeros; bits 0-31 and 40-63 are left unchanged.

Condition Code: The code remains unchanged.

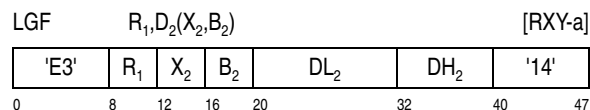
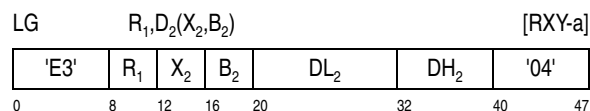
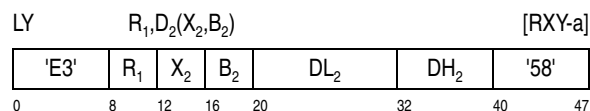
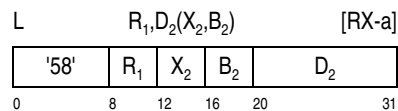
Program Exceptions: None.

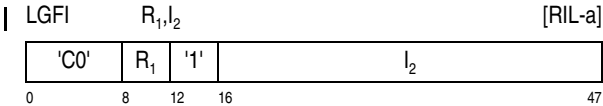
LOAD

Register-and-register formats:

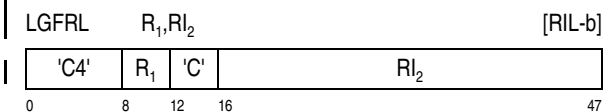
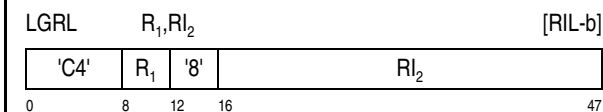
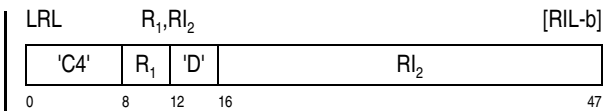


Register-and-storage formats:





LOAD RELATIVE LONG



The second operand is placed unchanged at the first-operand location, except that, for LOAD (LGFR, LGF), LOAD IMMEDIATE (LGFI), and LOAD RELATIVE LONG (LGFRL), it is sign extended.

For LOAD (LR, L, LY) and LOAD RELATIVE LONG (LRL), the operands are 32 bits, and, for LOAD (LGR, LG), and LOAD RELATIVE LONG (LGRL), the operands are 64 bits. For LOAD (LGFR, LGF), LOAD IMMEDIATE (LGFI), and LOAD RELATIVE LONG (LGFRL), the second operand is treated as a 32-bit signed binary integer, and the first operand is treated as a 64-bit signed binary integer.

The displacement for L is treated as a 12-bit unsigned binary integer. The displacement for LY, LG, and LGF is treated as a 20-bit signed binary integer.

For LOAD IMMEDIATE, the contents of the I_2 field form the second operand directly.

For LOAD RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

For LOAD RELATIVE LONG (LRL, LGFRL), the second operand must be aligned on a word boundary, and for LOAD RELATIVE LONG (LGRL), the second operand must be aligned on a doubleword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

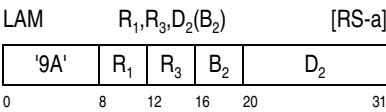
Program Exceptions:

- Access (fetch, operand 2 of L, LY, LG, LGF, LRL, LGRL, and LGFRL only)
- Operation (LY, if the long-displacement facility is not installed; LGFI, if the extended-immediate facility is not installed; LRL, LGRL, LGFRL, if the general-instructions-extension facility is not installed)
- Specification (LRL, LGFRL, LGRL only)

Programming Notes:

1. An example of the use of the LOAD instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. For LOAD RELATIVE LONG, the second operand must be aligned on an integral boundary corresponding to the operand's size.
3. When LOAD RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.

LOAD ACCESS MULTIPLE



The set of access registers starting with access register R_1 and ending with access register R_3 is loaded from the locations designated by the second-operand address.

The storage area from which the contents of the access registers are obtained starts at the location designated by the second-operand address and continues through as many storage words as the number

of access registers specified. The access registers are loaded in ascending order of their register numbers, starting with access register R_1 and continuing up to and including access register R_3 , with access register 0 following access register 15.

The displacement for LAM is treated as a 12-bit unsigned binary integer. The displacement for LAMY is treated as a 20-bit signed binary integer.

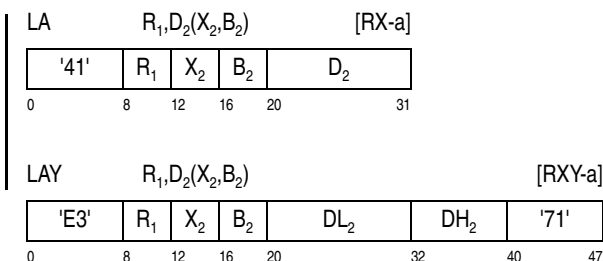
The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Operation (LAMY, if the long-displacement facility is not installed)
- Specification

LOAD ADDRESS



The address specified by the X_2 , B_2 , and D_2 fields is placed in general register R_1 . The address computation follows the rules for address arithmetic.

In the 24-bit addressing mode, the address is placed in bit positions 40-63, bits 32-39 are set to zeros, and bits 0-31 remain unchanged. In the 31-bit addressing mode, the address is placed in bit positions 33-63, bit 32 is set to zero, and bits 0-31 remain unchanged. In the 64-bit addressing mode, the address is placed in bit positions 0-63.

The displacement for LA is treated as a 12-bit unsigned binary integer. The displacement for LAY is treated as a 20-bit signed binary integer.

No storage references for operands take place, and the address is not inspected for access exceptions.

Condition Code: The code remains unchanged.

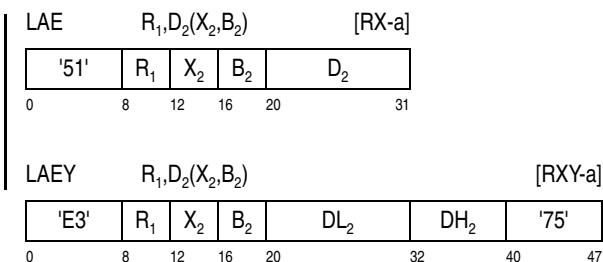
Program Exceptions:

- Operation (LAY if the long-displacement facility is not installed)

Programming Notes:

1. An example of the use of the LOAD ADDRESS instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. LOAD ADDRESS may be used to increment the rightmost bits of a general register, other than register 0, by the contents of the D_2 field of the instruction. LOAD ADDRESS (LAY) may also be used to decrement the rightmost bits of a register, other than register 0. The register to be incremented should be designated by R_1 and by either X_2 (with B_2 set to zero) or B_2 (with X_2 set to zero). The instruction increments 24 bits in the 24-bit addressing mode, 31 bits in the 31-bit addressing mode, and 64 bits in the 64-bit addressing mode.

LOAD ADDRESS EXTENDED



The address specified by the X_2 , B_2 , and D_2 fields is placed in general register R_1 . Access register R_1 is loaded with a value that depends on the current value of the address-space-control bits, bits 16 and 17 of the PSW. If the address-space-control bits are 01 binary, the value placed in the access register also depends on whether the B_2 field is zero or non-zero.

The address computation follows the rules for address arithmetic. In the 24-bit addressing mode, the address is placed in bit positions 40-63 of general register R_1 , bits 32-39 are set to zeros, and bits 0-31 remain unchanged. In the 31-bit addressing mode, the address is placed in bit positions 33-63, bit 32 is set to zero, and bits 0-31 remain unchanged. In the

64-bit addressing mode, the address is placed in bit positions 0-63.

The value placed in access register R_1 is as shown in the following table:

PSW Bits 16 and 17	Value Placed in Access Register R_1
00	00000000 hex (zeros in bit positions 0-31)
10	00000001 hex (zeros in bit positions 0-30 and one in bit position 31)
01	If B_2 field is zero: 00000000 hex (zeros in bit positions 0-31) If B_2 field is nonzero: Contents of access register B_2
11	00000002 hex (zeros in bit positions 0-29 and 31, and one in bit position 30)

However, when PSW bits 16 and 17 are 01 binary and the B_2 field is nonzero, bit positions 0-6 of access register B_2 must contain all zeros; otherwise, the results in general register R_1 and access register R_1 are unpredictable.

The displacement for LAE is treated as a 12-bit unsigned binary integer. The displacement for LAEY is treated as a 20-bit signed binary integer.

No storage references for operands take place, and the address is not inspected for access exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Operation (LAEY; if the general-instructions-extension facility is not installed)

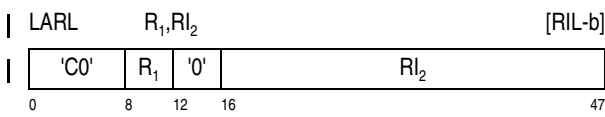
Programming Notes:

1. When DAT is on, the different values of the address-space-control bits correspond to translation modes as follows:

PSW Bits 16 and 17	Translation Mode
00	Primary-space mode
10	Secondary-space mode
01	Access-register mode
11	Home-space mode

2. In the access-register mode, the value 00000000 hex in an access register designates the primary address space, and the value 00000001 hex designates the secondary address space. The value 00000002 hex designates the home address space if the control program assigns entry 2 of the dispatchable-unit access list as designating the home address space and places a zero access-list-entry sequence number (ALESN) in that entry.

LOAD ADDRESS RELATIVE LONG



The address specified by the I_2 field is placed in general register R_1 . The address computation follows the rules for the branch address of BRANCH RELATIVE ON CONDITION LONG and BRANCH RELATIVE AND SAVE LONG.

In the 24-bit addressing mode, the address is placed in bit positions 40-63, bits 32-39 are set to zeros, and bits 0-31 remain unchanged. In the 31-bit addressing mode, the address is placed in bit positions 33-63, bit 32 is set to zero, and bits 0-31 remain unchanged. In the 64-bit addressing mode, the address is placed in bit positions 0-63.

The contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the computed address.

No storage references for operands take place, and the address is not inspected for access exceptions.

Condition Code: The code remains unchanged.

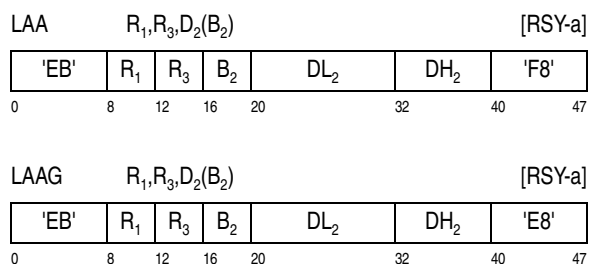
Program Exceptions: None.

Programming Notes:

1. Only even addresses (halfword addresses) can be generated. If an odd address is desired, LOAD ADDRESS can be used to add one to an address formed by LOAD ADDRESS RELATIVE LONG.

- When LOAD ADDRESS RELATIVE LONG is the target of an execute-type instruction, the address produced is relative to the location of the LOAD ADDRESS RELATIVE LONG instruction, not of the execute-type instruction. This is consistent with the operation of the relative-branch instructions.

LOAD AND ADD



The second operand is added to the third operand, and the sum is placed at the second-operand location. Subsequently, the original contents of the second operand (prior to the addition) are placed unchanged at the first-operand location.

For LAA, the operands are treated as being 32-bit signed binary integers. For LAAG, the operands are treated as being 64-bit signed binary integers.

The fetch of the second operand for purposes of loading and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs. A specific-operand-serialization operation is performed.

The displacement is treated as a 20-bit signed binary integer.

The second operand of LAA must be designated on a word boundary. The second operand of LAAG must be designated on a doubleword boundary. Otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

- Access (fetch and store, operand 2)
- Fixed-point overflow
- Operation (if the interlocked-access facility is not installed)
- Specification

Programming Notes:

- Except for the case where the R₁ and R₃ fields designate the same register, general register R₃ is unchanged.
- The operation of LOAD AND ADD, LOAD AND ADD LOGICAL, LOAD AND AND, LOAD AND EXCLUSIVE OR, and LOAD AND OR may be expressed as follows.

```
temp ← operand_2;
operand_2 ← operand_2 OP operand_3;
operand_1 ← temp;
```

OP represents the arithmetic or logical operation being performed by the instruction.

LOAD AND ADD LOGICAL



The second operand is added to the third operand, and the sum is placed at the second-operand location. Subsequently, the original contents of the second operand (prior to the addition) are placed unchanged at the first-operand location.

For LAAL, the operands are treated as being 32-bit unsigned binary integers. For LAALG, the operands are treated as being 64-bit unsigned binary integers.

The fetch of the second operand for purposes of loading and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs. A specific-operand-serialization operation is performed.

The displacement is treated as a 20-bit signed binary integer.

The second operand of LAAL must be designated on a word boundary. The second operand of LAALG must be designated on a doubleword boundary. Otherwise, a specification exception is recognized.

Resulting Condition Code:

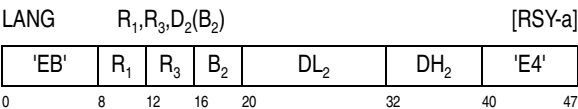
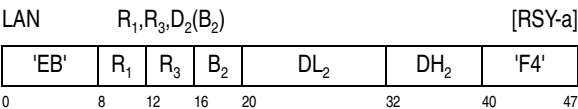
- 0 Result zero; no carry
- 1 Result not zero; no carry
- 2 Result zero; carry
- 3 Result not zero; carry

Program Exceptions:

- Access (fetch and store, operand 2)
- Operation (if the interlocked-access facility is not installed)
- Specification

Programming Note: See the programming notes for LOAD AND ADD.

LOAD AND AND



The AND of the second operand and third operand is placed at the second-operand location. Subsequently, the original contents of the second operand (prior to the AND operation) are placed unchanged at the first-operand location.

For LAN, the operands are 32 bits. For LANG, the operands are 64 bits.

The connective AND is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the corresponding bit positions in both operands contain ones; otherwise, the result bit is set to zero.

The fetch of the second operand for purposes of loading and the store into the second-operand location appear to be a block-concurrent interlocked-

update reference as observed by other CPUs. A specific-operand-serialization operation is performed.

The displacement is treated as a 20-bit signed binary integer.

The second operand of LAN must be designated on a word boundary. The second operand of LANG must be designated on a doubleword boundary. Otherwise, a specification exception is recognized.

Resulting Condition Code:

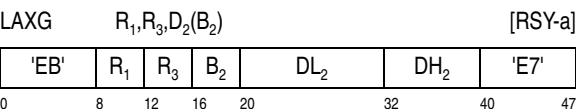
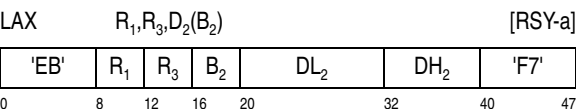
- 0 Result zero
- 1 Result not zero
- 2 --
- 3 --

Program Exceptions:

- Access (fetch and store, operand 2)
- Operation (if the interlocked-access facility is not installed)
- Specification

Programming Note: See the programming notes for LOAD AND ADD.

LOAD AND EXCLUSIVE OR



The EXCLUSIVE OR of the second operand and third operand is placed at the second-operand location. Subsequently, the original contents of the second operand (prior to the EXCLUSIVE OR operation) are placed unchanged at the first-operand location.

For LAX, the operands are 32 bits. For LAXG, the operands are 64 bits.

The connective exclusive OR is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the bits in the corresponding

bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

The fetch of the second operand for purposes of loading and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs. A specific-operand-serialization operation is performed.

The displacement is treated as a 20-bit signed binary integer.

The second operand of LAX must be designated on a word boundary. The second operand of LAXG must be designated on a doubleword boundary. Otherwise, a specification exception is recognized.

Resulting Condition Code:

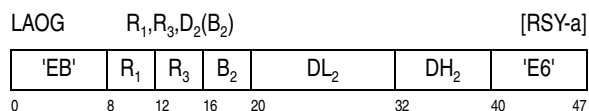
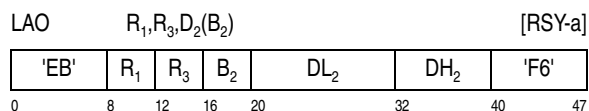
- 0 Result zero
- 1 Result not zero
- 2 --
- 3 --

Program Exceptions:

- Access (fetch and store, operand 2)
- Operation (if the interlocked-access facility is not installed)
- Specification

Programming Note: See the programming notes for LOAD AND ADD.

LOAD AND OR



The OR of the second operand and third operand is placed at the second-operand location. Subsequently, the original contents of the second operand (prior to the OR operation) are placed unchanged at the first-operand location.

For LAO, the operands are 32 bits. For LAOG, the operands are 64 bits.

The connective OR is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero.

The fetch of the second operand for purposes of loading and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs. A specific-operand-serialization operation is performed.

The displacement is treated as a 20-bit signed binary integer.

The second operand of LAO must be designated on a word boundary. The second operand of LAOG must be designated on a doubleword boundary. Otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result zero
- 1 Result not zero
- 2 --
- 3 --

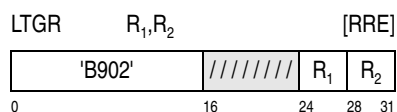
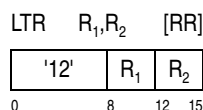
Program Exceptions:

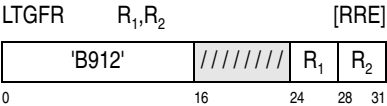
- Access (fetch and store, operand 2)
- Operation (if the interlocked-access facility is not installed)
- Specification

Programming Note: See the programming notes for LOAD AND ADD.

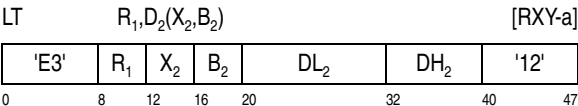
LOAD AND TEST

Register-and-register formats:





Register-and-storage formats:



The second operand is placed unchanged at the first-operand location, except that, for LTGF and LTGFR, it is sign extended. The sign and magnitude of the second operand, treated as a signed binary integer, are indicated in the condition code.

For LT and LTR, the operands are 32 bits, and, for LTG and LTGR, the operands are 64 bits. For LTGF and LTGFR, the second operand is 32 bits, and the first operand is treated as a 64-bit signed binary integer.

Resulting Condition Code:

- 0 Result zero
- 1 Result less than zero
- 2 Result greater than zero
- 3 --

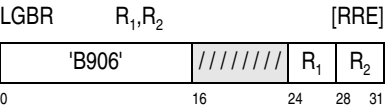
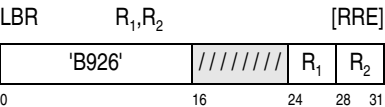
Program Exceptions:

- Access (fetch, operand 2 of LT, LTG, and LTGF only)
- Operation (LT and LTG, if the extended-immediate facility is not installed; LTGF, if the general-instructions-extension facility is not installed)

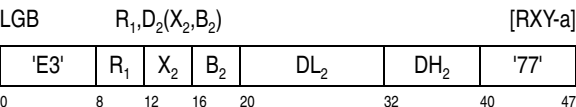
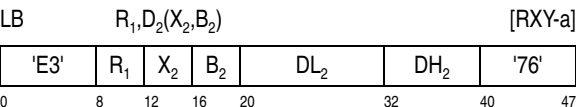
Programming Note: For LOAD AND TEST (LTR and LTGR) when the R₁ and R₂ fields designate the same register, the operation is equivalent to a test without data movement.

LOAD BYTE

Register-and-register formats:



Register-and-storage formats:



The second operand is sign extended and placed at the first-operand location. The second operand is one byte in length and is treated as an eight-bit signed binary integer. For LOAD BYTE (LB and LGB), the second operand is a byte in storage. For LOAD BYTE (LBR and LGBR), the second operand is in bits 56-63 of general register R₂.

For LOAD BYTE (LB and LBR), the first operand is treated as a 32-bit signed binary integer. For LOAD BYTE (LGB and LGBR), the first operand is treated as a 64-bit signed binary integer.

The displacement for LB and LGB is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of LB and LGB)
- Operation (LB and LGB, if the long-displacement facility is not installed; LBR and LGBR, if the extended-immediate facility is not installed)

LOAD BYTE HIGH



The second operand is sign extended and placed at the first-operand location. The second operand is one byte in length and is treated as an eight-bit signed binary integer.

The first operand is treated as a 32-bit signed binary integer in bits 0-31 of general register R₁; bits 32-63 of the register are unchanged.

The displacement is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Operation (if the high-word facility is not installed)

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

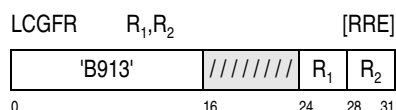
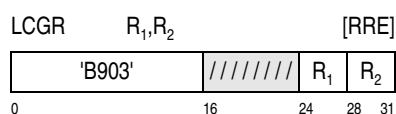
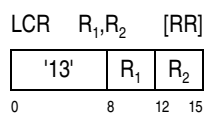
- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

- Fixed-point overflow (LCR and LCGR only)

Programming Note: The operation complements all numbers. Zero remains unchanged. For LCR or LCGR, the maximum negative 32-bit number or 64-bit number, respectively, remains unchanged, and an overflow condition occurs when the number is complemented. LCGFR complements the maximum negative 32-bit number without recognizing overflow.

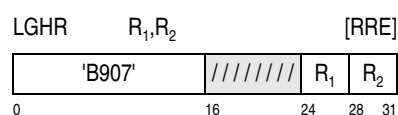
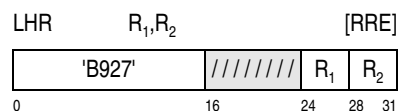
LOAD COMPLEMENT



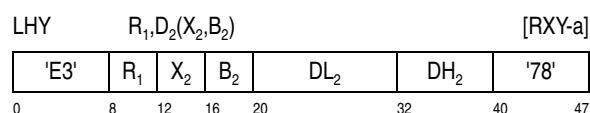
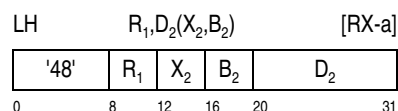
The two's complement of the second operand is placed at the first-operand location. For LOAD COMPLEMENT (LCR), the second operand and result are treated as 32-bit signed binary integers. For LOAD COMPLEMENT (LCGR), they are treated as 64-bit signed binary integers. For LOAD COMPLEMENT (LCGFR), the second operand is treated as a 32-bit signed binary integer, and the result is treated as a 64-bit signed binary integer.

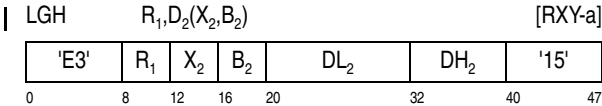
LOAD HALFWORD

Register-and-register formats:

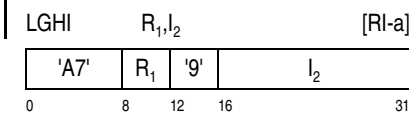
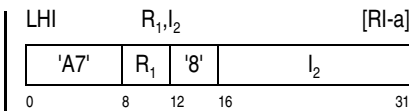


Register-and-storage formats:

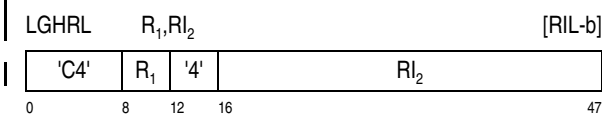
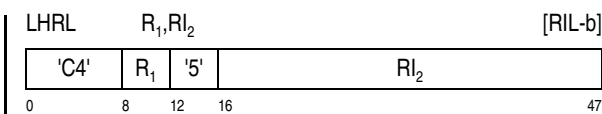




LOAD HALFWORD IMMEDIATE



LOAD HALFWORD RELATIVE LONG



The second operand is sign extended and placed at the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. For LOAD HALFWORD (LH, LHY, and LGH) and LOAD HALFWORD RELATIVE LONG, the second operand is in storage. For LOAD HALFWORD (LHI and LGHI), the second operand is the immediate field of the instruction. For LOAD HALFWORD (LHR and LGHR), the second operand is in bits 48-63 of general register R_2 .

For LOAD HALFWORD (LH, LHR, and LHY), LOAD HALFWORD IMMEDIATE (LHI), and LOAD HALFWORD RELATIVE LONG (LHRL), the first operand is treated as a 32-bit signed binary integer. For LOAD HALFWORD (LGH and LGHR), LOAD HALFWORD IMMEDIATE (LGHI), and LOAD HALFWORD RELATIVE LONG (LGHRL), the first operand is treated as a 64-bit signed binary integer.

The displacement for LH is treated as a 12-bit unsigned binary integer. The displacement for LHY and LGH is treated as a 20-bit signed binary integer.

For LOAD HALFWORD RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of LH, LHY, LGH, LHRL and LGHRL)
- Operation (LHY, if the long-displacement facility is not installed; LHR and LGHR, if the extended-immediate facility is not installed; LHRL and LGHRL, if the general-instructions-extension facility is not installed)

Programming Notes:

1. An example of the use of the LOAD HALFWORD instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. For LOAD HALFWORD RELATIVE LONG, the second operand is necessarily aligned on an integral boundary corresponding to the operand's size.
3. When LOAD HALFWORD RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.

LOAD HALFWORD HIGH



The second operand is sign extended and placed at the first-operand location. The second operand is two bytes in length and is treated as an 16-bit signed binary integer.

The first operand is treated as a 32-bit signed binary integer in bits 0-31 of general register R_1 ; bits 32-63 of the register are unchanged.

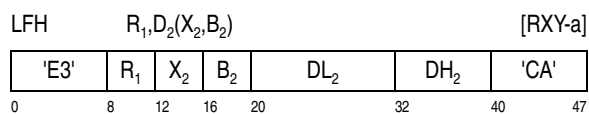
The displacement is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Operation (if the high-word facility is not installed)

LOAD HIGH



The second operand is placed unchanged at the first-operand location. The second operand is 32 bits, and the first operand is in bits 0-31 of general register R_1 ; bits 32-63 of the register are unchanged.

The displacement is treated as a 20-bit signed binary integer.

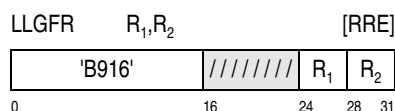
Condition Code: The code remains unchanged.

Program Exceptions:

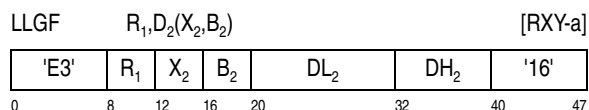
- Access (fetch, operand 2)
- Operation (if the high-word facility is not installed)

LOAD LOGICAL

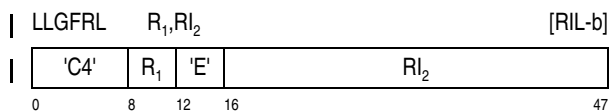
Register-and-register format:



Register-and-storage format:



LOAD LOGICAL RELATIVE LONG



The four-byte second operand is placed in bit positions 32-63 of general register R_1 , and zeros are placed in bit positions 0-31 of general register R_1 .

For LOAD LOGICAL (LLGFR), the second operand is in bit positions 32-63 of general register R_2 .

For LOAD LOGICAL RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

For LOAD LOGICAL RELATIVE LONG, the second operand must be aligned on a word boundary; otherwise, a specification exception is recognized.

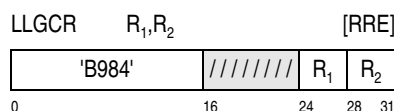
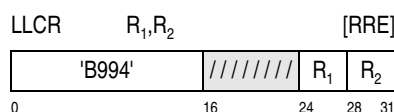
Condition Code: The code remains unchanged.

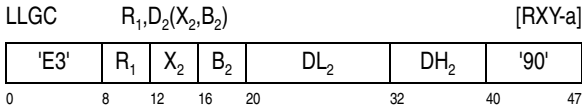
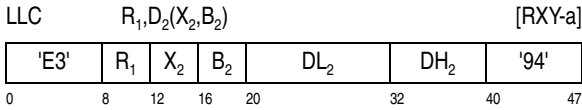
Program Exceptions:

- Access (fetch, operand 2 of LLGF and LLGFRL only)
- Operation (LLGFRL, if the general-instructions-extension facility is not installed)
- Specification (LLGFRL only)

LOAD LOGICAL CHARACTER

Register-and-register formats:



Register-and-storage formats:

The one-byte second operand is placed in bit positions 56-63 of general register R_1 . For LOAD LOGICAL CHARACTER (LLGC, LLGCR), zeros are placed in bit positions 0-55 of general register R_1 . For LOAD LOGICAL CHARACTER (LLC, LLCR), zeros are placed in bit positions 32-55 of general register R_1 ; bit positions 0-31 of general register R_1 are unchanged.

For LOAD LOGICAL CHARACTER (LLC, LLGC), the second operand is in storage. For LOAD LOGICAL CHARACTER (LLCR, LLGCR), the second operand is in bits 56-63 of general register R_2 .

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of LLC and LLGC)
- Operation (LLC, LLCR, and LLGCR, if the extended-immediate facility is not installed)

LOAD LOGICAL CHARACTER HIGH

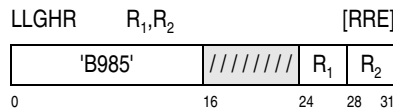
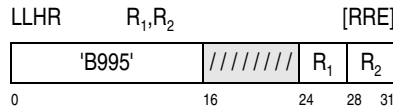
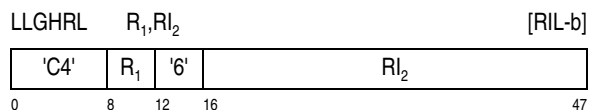
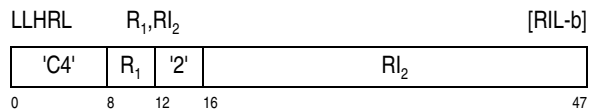
The one-byte second operand is placed in bit positions 24-31 of general register R_1 , and zeros are placed in bit positions 0-23 of general register R_1 ; bit positions 32-63 of general register R_1 are unchanged.

The displacement is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Operation (if the high-word facility is not installed)

LOAD LOGICAL HALFWORD**Register-and-register formats:****Register-and-storage formats:****LOAD LOGICAL HALFWORD RELATIVE LONG**

The two-byte second operand is placed in bit positions 48-63 of general register R_1 . For LOAD LOGICAL HALFWORD (LLGH, LLGHR) and LOAD LOGICAL HALFWORD RELATIVE LONG (LLGHRL), zeros are placed in bit positions 0-47 of general register R_1 . For LOAD LOGICAL HALFWORD (LLH, LLHR) and LOAD LOGICAL HALFWORD RELATIVE LONG (LLHRL), zeros are placed

in bit positions 32-47 of general register R_1 ; bit positions bits 0-31 of general register R_1 are unchanged.

For LOAD LOGICAL HALFWORD (LLGH, LLH) and LOAD LOGICAL HALFWORD RELATIVE LONG, the second operand is in storage. For LOAD LOGICAL HALFWORD (LLGHR, LLHR), the second operand is in bits 48-63 of general register R_2 .

For LOAD LOGICAL HALFWORD RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

Condition Code: The code remains unchanged.

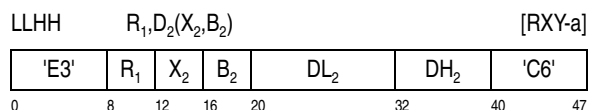
Program Exceptions:

- Access (fetch, operand 2 of LLGH, LLH, LLHRL, and LLGHRL)
- Operation (LLGHR, LLH, and LLHR, if the extended-immediate facility is not installed; LLHRL and LLGHRL, if the general-instructions-extension facility is not installed)

Programming Notes:

1. For LOAD LOGICAL HALFWORD RELATIVE LONG, the second operand is necessarily aligned on an integral boundary corresponding to the operand's size.
2. When LOAD LOGICAL HALFWORD RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.

LOAD LOGICAL HALFWORD HIGH



The two-byte second operand is placed in bit positions 16-31 of general register R_1 , and zeros are placed in bit positions 0-15 of general register R_1 ; bit positions 32-63 of general register R_1 are unchanged.

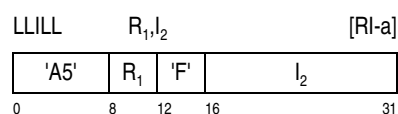
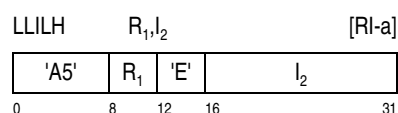
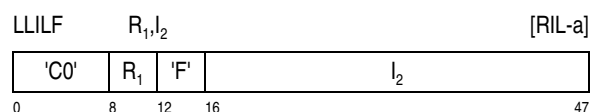
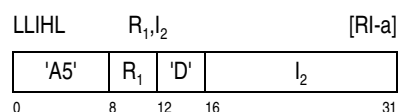
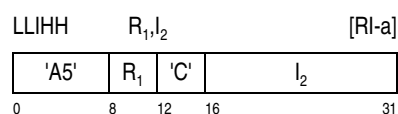
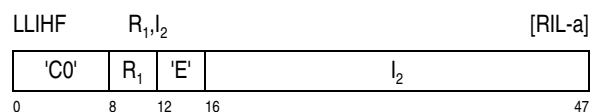
The displacement is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Operation (if the high-word facility is not installed)

LOAD LOGICAL IMMEDIATE



The second operand is placed in bit positions of the first operand. The remainder of the first operand is set to zeros.

For each instruction, the bit positions of the first operand and that are loaded with the second operand are as follows:

Instruction	Bit Positions Loaded
LLIHF	0-31
LLIHH	0-15
LLIHL	16-31
LLILF	32-63
LLILH	32-47
LLILL	48-63

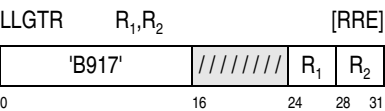
Condition Code: The code remains unchanged.

Program Exceptions:

- Operation (LLIHF and LLILF, if the extended-immediate facility is not installed)

LOAD LOGICAL THIRTY ONE BITS

Register-and-register format:



Register-and-storage format:



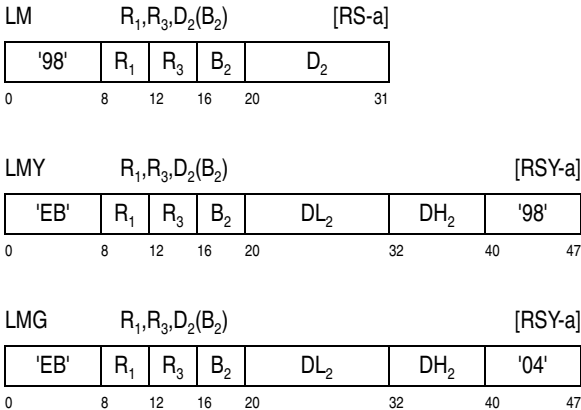
For LLGTR, bits 33-63 of general register R₂, with 33 zeros appended on the left, are placed in general register R₁. For LLGT, bits 1-31 of the four bytes at the second-operand location, with 33 zeros appended on the left, are placed in general register R₁.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of LLGT only)

LOAD MULTIPLE



Bit positions of the set of general registers starting with general register R₁ and ending with general register R₃ are loaded from storage beginning at the location designated by the second-operand address and continuing through as many locations as needed.

For LOAD MULTIPLE (LM, LMY), bit positions 32-63 of the general registers are loaded from successive four-byte fields beginning at the second-operand address, and bits 0-31 of the registers remain unchanged. For LOAD MULTIPLE (LMG), bit positions 0-63 of the general registers are loaded from successive eight-byte fields beginning at the second-operand address.

The general registers are loaded in the ascending order of their register numbers, starting with general register R₁ and continuing up to and including general register R₃, with general register 0 following general register 15.

The displacement for LM is treated as a 12-bit unsigned binary integer. The displacement for LMY and LMG is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

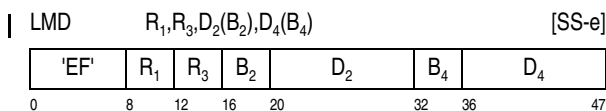
Program Exceptions:

- Access (fetch, operand 2)
- Operation (LMY, if the long-displacement facility is not installed)

Programming Note: All combinations of register numbers specified by R₁ and R₃ are valid. When the register numbers are equal, only four bytes, for LM or

LMY or eight bytes, for LMG, are transmitted. When the number specified by R_3 is less than the number specified by R_1 , the register numbers wrap around from 15 to 0.

LOAD MULTIPLE DISJOINT



Bit positions 0-31 of the set of general registers starting with general register R_1 and ending with general register R_3 are loaded from storage beginning at the location designated by the second-operand address and continuing through as many locations as needed. Bit positions 32-63 of the same registers are similarly loaded from storage beginning at the location designated by the fourth-operand address.

The general registers are loaded in the ascending order of their register numbers, starting with general register R_1 and continuing up to and including general register R_3 , with general register 0 following general register 15.

Condition Code: The code remains unchanged.

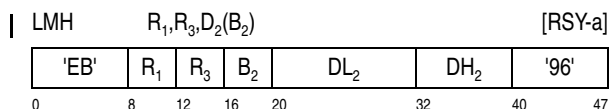
Program Exceptions:

- Access (fetch, operands 2 and 4)

Programming Notes:

- All combinations of register numbers specified by R_1 and R_3 are valid. When the register numbers are equal, only eight bytes are transmitted. When the number specified by R_3 is less than the number specified by R_1 , the register numbers wrap around from 15 to 0.
- The second-operand and fourth-operand addresses are computed before the contents of any register are changed.
- The combination of a LOAD MULTIPLE instruction and a LOAD MULTIPLE HIGH instruction provides equal or better performance than a LOAD MULTIPLE DISJOINT instruction for the same register range. LOAD MULTIPLE DISJOINT is for use when the second or fourth operand must be addressed by means of one of the registers loaded.

LOAD MULTIPLE HIGH



The high-order halves, bit positions 0-31, of the set of general registers starting with general register R_1 and ending with general register R_3 are loaded from storage beginning at the location designated by the second-operand address and continuing through as many locations as needed, that is, bit positions 0-31 are loaded from successive four-byte fields beginning at the second-operand address. Bits 32-63 of the registers remain unchanged.

The general registers are loaded in the ascending order of their register numbers, starting with general register R_1 and continuing up to and including general register R_3 , with general register 0 following general register 15.

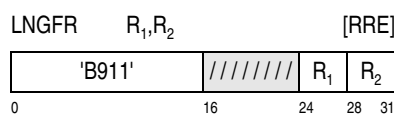
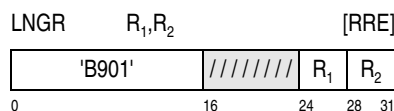
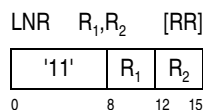
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)

Programming Note: All combinations of register numbers specified by R_1 and R_3 are valid. When the register numbers are equal, only four bytes are transmitted. When the number specified by R_3 is less than the number specified by R_1 , the register numbers wrap around from 15 to 0.

LOAD NEGATIVE



The two's complement of the absolute value of the second operand is placed at the first-operand location. For LOAD NEGATIVE (LNR), the second operand and result are treated as 32-bit signed binary integers, and, for LOAD NEGATIVE (LNGR), they are treated as 64-bit signed binary integers. For LOAD NEGATIVE (LNGFR), the second operand is treated as a 32-bit signed binary integer, and the result is treated as a 64-bit signed binary integer.

Resulting Condition Code:

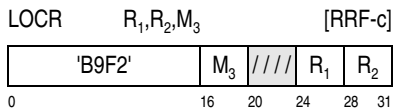
- 0 Result zero
- 1 Result less than zero
- 2 --
- 3 --

Program Exceptions: None.

Programming Note: The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged.

LOAD ON CONDITION

Register-and-register formats:



Register-and-storage formats:



The second operand is placed unchanged at the first-operand location if the condition code has one of the values specified by M_3 ; otherwise, the first operand remains unchanged.

For LOC and LOCR, the first and second operands are 32 bits, and for LOCG and LOCGR, the first and second operands are 64 bits.

The M_3 field is used as a four-bit mask. The four condition codes (0, 1, 2, and 3) correspond, left to right, with the four bits of the mask, as follows:

Condition Code	Mask Position Value
0	8
1	4
2	2
3	1

The current condition code is used to select the corresponding mask bit. If the mask bit selected by the condition code is one, the load is performed. If the mask bit selected is zero, the load is not performed.

The displacement for LOC and LOCG is treated as a 20-bit signed binary integer.

For LOC and LOCG, when the condition specified by the M_3 field is not met (that is, the load operation is not performed), it is model dependent whether an access exception, or PER zero-address detection is recognized for the second operand.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of LOC and LOCG)
- Operation (if the load/store-on-condition facility is not installed)

Programming Notes:

1. When the M_3 field contain all zeros and no exception condition exists, the instruction acts as a NOP. When the M_3 field contains all ones and no exception condition exists, the load operation is always performed. However, these are not the preferred means of implementing a NOP or unconditional load, respectively.
2. For LOC and LOCG, when the condition specified by the M_3 field is not met, it is model dependent whether the second operand is brought into the cache.

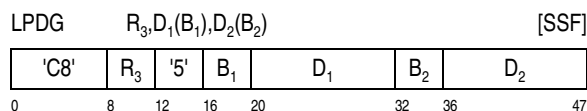
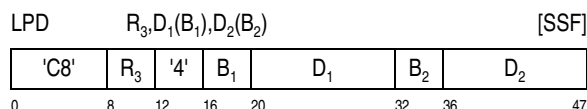
3. LOAD ON CONDITION provides a function similar to that of a separate BRANCH ON CONDITION instruction followed by a LOAD instruction, except that LOAD ON CONDITION does not provide an index register. For example, the following two instruction sequences are equivalent.

```

      LOCG      15,256(7),8      BC      7,SKIP
                                LG      15,256(0,7)
                                SKIP DS  0H
  
```

On models that implement predictive branching, the combination of the BRANCH ON CONDITION and LOAD instructions may perform somewhat better than the LOAD ON CONDITION instruction when the CPU is able to successfully predict the branch condition. However, on models where the CPU is not able to successfully predict the branch condition, such as when the condition is more random, the LOAD ON CONDITION instruction may provide significant performance improvement.

LOAD PAIR DISJOINT



General register R_3 designates the even numbered register of an even/odd register pair.

The first operand is placed unchanged into the even-numbered register of the third operand, and the second operand is placed unchanged into odd-numbered register of the third operand. The condition code indicates whether the first and second operands appear to be fetched by means of block-concurrent interlocked fetch.

For LPD, the first and second operands are words in storage, and the third operand is in bits 32-63 of general registers R_3 and $R_3 + 1$; bits 0-31 of the registers are unchanged. For LPDG, the first and second operands are doublewords in storage, and the third operand is in bits 0-63 of general registers R_3 and $R_3 + 1$.

When, as observed by other CPUs, the first and second operands appear to be fetched by means of block-concurrent interlocked fetch, condition code 0 is set. When the first and second operands do not appear to be fetched by means of block-concurrent interlocked update, condition code 3 is set. The third operand is loaded regardless of the condition code.

The displacement of the first and second operands is treated as a 12-bit unsigned binary integer.

The first and second operands of LPD must be designated on a word boundary. The first and second operands of LPDG must be designated on a double-word boundary. General register R_3 must designate the even numbered register. Otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Register pair loaded by means of interlocked fetch
- 1 --
- 2 --
- 3 Register pair not loaded by means of interlocked fetch

Program Exceptions:

- Access (fetch, operands 1 and 2)
- Operation (if the interlocked-access facility is not installed)
- Specification

Programming Notes:

1. The setting of the condition code is dependent upon storage accesses by other CPUs in the configuration.
2. When the resulting condition code is 3, the program may branch back to reexecute the LOAD PAIR DISJOINT instruction. However, after repeated unsuccessful attempts to attain an interlocked fetch, the program should use an alternate means of serializing access to the storage operands. It is recommended that the program reexecute the LOAD PAIR DISJOINT no more than 10 times before branching to the alternate path.
3. The program should be able to accommodate a situation where condition code 0 is never set.

LOAD PAIR FROM QUADWORD



The quadword second operand is loaded into the first-operand location. The second operand appears to be fetched with quadword concurrency as observed by other CPUs. The left doubleword of the quadword is loaded into general register R₁, and the right doubleword is loaded into general register R₁ + 1.

The R₁ field designates an even-odd pair of general registers and must designate an even-numbered register. The second operand must be designated on a quadword boundary. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

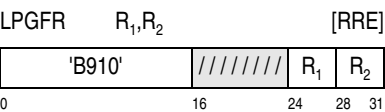
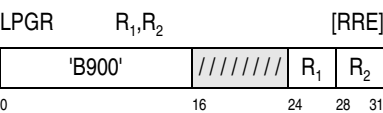
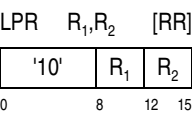
Program Exceptions:

- Access (fetch, operand 2)
- Specification

Programming Notes:

1. The LOAD MULTIPLE (LM or LMG) instruction does not necessarily provide quadword-concurrent access.
2. The performance of LOAD PAIR FROM QUADWORD on some models may be significantly slower than that of LOAD MULTIPLE (LMG). Unless quadword consistency is required, LMG should be used instead of LPQ.

LOAD POSITIVE



The absolute value of the second operand is placed at the first-operand location. For LOAD POSITIVE (LPR), the second operand and result are treated as 32-bit signed binary integers, and, for LOAD POSITIVE (LPGR), they are treated as 64-bit signed binary integers. For LOAD POSITIVE (LPGFR), the second operand is treated as a 32-bit signed binary integer, and the result is treated as a 64-bit signed binary integer.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 --
- 2 Result greater than zero; no overflow
- 3 Overflow

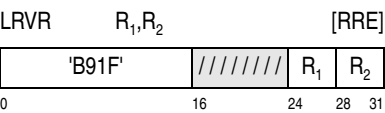
Program Exceptions:

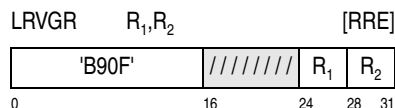
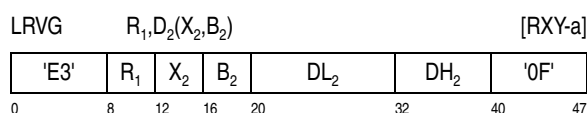
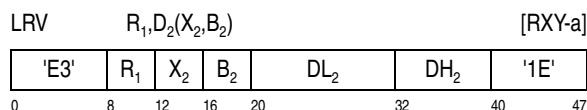
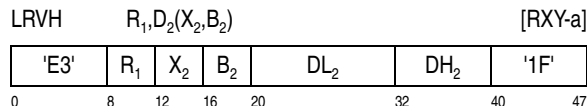
- Fixed-point overflow (LPR and LPGR only)

Programming Note: The operation complements negative numbers; positive numbers and zero remain unchanged. For LPR or LPGR, an overflow condition occurs when the maximum negative 32-bit number or 64-bit number, respectively, is complemented; the number remains unchanged. LPGFR complements the maximum negative 32-bit number without recognizing overflow.

LOAD REVERSED

Register-and-register formats:



**Register-and-storage formats:**

The second operand is placed at the first-operand location with the left-to-right sequence of the bytes reversed.

For LOAD REVERSED (LRVH), the second operand is two bytes, the result is placed in bit positions 48-63 of general register R_1 , and bits 0-47 of the register remain unchanged.

For LOAD REVERSED (LRVR, LRV), the second operand is four bytes, the result is placed in bit positions 32-63 of general register R_1 , and bits 0-31 of the register remain unchanged. For LOAD REVERSED (LRVR), the second operand is in bit positions 32-63 of general register R_2 .

For LOAD REVERSED (LRVGR, LRVG), the second operand is eight bytes.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of LRVH, LRV, LRVG only)

Programming Notes:

- The instruction can be used to convert two, four, or eight bytes from a "little-endian" format to a "big-endian" format, or vice versa. In the big-endian format, the bytes in a left-to-right

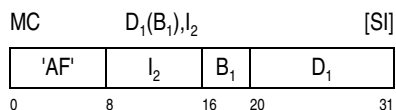
sequence are in the order most significant to least significant. In the little-endian format, the bytes are in the order least significant to most significant. For example, the bytes ABCD in the big-endian format are DCBA in the little-endian format.

- LOAD REVERSED (LRVR) can be used with a two-byte value already in a register as shown in the following example. In the example, the two bytes of interest are in bit positions 48-63 of the $R1$ register.

```
LRVR  R1,R1
SRA   R1,16
```

The LOAD REVERSED instruction places the two bytes of interest in bit positions 32-47 of the register, with the order of the bytes reversed. The SHIFT RIGHT SINGLE (SRA) instruction shifts the two bytes to bit positions 48-63 of the register and extends them on their left, in bit positions 32-47, with their sign bit. The instruction SHIFT RIGHT SINGLE LOGICAL (SRL) should be used, instead, if the two bytes of interest are unsigned.

- The storage-operand references of LOAD REVERSED may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)

MONITOR CALL

Subject to the control of the monitor class and the monitor masks, a monitor event may be recognized. When the enhanced-monitor facility is not installed, a monitor event results in a monitor-event program interruption. When the enhanced-monitor facility is installed, a monitor event results in either a monitor-event program interruption or a monitor-event counting operation.

Bit positions 12-15 of the instruction (bit positions 4-7 of the I_2 field) contain a 4-bit unsigned binary number specifying the monitor class. The monitor-mask bits are in bit positions 48-63 of control register 8, which correspond to monitor classes 0-15, respectively.

When the monitor-mask bit corresponding to the specified monitor class is one, a monitor event occurs. When the monitor-mask bit corresponding to the specified monitor class is zero, no monitor-event occurs, and the instruction is executed as a no-operation.

When the enhanced-monitor facility is installed, the enhanced-monitor-mask bits are in bit positions 16-31 of control register 8, which correspond to monitor classes 0-15 respectively. When a monitor event occurs and either (a) the enhanced-monitor facility is not installed, or (b) the facility is installed but the enhanced-monitor-mask bit corresponding to the monitor class is zero, then a monitor-event program interruption occurs, as described below. When a monitor event occurs and the enhanced-monitor-mask bit corresponding to the monitor class is one, a monitor-event counting operation is performed.

The first-operand address is not used to address data; instead, the address specified by the B₁ and D₁ fields forms the monitor code. Address computation follows the rules of address arithmetic; in the 24-bit addressing mode, bits 0-39 are set to zeros; in the 31-bit addressing mode, bits 0-32 are set to zeros.

Monitor-Event Program Interruption

When a monitor-event program interruption occurs:

- The monitor code formed by the first-operand address is placed in the doubleword at real location 176.
- The contents of the I₂ field are stored at real location 149, with zeros stored at real location 148.
- Bit 9 of the program-interruption code is set to one.

Monitor-Event Counting Operation

The monitor-event counting operation is described in "Monitor-Event Counting" on page 5-102.

Special Conditions

Bit positions 8-11 of the instruction must contain zeros; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Monitor event
- Specification

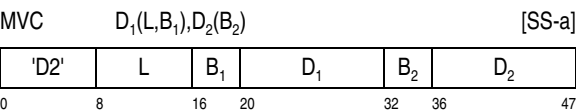
Programming Notes:

1. The monitor-event program interruption provides the capability for passing control to a monitoring program when selected points are reached in the monitored program. This is accomplished by implanting MONITOR CALL instructions at the desired points in the monitored program. This function may be useful in performing various measurement functions; specifically, tracing information can be generated indicating which programs were executed, counting information can be generated indicating how often particular programs were used, and timing information can be generated indicating the amount of time a particular program required for execution.
2. The monitor masks provide a means of disallowing all monitor events or allowing monitor events for all or selected classes.
3. When used to generate a monitor-event program interruption, the monitor code provides a means of associating descriptive information, in addition to the class number, with each MONITOR CALL.

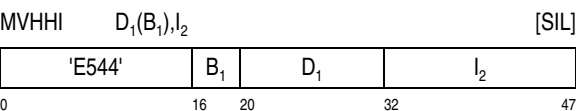
Without the use of a base register, up to 4,096 distinct monitor codes can be associated with a monitor event. With the base register designated by a nonzero value in the B₁ field, each monitoring interruption can be identified by a 24-bit, 31-bit, or 64-bit code, depending on the addressing mode.

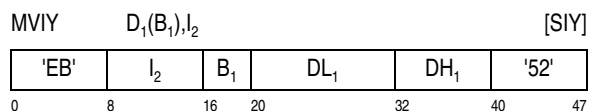
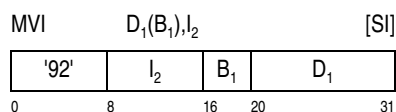
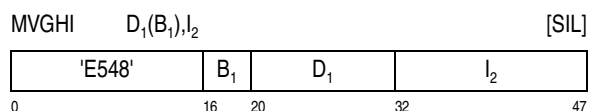
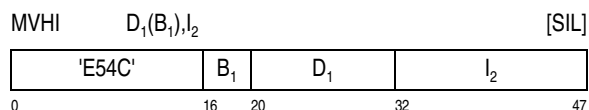
MOVE

Storage-and-storage format:



Storage-and-immediate formats:





The second operand is placed at the first-operand location.

For MOVE (MVC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand byte.

For MOVE (MVI, MVIY), the first operand is one byte in length, and only one byte is stored.

For MOVE (MVGHI, MVHHI, and MVHI), the second operand is treated as a 16-bit signed integer, sign-extended as necessary, and placed in the first-operand location. The first operand is two, four, or eight bytes for MVHHI, MVHI, and MVGHI, respectively.

The displacements for MVGHI, MVHHI, MVHI, MVI and both operands of MVC are treated as 12-bit unsigned binary integers. The displacement for MVIY is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

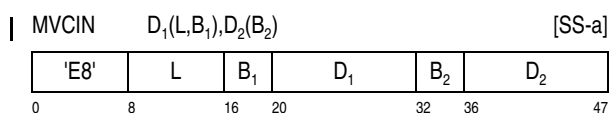
- Access (fetch, operand 2 of MVC; store, operand 1, MVC, MVGHI, MVHHI, MVHI, MVI, and MVIY)
- Operation (MVIY, if the long-displacement facility is not installed; MVGHI, MVHHI, and MVHI, if the

general-instructions-extension facility is not installed)

Programming Notes:

- Examples of the use of the MOVE instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
- For MOVE (MVC), it is possible to propagate one byte through an entire field by having the first operand start one byte to the right of the second operand.

MOVE INVERSE



The second operand is placed at the first-operand location with the left-to-right sequence of the bytes inverted.

The first-operand address designates the leftmost byte of the first operand. The second-operand address designates the rightmost byte of the second operand. Both operands have the same length.

The result is obtained as if the second operand were processed from right to left and the first operand from left to right. The second operand may wrap around from location $2^{24} - 1$ in the 24-bit addressing mode, to location $2^{31} - 1$ in the 31-bit addressing mode, or to location $2^{64} - 1$ in the 64-bit addressing mode. The first operand may wrap around from location $2^{24} - 1$ to location 0 in the 24-bit addressing mode, from location $2^{31} - 1$ to location 0 in the 31-bit addressing mode, or from location $2^{64} - 1$ to location 0 in the 64-bit addressing mode.

When the operands overlap by more than one byte, the contents of the overlapped portion of the result field are unpredictable.

Condition Code: The code remains unchanged.

Program Exceptions:

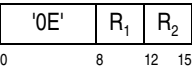
- Access (fetch, operand 2; store, operand 1)

Programming Notes:

- 1. An example of the use of the MOVE INVERSE instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
- 2. The contents of each byte moved remain unchanged.
- 3. MOVE INVERSE is the only SS-format instruction for which the second-operand address designates the rightmost, instead of the leftmost, byte of the second operand.
- 4. The storage-operand references for MOVE INVERSE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95)

MOVE LONG

MVCL R₁,R₂ [RR]



The second operand is placed at the first-operand location, provided overlapping of operand locations would not affect the final contents of the first-operand location. The remaining rightmost byte positions, if any, of the first-operand location are filled with padding bytes.

The R₁ and R₂ fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers R₁ and R₂, respectively. The number of bytes in the first-operand and second-operand locations is specified by unsigned binary integers in bit positions 40-63 of general registers R₁ + 1 and R₂ + 1, respectively. Bit positions 32-39 of general register R₂ + 1 contain the padding byte. The contents of bit positions 0-39 of general register R₁ + 1 and of bit positions 0-31 of general register R₂ + 1 are ignored.

The handling of the addresses in general registers R₁ and R₂ is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R₁ and R₂ constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The contents of the registers just described are shown in Figure 7-252 on page 7-235.

The result is obtained as if the movement starts at the left end of both fields and proceeds to the right, byte by byte. The operation is ended when the number of bytes specified by bits 40-63 of general register R₁ + 1 have been moved into the first-operand location. If the second operand is shorter than the first operand, the remaining rightmost bytes of the first-operand location are filled with the padding byte.

As part of the execution of the instruction, the values of the two length fields are compared for the setting of the condition code, and a check is made for destructive overlap of the operands. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it, assuming the inspection for overlap is performed by the use of logical operand addresses. When the operands overlap destructively, no movement takes place, and condition code 3 is set.

Operands do not overlap destructively, and movement is performed, if the leftmost byte of the first operand does not coincide with any of the second-operand bytes participating in the operation other than the leftmost byte of the second operand. When an operand wraps around from location 2²⁴ - 1 (or 2³¹ - 1 or 2⁶⁴ - 1) to location 0, operand bytes in locations up to and including 2²⁴ - 1 (or 2³¹ - 1 or 2⁶⁴ - 1) are considered to be to the left of bytes in locations from 0 up.

In the 24-bit addressing mode, wraparound is from location 2²⁴ - 1 to location 0; in the 31-bit addressing mode, wraparound is from location 2³¹ - 1 to location 0; in the 64-bit addressing mode, wraparound is from location 2⁶⁴ - 1 to location 0.

In the access-register mode, the contents of access register R₁ and access register R₂ are compared. If the R₁ or R₂ field is zero, 32 zeros are used rather than the contents of access register 0. If all 32 bits of the compared values are equal, then the destructive

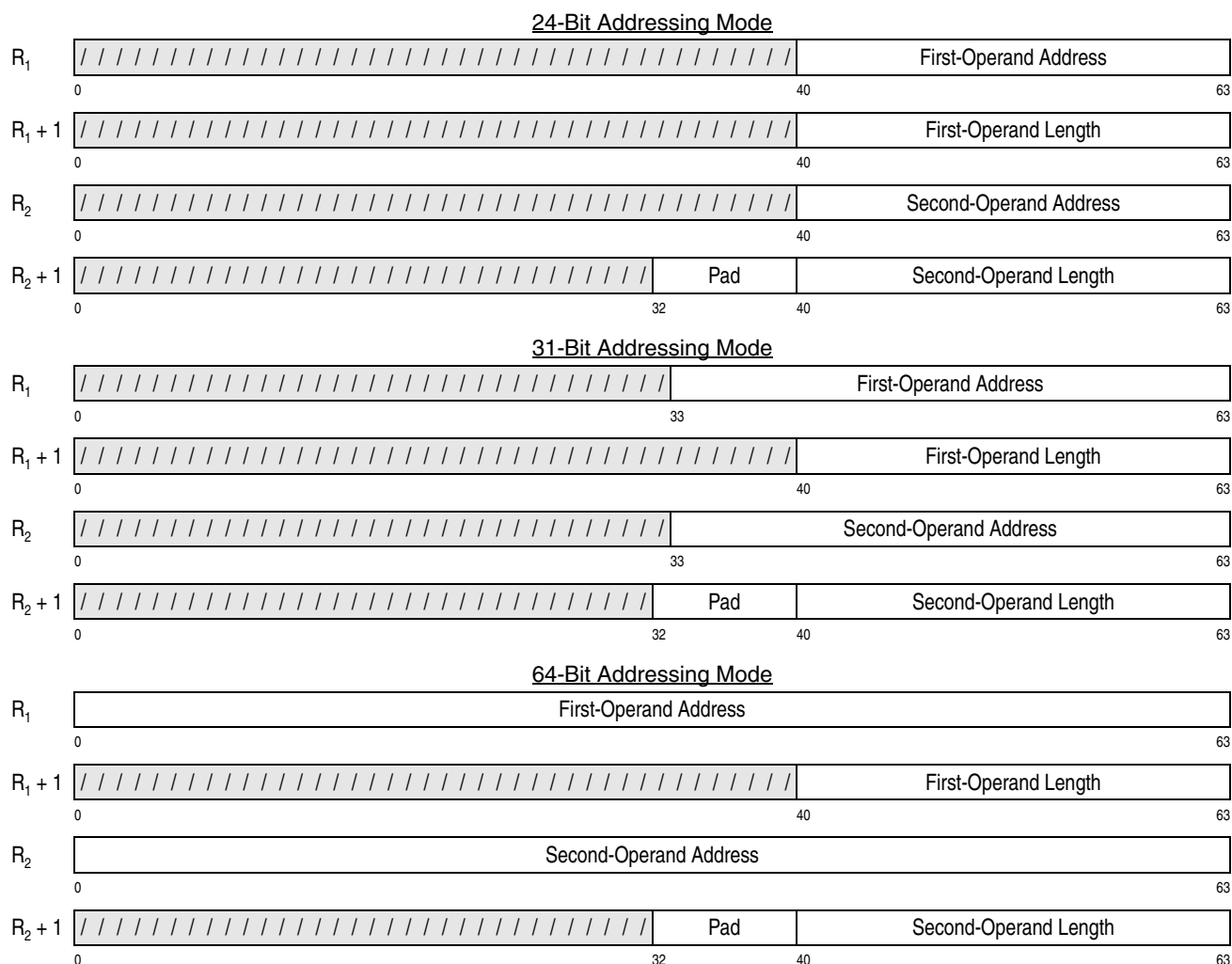


Figure 7-252. Register Contents for MOVE LONG

overlap test is made. If all 32 bits of the compared values are not equal, destructive overlap is declared not to exist. If, for this case, the operands actually overlap in real storage, it is unpredictable whether the result reflects the overlap condition.

When the length specified by bits 40-63 of general register $R_1 + 1$ is zero, no movement takes place, and condition code 0 or 1 is set to indicate the relative values of the lengths.

The execution of the instruction is interruptible. When an interruption occurs, other than one that follows termination, the lengths in general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes moved, and the addresses in general registers R_1 and R_2 are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. In the 24-bit or 31-bit addressing

mode, the leftmost bits which are not part of the address in bit positions 32-63 of general registers R_1 and R_2 are set to zeros, and the contents of bit positions 0-31 remain unchanged. In any addressing mode, the contents of bit positions 0-39 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged; and the condition code is unpredictable. If the operation is interrupted during padding, the length field in general register $R_2 + 1$ is 0, the address in general register R_2 is incremented by the original length in general register $R_2 + 1$, and general registers R_1 and $R_1 + 1$ reflect the extent of the padding operation.

When the first-operand location includes the location of the instruction or of an execute-type instruction, the instruction may be refetched from storage and reinterpreted even in the absence of an interruption during execution. The exact point in the execution at which such a refetch occurs is unpredictable.

Padding byte values of B0 hex and B8 hex may be used during the nonpadding part of the operation by some models, in certain cases, as an indication of whether the movement should be performed bypassing the cache or using the cache, respectively. Thus, a padding byte of B0 hex indicates no intention to reference the destination area after the move, and a padding byte of B8 hex indicates an intention to reference the destination area.

For the nonpadding part of the operation when the padding byte is not B1 hex, accesses to the operands for MOVE LONG may be multiple-access references, and these accesses do not necessarily appear to occur in a left-to-right direction as observed by other CPUs and by the channel subsystem. When the padding byte is B1 hex, accesses to the operands are single-access references, and these accesses appear to occur in a left-to-right direction as observed by other CPUs and by the channel subsystem. During the nonpadding part of the operation, operands appear to be accessed doubleword concurrent as observed by other CPUs, provided that both operands start on doubleword boundaries, are an integral number of doublewords in length, and do not overlap.

As observed by other CPUs and by the channel subsystem, that portion of the first operand which is filled with the padding byte is not necessarily stored into in a left-to-right direction and may appear to be stored into more than once. During the padding operation, stores by other CPUs or by the channel subsystem into that portion of the first operand which is filled with the padding byte may cause unpredictable results.

At the completion of the operation, the length in general register $R_1 + 1$ is decremented by the number of bytes stored at the first-operand location, and the address in general register R_1 is incremented by the same amount. The length in general register $R_2 + 1$ is decremented by the number of bytes moved out of the second-operand location, and the address in general register R_2 is incremented by the same amount. In the 24-bit or 31-bit addressing mode, the leftmost bits which are not part of the address in bit positions 32-63 of general registers R_1 and R_2 are set to zeros, even when one or both of the original length values are zeros or when condition code 3 is set. The contents of bit positions 0-31 of the registers remain unchanged. In any addressing mode, the contents of bit positions 0-39 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged.

When condition code 3 is set, no exceptions associated with operand access are recognized. When the length of an operand is zero, no access exceptions for that operand are recognized. Similarly, when the second operand is longer than the first operand, access exceptions are not recognized for the part of the second-operand field that is in excess of the first-operand field. For operands longer than 2K bytes, access exceptions are not recognized for locations more than 2K bytes beyond the current location being processed. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Also, when the R_1 field is odd, PER storage-alteration events are not recognized, and no change bits are set.

Resulting Condition Code:

- 0 Operand lengths equal; no destructive overlap
- 1 First-operand length low; no destructive overlap
- 2 First-operand length high; no destructive overlap
- 3 No movement performed because of destructive overlap

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Specification

Programming Notes:

1. An example of the use of the MOVE LONG instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. MOVE LONG may be used for clearing storage by setting the padding byte to zero and the second-operand length to zero. On most models, this is the fastest instruction for clearing storage areas in excess of 256 bytes. However, the stores associated with this clearing may be multiple-access stores and should not be used to clear an area if the possibility exists that another CPU or a channel program will attempt to access and use the area as soon as it appears to be zero. For more details, see "Storage-Operand Consistency" on page 5-95.
3. The program should avoid specification of a length for either operand which would result in an addressing exception. Addressing (and also protection) exceptions may result in termination of the entire operation, not just the current unit of operation. The termination may be such that the

contents of all result fields are unpredictable; in the case of MOVE LONG, this includes the condition code and the two even-odd general-register pairs, as well as the first-operand location in main storage. The following are situations that have actually occurred on one or more models:

- a. When a protection exception occurs on a 4K-byte block of a first operand which is several blocks in length, stores to the protected block are suppressed. However, the move continues into the subsequent blocks of the first operand, which are not protected. Similarly, an addressing exception on a block does not necessarily suppress processing of subsequent blocks which are available.
 - b. Some models may update the general registers only when an external, I/O, repressible machine-check, or restart interruption occurs, or when a program interruption occurs for which it is required to nullify or suppress a unit of operation. Thus, if, after a move into several blocks of the first operand, an addressing or protection exception occurs, the general registers may remain unchanged.
4. When the first-operand length is zero, the operation consists in setting the condition code and, in the 24-bit or 31-bit addressing mode, of setting the leftmost bits in bit positions 32-63 of general registers R_1 and R_2 to zero.
 5. When the contents of the R_1 and R_2 fields are the same, the contents of the designated registers are incremented or decremented only by the number of bytes moved, not by twice the number of bytes moved. Condition code 0 is set.
 6. The following is a detailed description of those cases in which movement takes place, that is, where destructive overlap does not exist.

In the access-register mode, the contents of the access registers used are called the effective space designations. When the effective space designations are not equal, destructive overlap is declared not to exist and movement occurs. When the effective space designations are the same or when not in the access-register mode, then the following cases apply.

Depending on whether the second operand wraps around from location $2^{24} - 1$ (or $2^{31} - 1$ or $2^{64} - 1$, depending on the addressing mode) to location 0, movement takes place in the following cases:

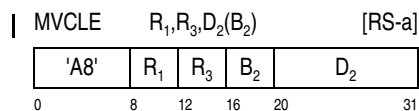
- a. When the second operand does not wrap around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, *or* if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.
- b. When the second operand wraps around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, *and* if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.

The rightmost second-operand byte is determined by using the smaller of the first-operand and second-operand lengths.

When the second-operand length is one or zero, destructive overlap cannot exist.

7. Special precautions should be taken if MOVE LONG is made the target of an execute-type instruction. See the programming note concerning interruptible instructions under EXECUTE.
8. Since the execution of MOVE LONG is interruptible, the instruction cannot be used for situations where the program must rely on uninterrupted execution of the instruction. Similarly, the program should normally not let the first operand of MOVE LONG include the location of the instruction or of an execute-type instruction because the new contents of the location may be interpreted for a resumption after an interruption, or the instruction may be refetched without an interruption.
9. Further programming notes concerning interruptible instructions are included in "Interruptible Instructions" in Chapter 5, "Program Execution."
10. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

MOVE LONG EXTENDED



All or part of the third operand is placed at the first-operand location. The remaining rightmost byte positions, if any, of the first-operand location are filled with padding bytes. The operation proceeds until the end of the first-operand location is reached or a CPU-determined number of bytes have been placed at the first-operand location, whichever occurs first. The result is indicated in the condition code.

The R_1 and R_3 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and third operand is designated by the contents of general registers R_1 and R_3 , respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 32-63 of general registers $R_1 + 1$ and $R_3 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and third-operand locations is specified by the entire contents of general registers $R_1 + 1$ and $R_3 + 1$, respectively, and those contents are treated as 64-bit unsigned binary integers.

The handling of the addresses in general registers R_1 and R_3 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_3 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The second-operand address is not used to address data; instead, the rightmost eight bits of the second-operand address, bits 56-63, are the padding byte. Bits 0-55 of the second-operand address are ignored.

The contents of the registers and address just described are shown in Figure 7-253 on page 7-239.

The result is obtained as if the movement starts at the left end of both fields and proceeds to the right, byte by byte. The operation is ended when the number of bytes specified in general register $R_1 + 1$ have been placed at the first-operand location or when a CPU-determined number of bytes have been placed, whichever occurs first. If the third operand is shorter than the first operand, the remaining rightmost bytes of the first-operand location are filled with the padding byte.

When the operation is completed because the end of the first operand has been reached, the condition code is set to 0 if the two operand lengths are equal, it is set to 1 if the first-operand length is less than the third-operand length, or it is set to 2 if the first-operand length is greater than the third-operand length. When the operation is completed because a CPU-determined number of bytes have been moved without reaching the end of the first operand, condition code 3 is set.

No test is made for destructive overlap, and the results in the first-operand location are unpredictable when destructive overlap exists. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it.

Operands do not overlap destructively if the leftmost byte of the first operand does not coincide with any of the third-operand bytes participating in the operation other than the leftmost byte of the third operand. When an operand wraps around from location $2^{24} - 1$ (or $2^{31} - 1$ or $2^{64} - 1$) to location 0, operand bytes in locations up to and including $2^{24} - 1$ (or $2^{31} - 1$ or $2^{64} - 1$) are considered to be to the left of bytes in locations from 0 up.

In the 24-bit addressing mode, wraparound is from location $2^{24} - 1$ to location 0; in the 31-bit addressing mode, wraparound is from location $2^{31} - 1$ to location 0; and, in the 64-bit addressing mode, wraparound is from location $2^{64} - 1$ to location 0.

When the length specified in general register $R_1 + 1$ is zero, no movement takes place, and condition code 0 or 1 is set to indicate the relative values of the lengths.

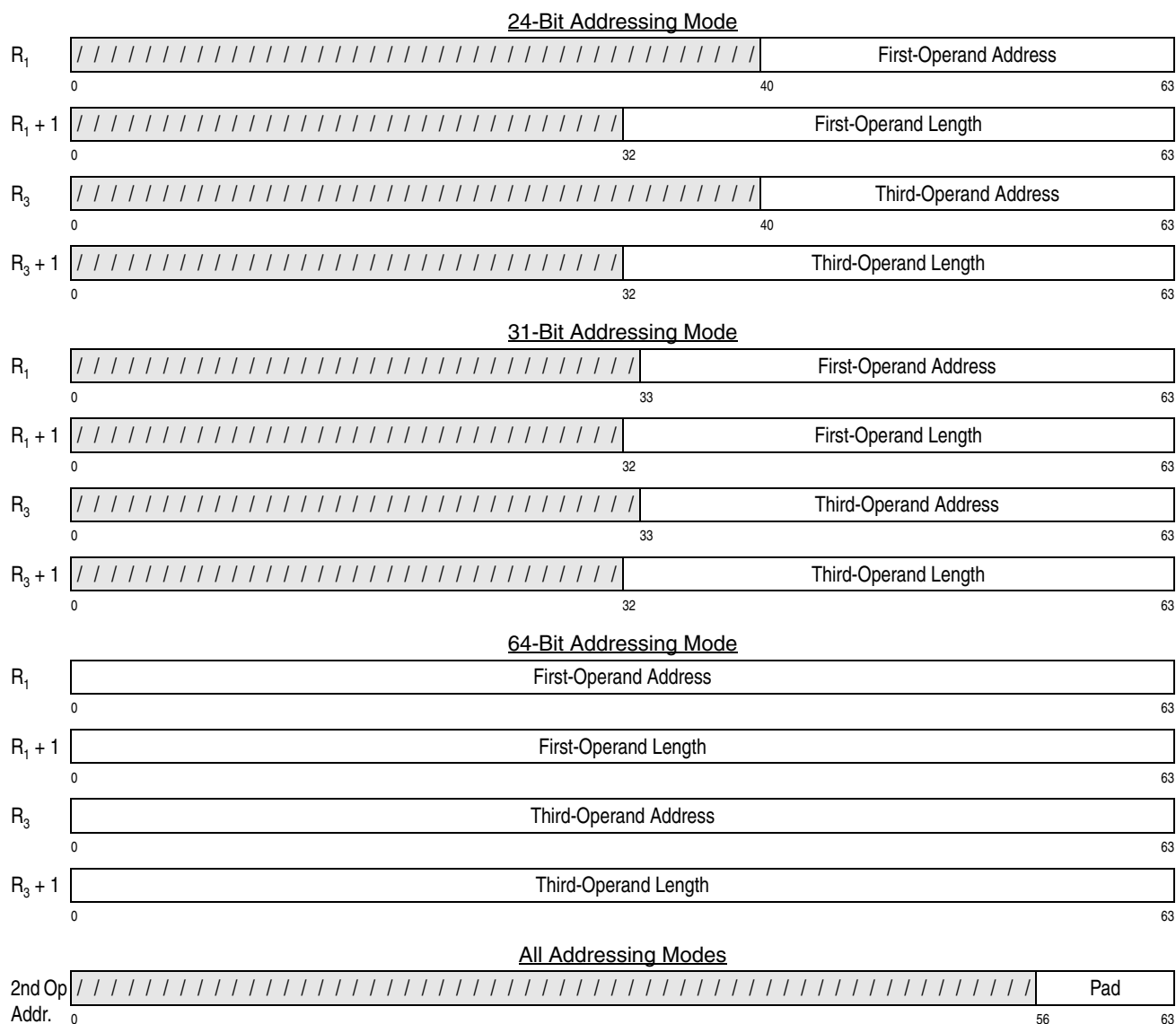


Figure 7-253. Register Contents and Second-Operand Address for MOVE LONG EXTENDED

Padding byte values of B0 hex and B8 hex may be used during the nonpadding part of the operation by some models, in certain cases, as an indication of whether the movement should be performed bypassing the cache or using the cache, respectively. Thus, a padding byte of B0 hex indicates no intention to reference the destination area after the move, and a padding byte of B8 hex indicates an intention to reference the destination area.

For the nonpadding part of the operation when the padding byte is not B1 hex, accesses to the operands for MOVE LONG EXTENDED may be multiple-access references, and these accesses do not necessarily appear to occur in a left-to-right direction as

observed by other CPUs and by the channel subsystem. When the padding byte is B1 hex, accesses to the operands are single-access references, and these accesses appear to occur in a left-to-right direction as observed by other CPUs and by the channel subsystem. During the nonpadding part of the operation, operands appear to be accessed doubleword concurrent as observed by other CPUs, provided that both operands start on doubleword boundaries, are an integral number of doublewords in length, and do not overlap.

As observed by other CPUs and by the channel subsystem, that portion of the first operand which is filled with the padding byte is not necessarily stored into in

a left-to-right direction and may appear to be stored into more than once. During the padding operation, stores by other CPUs or by the channel subsystem into that portion of the first operand which is filled with the padding byte may cause unpredictable results.

At the completion of the operation, the length in general register $R_1 + 1$ is decremented by the number of bytes stored at the first-operand location, and the address in general register R_1 is incremented by the same amount. The length in general register $R_3 + 1$ is decremented by the number of bytes moved out of the third-operand location, and the address in general register R_3 is incremented by the same amount.

If the operation is completed because a CPU-determined number of bytes have been moved without reaching the end of the first operand, the lengths in general registers $R_1 + 1$ and $R_3 + 1$ are decremented by the number of bytes moved, and the addresses in general registers R_1 and R_3 are incremented by the same number, so that the instruction, when reexecuted, resumes at the next byte to be moved. If the operation is completed during padding, the length field in general register $R_3 + 1$ is zero, the address in general register R_3 is incremented by the original length in general register $R_3 + 1$, and general registers R_1 and $R_1 + 1$ reflect the extent of the padding operation.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, R_3 , and $R_3 + 1$, always remain unchanged.

The padding byte may be formed from $D_2(B_2)$ multiple times during the execution of the instruction, and the registers designated by R_1 and R_3 may be updated multiple times. Therefore, if B_2 equals R_1 , $R_1 + 1$, R_3 , or $R_3 + 1$ and is subject to change during the execution of the instruction, the results are unpredictable.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed. The maximum amount is approximately 4K bytes of either operand.

At the completion of the operation in the 24-bit or 31-bit addressing mode, the leftmost bits which are not part of the address in bit positions 32-63 of general registers R_1 and R_3 may be set to zeros or may

remain unchanged from their original values, even when one or both of the original length values are zeros.

When the length of an operand is zero, no access exceptions for that operand are recognized. Similarly, when the third operand is longer than the first operand, access exceptions are not recognized for the part of the third-operand field that is in excess of the first-operand field. For operands longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the current location being processed. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Also, when the R_1 field is odd, PER storage-alteration events are not recognized, and no change bits are set.

Resulting Condition Code:

- 0 All bytes moved, operand lengths equal
- 1 All bytes moved, first-operand length low
- 2 All bytes moved, first-operand length high
- 3 CPU-determined number of bytes moved without reaching end of first operand

Program Exceptions:

- Access (fetch, operand 3; store, operand 1)
- Specification

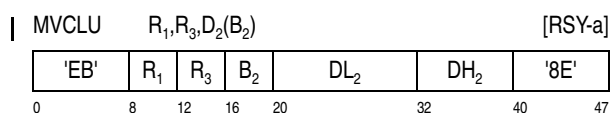
Programming Notes:

1. MOVE LONG EXTENDED is intended for use in place of MOVE LONG when the operand lengths are specified as 32-bit or 64-bit binary integers and a test for destructive overlap is not required. MOVE LONG EXTENDED sets condition code 3 in cases in which MOVE LONG would be interrupted.
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the movement. The program need not determine the number of bytes that were moved.
3. The function of not processing more than approximately 4K bytes of either operand is intended to permit software polling of a flag that may be set by a program on another CPU during long operations.
4. MOVE LONG EXTENDED may be used for clearing storage by setting the padding byte to zero and the third-operand length to zero. How-

ever, the stores associated with this clearing may be multiple-access stores and should not be used to clear an area if the possibility exists that another CPU or a channel program will attempt to access and use the area as soon as it appears to be zero. For more details, see “Storage-Operand Consistency” on page 5-95.

5. When the contents of the R_1 and R_3 fields are the same, the contents of the designated registers are incremented or decremented only by the number of bytes moved, not by twice the number of bytes moved. The condition code is finally set to 0 after possible settings to 3.
6. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

MOVE LONG UNICODE



All or part of the third operand is placed at the first-operand location. The remaining rightmost positions, if any, of the first-operand location are filled with one or more padding bytes, alternating between the even padding byte and the odd padding byte. The operation proceeds until the end of the first-operand location is reached or a CPU-determined number of bytes have been placed at the first-operand location, whichever occurs first. The result is indicated in the condition code.

The R_1 and R_3 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and third operand is designated by the contents of general registers R_1 and R_3 , respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 32-63 of general registers $R_1 + 1$ and $R_3 + 1$, respectively, and those contents are treated as 32-bit unsigned binary integers. In the 64-bit addressing mode, the number of bytes in the first-operand and third-operand locations is specified by the contents of bit positions 0-63 of general registers $R_1 + 1$ and $R_3 + 1$, respectively,

and those contents are treated as 64-bit unsigned binary integers.

On some models, the contents of general registers $R_1 + 1$ and $R_3 + 1$ must specify an even number of bytes; otherwise, a specification exception is recognized.

The handling of the addresses in general registers R_1 and R_3 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_3 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of the registers constitute the address.

The second-operand address is not used to address data; instead, the rightmost 16 bits of the second-operand address, bits 48-63, are the two padding bytes; bits 48-55 are the even padding byte, and bits 56-63 are the odd padding byte. Bits 0-47 of the second-operand address are ignored.

The contents of the registers and address just described are shown in Figure 7-254 on page 7-242.

The result is obtained as if the movement starts at the left end of both fields and proceeds to the right, byte by byte. The operation is ended when the number of bytes specified by the contents of general register $R_1 + 1$ have been placed at the first-operand location or when a CPU-determined number of bytes have been placed, whichever occurs first.

If the third operand is shorter than the first operand, the remaining rightmost positions of the first-operand location are filled with alternating padding bytes. When the number of bytes remaining in the first-operand location is an even number, the even padding byte is placed first; when the number of bytes remaining in the first-operand location is odd, the odd padding byte is placed first.

When the operation is completed because the end of the first operand has been reached, the condition code is set to 0 if the two operand lengths are equal, it is set to 1 if the first-operand length is less than the third-operand length, or it is set to 2 if the first-operand length is greater than the third-operand length.

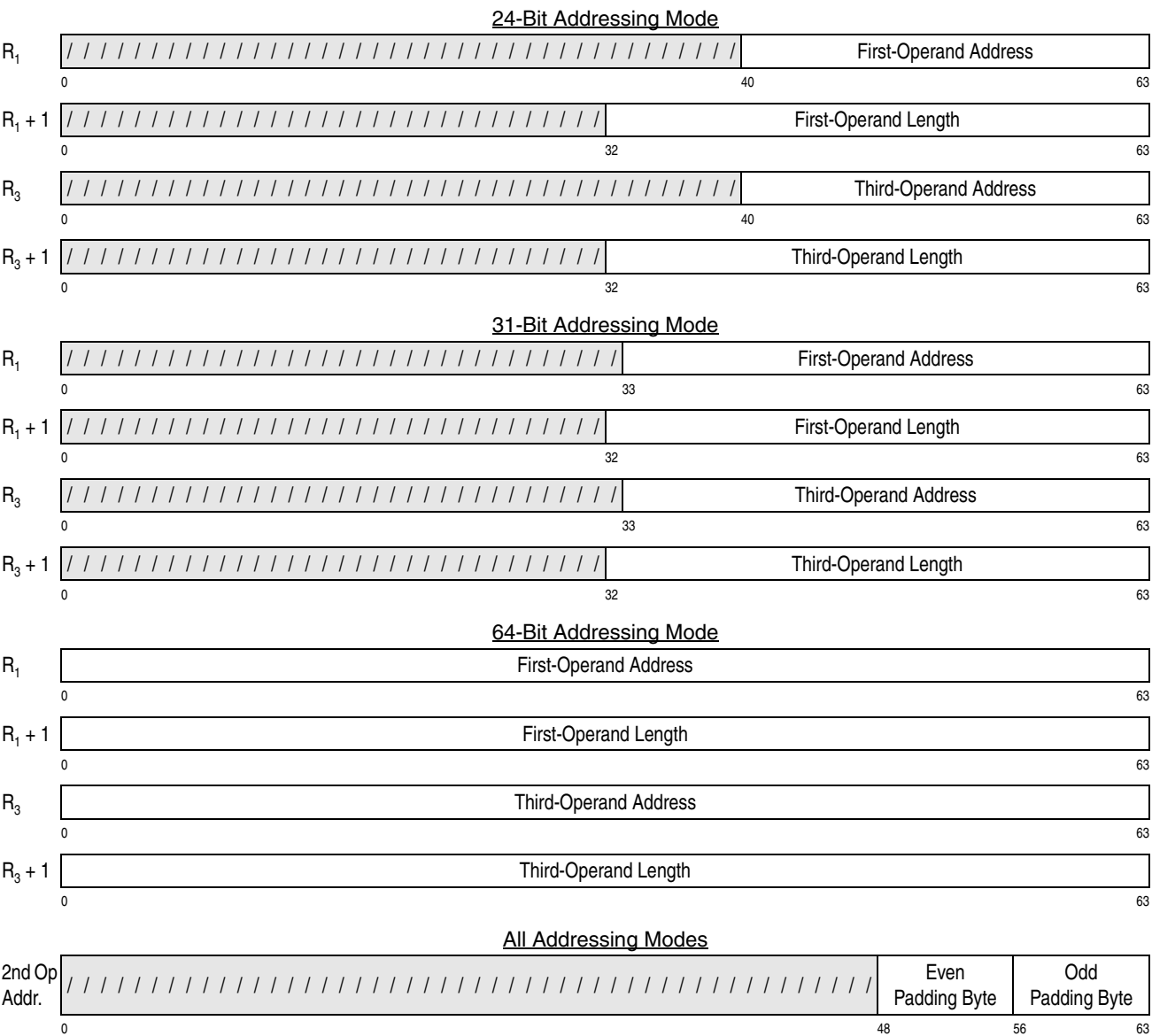


Figure 7-254. Register Contents and Second-Operand Address for MOVE LONG UNICODE

When the operation is completed because a CPU-determined number of bytes have been moved without reaching the end of the first operand, condition code 3 is set.

No test is made for destructive overlap, and the results in the first-operand location are unpredictable when destructive overlap exists. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it.

Operands do not overlap destructively if the leftmost byte of the first operand does not coincide with any of

the third-operand bytes participating in the operation other than the leftmost byte of the third operand. When an operand wraps around from location $2^{24} - 1$ (or $2^{31} - 1$ or $2^{64} - 1$) to location 0, operand bytes in locations up to and including $2^{24} - 1$ (or $2^{31} - 1$ or $2^{64} - 1$) are considered to be to the left of bytes in locations from 0 up.

In the 24-bit addressing mode, wraparound is from location $2^{24} - 1$ to location 0; in the 31-bit addressing mode, wraparound is from location $2^{31} - 1$ to location 0; and, in the 64-bit addressing mode, wraparound is from location $2^{64} - 1$ to location 0.

When the length specified in general register $R_1 + 1$ is zero, no movement takes place, and condition code 0 or 1 is set to indicate the relative values of the lengths.

For the nonpadding part of the operation, accesses to the operands for MOVE LONG UNICODE may be multiple-access references. These accesses do not necessarily appear to occur in a left-to-right direction as observed by other CPUs and by channel programs. During the nonpadding part of the operation, operands appear to be accessed doubleword concurrent as observed by other CPUs, provided that both operands start on doubleword boundaries, are an integral number of doublewords in length, and do not overlap.

As observed by other CPUs and by the channel subsystem, that portion of the first operand which is filled with the padding bytes is not necessarily stored into in a left-to-right direction and may appear to be stored into more than once. During the padding operation, stores by other CPUs or by the channel subsystem into that portion of the first operand which is filled with the padding bytes may cause unpredictable results.

At the completion of the operation, the length in general register $R_1 + 1$ is decremented by the number of bytes stored at the first-operand location, and the address in general register R_1 is incremented by the same amount. The length in general register $R_3 + 1$ is decremented by the number of bytes moved out of the third-operand location, and the address in general register R_3 is incremented by the same amount.

If the operation is completed because a CPU-determined number of bytes have been moved without reaching the end of the first operand, the lengths in general registers $R_1 + 1$ and $R_3 + 1$ are decremented by the number of bytes moved, and the addresses in general registers R_1 and R_3 are incremented by the same number, so that the instruction, when reexecuted, resumes at the next byte to be moved. If the operation is completed during padding, the length field in general register $R_3 + 1$ is zero, the address in general register R_3 is incremented by the number of bytes moved from operand 3, and general registers R_1 and $R_1 + 1$ reflect the extent of the padding operation.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, R_2 , and $R_2 + 1$, always remain unchanged.

The padding bytes may be formed from $D_2(B_2)$ multiple times during the execution of the instruction, and the registers designated by R_1 and R_3 may be updated multiple times. Therefore, if B_2 equals R_1 , $R_1 + 1$, R_3 , or $R_3 + 1$ and is subject to change during the execution of the instruction, the results are unpredictable.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

At the completion of the operation in the 24-bit or 31-bit addressing mode, the leftmost bits which are not part of the address in bit positions 32-63 of general registers R_1 and R_3 may be set to zeros or may remain unchanged from their original values, including the case when one or both of the original length values are zeros.

When the length of an operand is zero, no access exceptions for that operand are recognized. Similarly, when the third operand is longer than the first operand, access exceptions are not recognized for the part of the third-operand field that is in excess of the first-operand field. For operands longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the current location being processed. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Also, when the R_1 field is odd, PER storage-alteration events are not recognized, and no change bits are set. On models that recognize a specification exception if the length associated with an operand is odd, access exceptions and PER storage-alteration events are not recognized and no change bits are set if the length is odd.

Resulting Condition Code:

- 0 All bytes moved, operand lengths equal
- 1 All bytes moved, first-operand length low
- 2 All bytes moved, first-operand length high
- 3 CPU-determined number of bytes moved without reaching end of first operand

Program Exceptions:

- Access (fetch, operand 3; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

Programming Notes:

1. **MOVE LONG UNICODE** is intended for use in place of **MOVE LONG** or **MOVE LONG EXTENDED** when padding with double-byte characters. The character may be a Unicode character or any other double-byte character. **MOVE LONG UNICODE** may set condition code 3 in cases in which **MOVE LONG** would be interrupted.
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the movement. The program need not determine the number of bytes that were moved.
3. **MOVE LONG UNICODE** may be used for filling storage with padding bytes by placing the padding bytes in the second-operand address and setting the third-operand length to zero. However, the stores associated with this clearing may be multiple-access stores and should not be used to clear an area if the possibility exists that another CPU or a channel program will attempt to access and use the area as soon as it appears to be zero. For more details, see “Storage-Operand Consistency” on page 5-95.
4. When the contents of the R_1 and R_3 fields are the same, the contents of the designated registers are incremented or decremented only by the number of bytes moved, not by two times the number of bytes moved. The condition code is finally set to 0 after possible settings to 3.
5. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
6. The padding bytes may be represented in the displacement field of the instruction. The following example illustrates padding with a Unicode space character.

MVCLU 6,8,X'0020'

When the B₂ field of the instruction designates general register 0, and the long-displacement facility is not installed, the padding bytes are limited to a character whose representation is less than or equal to 0FFF hex.

7. **MOVE LONG UNICODE** is intended to be used to move and pad pairs of bytes. Although some models allow the first- and third-operand length registers to designate an odd length, without rec-

ognizing a specification exception, the program should always be coded to provide an even length. This ensures that the program will operate compatibly in configurations that recognize a specification exception for an odd length value.

MOVE NUMERICS

MVN $D_1(L, B_1), D_2(B_2)$ [SS-a]

'D1'	L	B_1	D_1	B_2	D_2
0	8	16	20	32	36
47					

The rightmost four bits of each byte in the second operand are placed in the rightmost bit positions of the corresponding bytes in the first operand. The leftmost four bits of each byte in the first operand remain unchanged.

Each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

Condition Code: The code remains unchanged.

Program Exceptions:

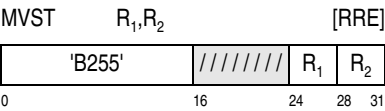
- Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes:

1. An example of the use of the MOVE NUMERICS instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. MOVE NUMERICS moves the numeric portion of a decimal-data field that is in the zoned format. The zoned-decimal format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.
3. Accesses to the first operand of MOVE NUMERICS consist in fetching the leftmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An

example of this effect is shown for OR (OI) in “Multiprogramming and Multiprocessing Examples” in Appendix A, “Number Representation and Instruction-Use Examples.”

MOVE STRING



All or part of the second operand is placed in the first-operand location. The operation proceeds until the end of the second operand is reached or a CPU-determined number of bytes have been moved, whichever occurs first. The CPU-determined number is at least one. The result is indicated in the condition code.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers R₁ and R₂, respectively.

The handling of the addresses in general registers R₁ and R₂ is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R₁ and R₂ constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The end of the second operand is indicated by an ending character in the last byte position of the operand. The ending character to be used to determine the end of the second operand is specified in bit positions 56-63 of general register 0. Bit positions 32-55 of general register 0 are reserved for possible future extensions and must contain all zeros; otherwise, a specification exception is recognized.

The operation proceeds left to right and ends as soon as the second-operand ending character has been moved or a CPU-determined number of second-operand bytes have been moved, whichever occurs first. The CPU-determined number is at least one. When the ending character is in the first byte position of the second operand, only the ending char-

acter is moved. When the ending character has been moved, condition code 1 is set. When a CPU-determined number of second-operand bytes not including an ending character have been moved, condition code 3 is set. Destructive overlap is not recognized. If the second operand is used as a source after it has been used as a destination, the results are unpredictable.

When condition code 1 is set, the address of the ending character in the first operand is placed in general register R₁, and the contents of general register R₂ remain unchanged. When condition code 3 is set, the address of the next byte to be processed in the first and second operands is placed in general registers R₁ and R₂, respectively. Whenever an address is placed in a general register, bits 32-39 of the register, in the 24-bit addressing mode, or bit 32, in the 31-bit addressing mode, are set to zeros. Bits 0-31 of the R₁ and R₂ registers always remain unchanged in the 24-bit or 31-bit mode.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

Access exceptions for the first and second operands are recognized only for that portion of the operand that is necessarily used in the operation.

The storage-operand-consistency rules are the same as for the MOVE (MVC) instruction, except that destructive overlap is not recognized.

Resulting Condition Code:

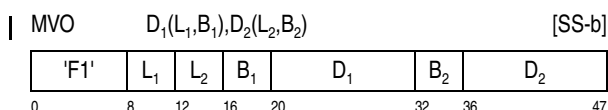
- 0 --
- 1 Entire second operand moved; general register R₁ updated with address of ending character in first operand; general register R₂ unchanged
- 2 --
- 3 CPU-determined number of bytes moved; general registers R₁ and R₂ updated with addresses of next bytes

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Specification

Programming Notes:

1. An example of the use of the MOVE STRING instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the data movement. The program need not determine the number of bytes that were moved.
3. R_1 or R_2 may be zero, in which case general register 0 is treated as containing an address and also the ending character.
4. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.
5. If the second operand is used as a source after it has been used as a destination, an ending character in the second operand may not be recognized.

MOVE WITH OFFSET

The second operand is placed to the left of and adjacent to the rightmost four bits of the first operand.

The rightmost four bits of the first operand are attached as the rightmost bits to the second operand, the second-operand bits are offset by four bit positions, and the result is placed at the first-operand location.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand is too short to contain all of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if the operands were processed one byte at a time, as if each result byte were stored immediately after fetching the necessary operand bytes, and as if the left digit of each second-operand byte were to remain

available for the next result byte and need not be refetched.

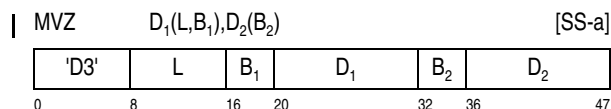
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes:

1. An example of the use of the MOVE WITH OFFSET instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. MOVE WITH OFFSET may be used to shift packed decimal data by an odd number of digit positions. The packed-decimal format is described in Chapter 8, “Decimal Instructions.” The operands are not checked for valid sign and digit codes. In many cases, however, SHIFT AND ROUND DECIMAL may be more convenient to use.
3. Access to the rightmost byte of the first operand of MOVE WITH OFFSET consists in fetching the rightmost four bits and subsequently storing the updated value of this byte. These fetch and store accesses to the rightmost byte of the first operand do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in “Multiprogramming and Multiprocessing Examples” in Appendix A, “Number Representation and Instruction-Use Examples.”
4. The storage-operand references for MOVE WITH OFFSET may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

MOVE ZONES

The leftmost four bits of each byte in the second operand are placed in the leftmost four bit positions

of the corresponding bytes in the first operand. The rightmost four bits of each byte in the first operand remain unchanged.

Each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

Condition Code: The code remains unchanged.

Program Exceptions:

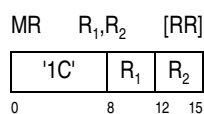
- Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes:

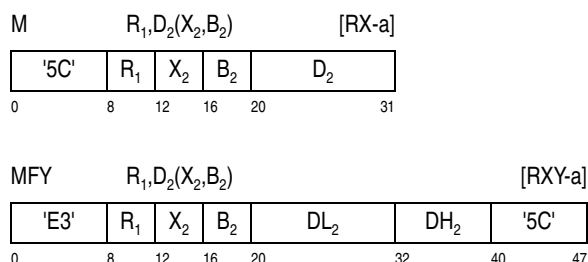
1. An example of the use of the MOVE ZONES instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. MOVE ZONES moves the zoned portion of a decimal field in the zoned format. The zoned format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.
3. Accesses to the first operand of MOVE ZONES consist in fetching the rightmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for the OR (OI) instruction in "Multiprogramming and Multiprocessing Examples" in Appendix A, "Number Representation and Instruction-Use Examples."

MULTIPLY

Register-and-register format:



Register-and-storage formats:



The 32-bit first operand (the multiplicand) is multiplied by the 32-bit second-operand (the multiplier), and the 64-bit product is placed at the first-operand location.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

Both the multiplicand and multiplier are treated as 32-bit signed binary integers. The multiplicand is in bit positions 32-63 of general register $R_1 + 1$. For MULTIPLY (MR), the multiplier is in bit positions 32-63 of general register R_2 . The contents of general register R_1 and of bit positions 0-31 of general register $R_1 + 1$ and, for MR, of general register R_2 are ignored.

The product is a 64-bit signed binary integer. Bits 0-31 of the product replace bits 32-63 of general register R_1 . Bits 32-63 of the product replace bits 32-63 of general register $R_1 + 1$. Bits 0-31 of general registers R_1 and $R_1 + 1$ remain unchanged. An overflow cannot occur.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

The displacement for M is treated as a 12-bit unsigned binary integer. The displacement for MFY is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

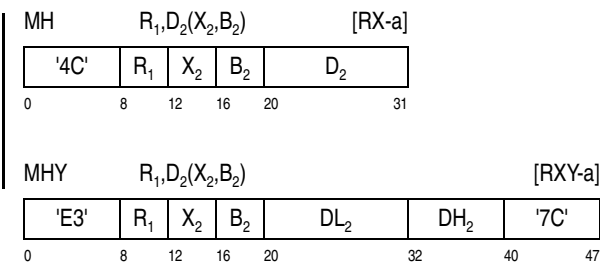
Program Exceptions:

- Access (fetch, operand 2 of M and MFY only)
- Operation (MFY; if the general-instructions-extension facility is not installed)
- Specification

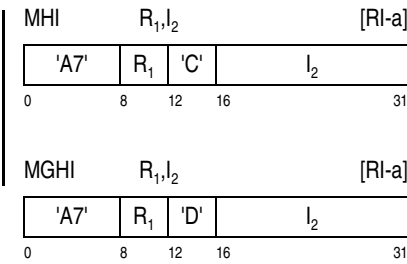
Programming Notes:

- 1. An example of the use of the MULTIPLY instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
- 2. The significant part of the product usually occupies 62 bit positions or fewer. Only when two maximum 32-bit negative numbers are multiplied are 63 significant product bits formed.
- 3. Care should be taken not to confuse the mnemonic for MULTIPLY UNNORMALIZED (MY) with the mnemonic for MULTIPLY (MFY).

MULTIPLY HALFWORD



MULTIPLY HALFWORD IMMEDIATE



The 32-bit or 64-bit first operand (the multiplicand) is multiplied by the 16-bit second operand (the multiplier), and the rightmost 32 or 64 bits of the product are placed at the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer.

For MULTIPLY HALFWORD and MULTIPLY HALFWORD IMMEDIATE (MHI), the multiplicand is treated as a 32-bit signed binary integer in bit positions 32-63 of general register R_1 , and it is replaced by the rightmost 32 bits of the signed-binary-integer prod-

uct. The bits to the left of the 32 rightmost bits of the product are not tested for significance; no overflow indication is given. Bits 0-31 of general register R_1 are ignored and remain unchanged.

For MULTIPLY HALFWORD IMMEDIATE (MGHI), the multiplicand is treated as a 64-bit signed binary integer in bit positions 0-63 of general register R_1 , and it is replaced by the rightmost 64 bits of the signed-binary-integer product. The bits to the left of the 64 rightmost bits of the product are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

The displacement for MH is treated as a 12-bit unsigned binary integer. The displacement for MHY is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

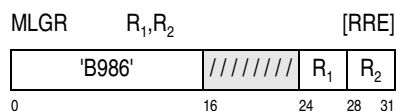
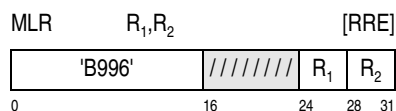
- Access (fetch, operand 2 of MH and MHY only)
- Operation (MHY; if the general-instructions-extension facility is not installed)

Programming Notes:

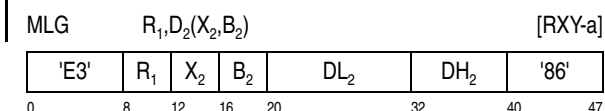
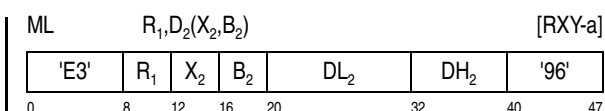
- 1. An example of the use of the MULTIPLY HALFWORD instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
- 2. For MH, MHY, and MHI, the significant part of the product usually occupies 46 bit positions or fewer. Only when two maximum negative numbers are multiplied are 47 significant product bits formed. Since the rightmost 32 bits of the product are placed unchanged at the first-operand location, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow. For a negative product, the 32 bits placed in register R_1 are the rightmost part of the product in two's-complement notation. For MGHI, the significant part of the product usually occupies 78 bit positions or fewer.

MULTIPLY LOGICAL

Register-and-register formats:



Register-and-storage formats:



The 32-bit or 64-bit first operand (the multiplicand) is multiplied by the 32-bit or 64-bit second operand (the multiplier), and the 64-bit or 128-bit product is placed at the first-operand location.

The R₁ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

For MULTIPLY LOGICAL (MLR, ML), both the multiplicand and the multiplier are treated as 32-bit unsigned binary integers. The multiplicand is in bit positions 32-63 of general register R₁ + 1. For MULTIPLY LOGICAL (MLR), the multiplier is in bit positions 32-63 of general register R₂. The contents of general register R₁ and of bit positions 0-31 of general register R₁ + 1 and, for MLR, of general register R₂ are ignored. The product is a 64-bit unsigned binary integer. Bits 0-31 of the product replace bits 32-63 of general register R₁, and bits 32-63 of the product replace bits 32-63 of general register R₁ + 1. Bits 0-31 of general registers R₁ and R₁ + 1 remain unchanged. An overflow cannot occur.

For MULTIPLY LOGICAL (MLGR, MLG), the multiplicand and the multiplier are treated as 64-bit unsigned binary integers. The multiplicand is in general regis-

ter R₁ + 1. The contents of general register R₁ are ignored. The product is a 128-bit unsigned binary integer. Bits 0-63 of the product replace the contents of general register R₁, and bits 64-127 of the product replace the contents of general register R₁ + 1. An overflow cannot occur.

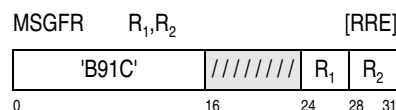
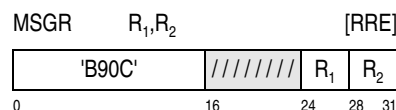
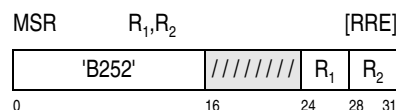
Condition Code: The code remains unchanged.

Program Exceptions:

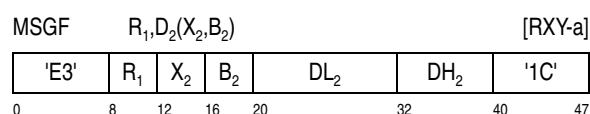
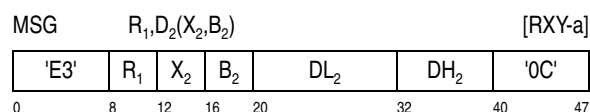
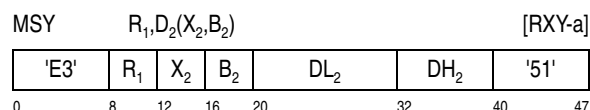
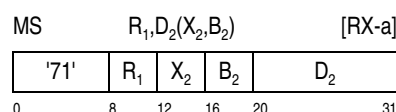
- Access (fetch, operand 2 of ML and MLG only)
- Specification

MULTIPLY SINGLE

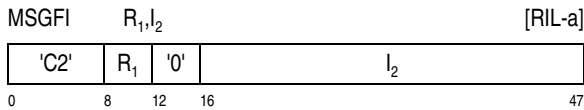
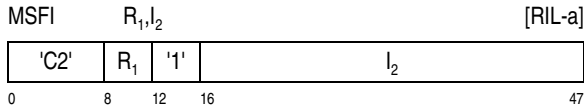
Register-and-register formats:



Register-and-storage formats:



MULTIPLY SINGLE IMMEDIATE



The first operand (multiplicand) is multiplied by the second operand (multiplier), and the rightmost 32 or 64 bits of the product are placed at the first-operand location.

For MULTIPLY SINGLE (MSR, MS, MSY) and for MULTIPLY SINGLE IMMEDIATE (MSFI), the multiplicand, multiplier, and product are treated as 32-bit signed binary integers. The multiplicand is taken from bit positions 32-63 of general register R_1 and is replaced by the rightmost 32 bits of the signed-binary-integer product. Bits 0-31 of general register R_1 remain unchanged. For MSR, the multiplier is in bit positions 32-63 of general register R_2 . For MSFI, the multiplier is in the I_2 field of the instruction. The bits to the left of the 32 rightmost bits of the product are not tested for significance; no overflow indication is given.

For MULTIPLY SINGLE (MSGR, MSGFR, MSG, MSGF) and for MULTIPLY SINGLE IMMEDIATE (MSGFI), the multiplicand, multiplier, and product are treated as 64-bit signed binary integers, except that, for MSGF, MSGFI, and MSGFR, the multiplier is treated as a 32-bit signed binary integer. The multiplicand is taken from general register R_1 and is replaced by the rightmost 64 bits of the signed-binary-integer product. For MSGFR, the multiplier is in bit positions 32-63 of general register R_2 . For MSGFI, the multiplier is in the I_2 field of the instruction. The bits to the left of the 64 rightmost bits of the product are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

The displacement for MS is treated as a 12-bit unsigned binary integer. The displacement for MSY,

MSG, and MSGF is treated as a 20-bit signed binary integer.

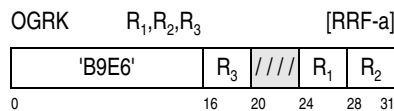
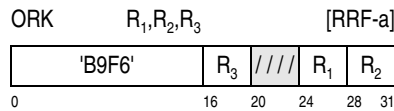
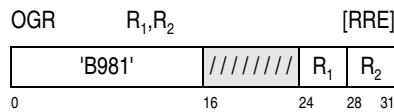
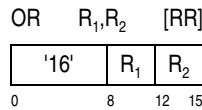
Condition Code: The code remains unchanged.

Program Exceptions:

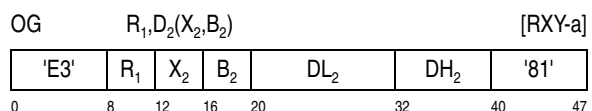
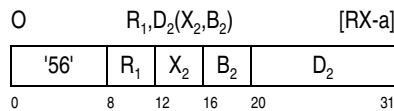
- Access (fetch, operand 2 of MS, MSY, MSG, MSGF only)
- Operation (MSY, if the long-displacement facility is not installed; MSFI, MSGFI, if the general-instructions-extension facility is not installed)

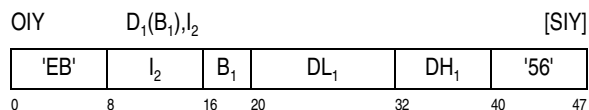
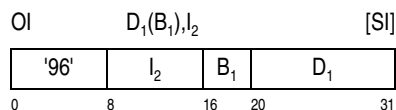
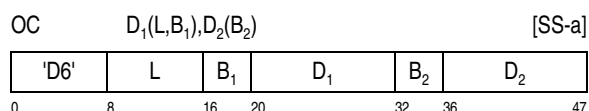
OR

Register-and-register formats:



Register-and-storage formats:



Storage-and-immediate formats:**Storage-and-storage format:**

For O, OC, OG, OGR, OI, OIY, OR, and OY, the OR of the first and second operands is placed at the first-operand location. For OGRK and ORK, the OR of the second and third operands is placed at the first-operand location.

The connective OR is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero.

For OR (OC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

For OR (OI, OIY), the first operand is one byte in length, and only one byte is stored.

For OR (O, OR, ORK, and OY), the operands are 32 bits, and for OR (OG, OGR, and OGRK), they are 64 bits.

The displacements for O, OI, and both operands of OC are treated as 12-bit unsigned binary integers. The displacement for OY, OIY, and OG is treated as a 20-bit signed binary integer.

Resulting Condition Code:

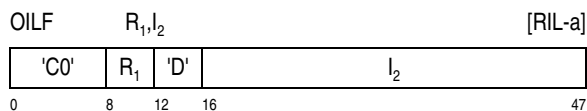
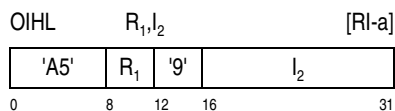
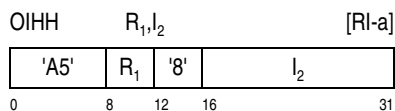
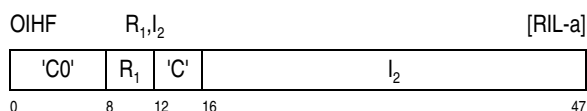
- 0 Result zero
- 1 Result not zero
- 2 --
- 3 --

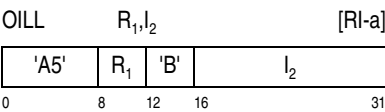
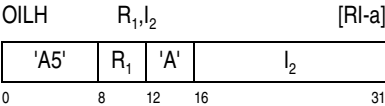
Program Exceptions:

- Access (fetch, operand 2, O, OY, OG, and OC; fetch and store, operand 1, OI, OIY, and OC)
- Operation (OY and OIY, if the long-displacement facility is not installed; OGRK and ORK, if the distinct-operands facility is not installed)

Programming Notes:

- Examples of the use of the OR instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
- OR may be used to set a bit to one.
- Accesses to the first operand of OR (OI) and OR (OC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown in "Multiprogramming and Multiprocessing Examples" in Appendix A, "Number Representation and Instruction-Use Examples."

OR IMMEDIATE



The second operand is ORed with bits of the first operand, and the result replaces those bits of the first operand. The remainder of the first operand remains unchanged.

For each instruction, the bits of the first operand that are ORed with the second operand and then replaced are as follows:

Instruction	Bits ORed and Replaced
OIHF	0-31
OIHH	0-15
OIHL	16-31
OILF	32-63
OILH	32-47
OILL	48-63

The connective OR is applied to the operands bit by bit. The contents of a bit position in the result are set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero.

Resulting Condition Code:

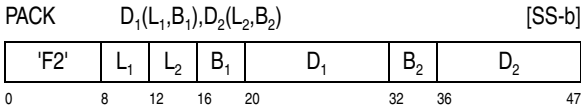
- 0 Result is zero
- 1 Result is not zero
- 2 --
- 3 --

Program Exceptions:

- Operation (OIHF and OILF, if the extended-immediate facility is not installed)

Programming Note: The setting of the condition code is based only on the bits that are ORed and replaced.

PACK



The format of the second operand is changed from zoned to signed-packed-decimal, and the result is placed at the first-operand location. The zoned and signed-packed-decimal formats are described in Chapter 8, “Decimal Instructions.”

The second operand is treated as having the zoned format. The numeric bits of each byte are treated as a digit. The zone bits are ignored, except the zone bits in the rightmost byte, which are treated as a sign.

The sign and digits are moved unchanged to the first operand and are not checked for valid codes. The sign is placed in the rightmost four bit positions of the rightmost byte of the result field, and the digits are placed adjacent to the sign and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if each result byte were stored immediately after fetching the necessary operand bytes. Two second-operand bytes are needed for each result byte, except for the rightmost byte of the result field, which requires only the rightmost second-operand byte.

Condition Code: The code remains unchanged.

Program Exceptions:

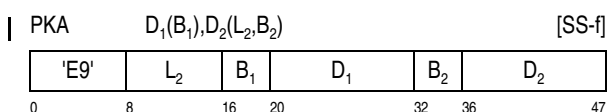
- Access (fetch, operand 2; store, operand 1)

Programming Notes:

1. An example of the use of the PACK instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”

2. PACK may be used to interchange the two hexadecimal digits in one byte by specifying a zero in the L_1 and L_2 fields and the same address for both operands.
3. To remove the zone bits of all bytes of a field, including the rightmost byte, both operands should be extended on the right with a dummy byte, which subsequently should be ignored in the result field.
4. The storage-operand references for PACK may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)

PACK ASCII



The format of the second operand is changed from ASCII to signed-packed-decimal, and the result is placed at the first-operand location. The signed-packed-decimal format is described in Chapter 8, "Decimal Instructions."

The second-operand bytes are treated as containing decimal digits, having the binary encoding 0000-1001 for 0-9, in their rightmost four bit positions. The leftmost four bit positions of a byte are ignored. The second operand is considered to be positive.

The implied positive sign (1100 binary) and the source digits are placed at the first-operand location. The source digits are moved unchanged and are not checked for valid codes. The sign is placed in the rightmost four bit positions of the rightmost byte of the result field, and the digits are placed adjacent to the sign and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros.

The length of the first operand is 16 bytes.

The length of the second operand is designated by the contents of the L_2 field. The second-operand length must not exceed 32 bytes (L_2 must be less than or equal to 31); otherwise, a specification exception is recognized.

When the length of the second operand is 32 bytes, the leftmost byte is ignored.

The results are unpredictable if the first and second operands overlap in any way.

As observed by other CPUs and by channel programs, the first-operand location is not necessarily stored into in any particular order.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

Programming Notes:

1. Although PACK ASCII is primarily intended to change the format of ASCII decimal digits, its use is not restricted to ASCII since the leftmost four bits of each byte are ignored.
2. The following example illustrates the use of the instruction to pack ASCII digits:

```

ASDIGITS DS    0CL31
          DC    X'3132333435'
          DC    X'3637383930'
          DC    X'3132333435'
          DC    X'3637383930'
          DC    X'3132333435'
          DC    X'3637383930'
          DC    X'31'
PKDIGITS DS    PL16
          :
          PKA    PKDIGITS,ASDIGITS(31)

```

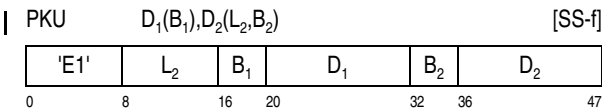
3. The instruction can also be used to pack EBCDIC digits, which is especially useful when

the length of the second operand is greater than the 16-byte second-operand limit of PACK.

```
EBDIGITS DS 0CL31
          DC X'F1F2F3F4F5'
          DC X'F6F7F8F9F0'
          DC X'F1F2F3F4F5'
          DC X'F6F7F8F9F0'
          DC X'F1F2F3F4F5'
          DC X'F6F7F8F9F0'
          DC X'F1'
PKDIGITS DS PL16
          :
          PKA PKDIGITS,EBDIGITS(31)
```

4. The storage-operand references for PACK ASCII may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

PACK UNICODE



The format of the second operand is changed from Unicode to signed-packed-decimal, and the result is placed at the first-operand location. The signed-packed-decimal format is described in Chapter 8, “Decimal Instructions.”

The two-byte second-operand characters are treated as Unicode Basic Latin characters containing decimal digits, having the binary encoding 0000-1001 for 0-9, in their rightmost four bit positions. The leftmost 12 bit positions of a character are ignored. The second operand is considered to be positive.

The implied positive sign (1100 binary) and the source digits are placed at the first-operand location. The source digits are moved unchanged and are not checked for valid codes. The sign is placed in the rightmost four bit positions of the rightmost byte of the result field, and the digits are placed adjacent to the sign and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left. When necessary, the second

operand is considered to be extended on the left with zeros.

The length of the first operand is 16 bytes.

The byte length of the second operand is designated by the contents of the L₂ field. The second-operand length must not exceed 32 characters or 64 bytes, and the byte length must be even (L₂ must be less than or equal to 63 and must be odd); otherwise, a specification exception is recognized.

When the length of the second operand is 32 characters (64 bytes), the leftmost character is ignored.

The results are unpredictable if the first and second operands overlap in any way.

As observed by other CPUs and by channel programs, the first-operand location is not necessarily stored into in any particular order.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

Programming Notes:

1. The following example illustrates the use of PACK UNICODE to pack European numbers:

```
UNDIGITS DS 0CL62
          DC X'00310032003300340035'
          DC X'00360037003800390030'
          DC X'00310032003300340035'
          DC X'00360037003800390030'
          DC X'00310032003300340035'
          DC X'00360037003800390030'
          DC X'0031'
PKDIGITS DS PL16
          :
          PKU PKDIGITS,UNDIGITS(62)
```

2. Because the leftmost 12 bits of each character are ignored, those Unicode decimal digits where the digit zero has four rightmost zero bits can

also be packed by the instruction. For example, for Thai digits:

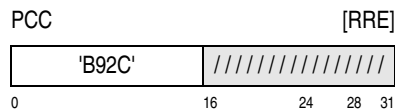
```

UNDIGITS DS 0CL62
DC X'0E510E520E530E540E55'
DC X'0E560E570E580E590E50'
DC X'0E510E520E530E540E55'
DC X'0E560E570E580E590E50'
DC X'0E510E520E530E540E55'
DC X'0E560E570E580E590E50'
DC X'0E51'
PKDIGITS DS PL16
:
PKU PKDIGITS,UNDIGITS(62)

```

3. The storage-operand references for PACK UNICODE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

PERFORM CRYPTOGRAPHIC COMPUTATION



A function specified by the function code in general register 0 is performed.

Bits 16-31 of the instruction are ignored.

Bit positions 57-63 of general register 0 contain the function code. Figure 7-255 on page 7-255 shows the assigned function codes. All other function codes are unassigned. Bit 56 of general register 0 must be zero; otherwise, a specification exception is recognized. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

The function codes for PERFORM CRYPTOGRAPHIC COMPUTATION are as follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	PCC-Query	16	—
1	PCC-Compute-Last-Block-CMAC-Using-DEA	32	8
2	PCC-Compute-Last-Block-CMAC-Using-TDEA-128	40	8
3	PCC-Compute-Last-Block-CMAC-Using-TDEA-192	48	8
9	PCC-Compute-Last-Block-CMAC-Using-Encrypted-DEA	56	8
10	PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-128	64	8
11	PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-192	72	8
18	PCC-Compute-Last-Block-CMAC-Using-AES-128	56	16
19	PCC-Compute-Last-Block-CMAC-Using-AES-192	64	16
20	PCC-Compute-Last-Block-CMAC-Using-AES-256	72	16
26	PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-128	88	16
27	PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-192	96	16
28	PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-256A	104	16
50	PCC-Compute-XTS-Parameter-Using-AES-128	80	16
52	PCC-Compute-XTS-Parameter-Using-AES-256	96	16
58	PCC-Compute-XTS-Parameter-Using-Encrypted-AES-128	112	16
60	PCC-Compute-XTS-Parameter-Using-Encrypted-AES-256	128	16
Explanation:			
— Not applicable			

Figure 7-255. Function Codes for PERFORM CRYPTOGRAPHIC COMPUTATION

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions.

Figure 7-256 on page 7-256 shows the contents of general registers 0 and 1.

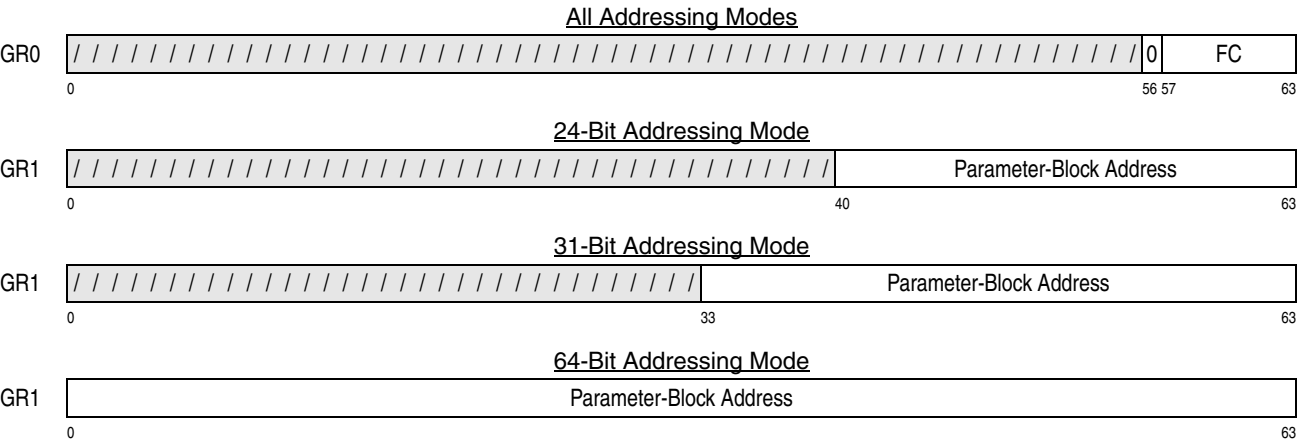


Figure 7-256. General Register Assignment for PCC

In the access-register mode, access register 1 specifies the address space containing the parameter block.

As observed by other CPUs and channel programs, reference to the parameter block may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

Symbols Used in Function Descriptions

The following symbols are used in the subsequent description of the PERFORM CRYPTOGRAPHIC COMPUTATION functions. Further description of the AES standard may be found in *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, dated November 26, 2001. The XTS multiplication operation is the same as the GCM (Galois/counter mode) multiplication operation. Further description of the GCM multiplication over $GF(2^{128})$ may be found in the draft of NIST Special Publication 800-38D, dated June 2007.

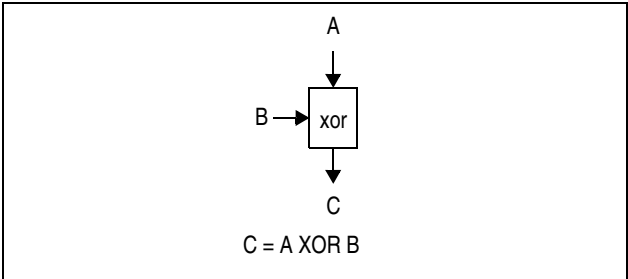


Figure 7-257. Symbol For Bit-Wise Exclusive OR

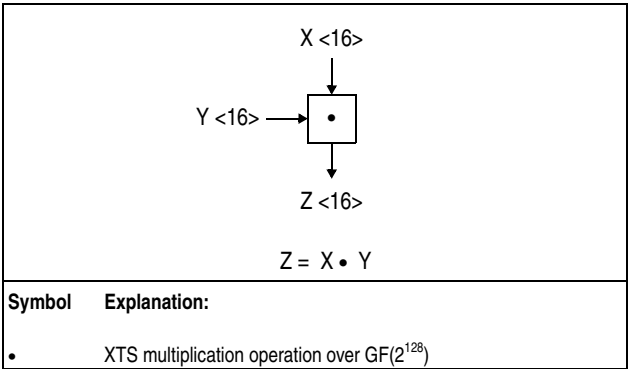


Figure 7-258. Symbol For XTS Multiplication Operation Over $GF(2^{128})$

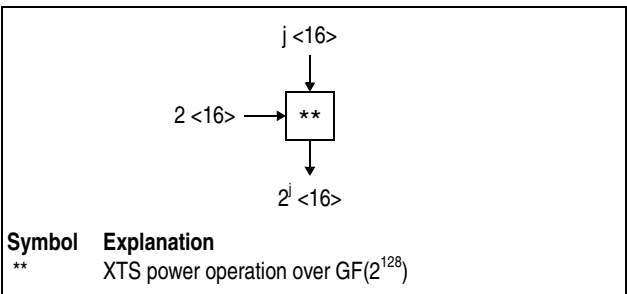


Figure 7-259. Symbol For XTS Power Operation Over $GF(2^{128})$

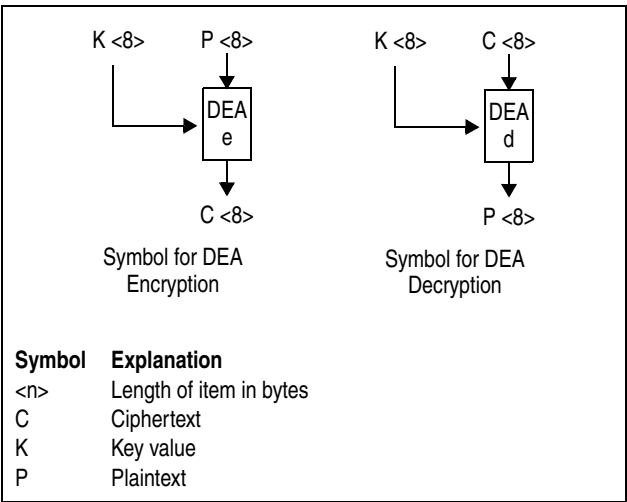


Figure 7-260. Symbols for DEA Encryption and Decryption

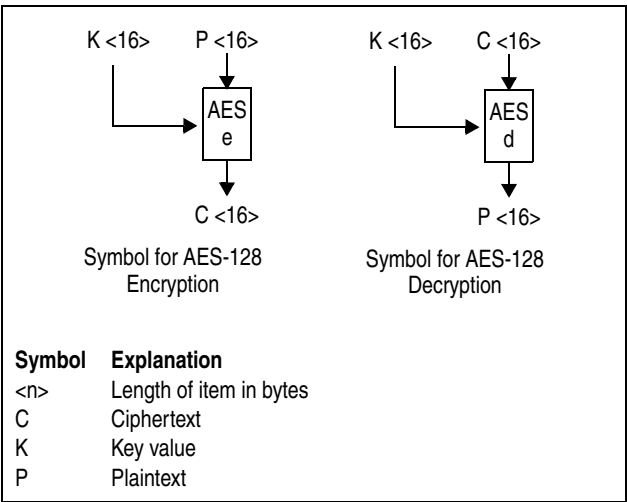


Figure 7-261. Symbols for AES-128 Encryption and Decryption

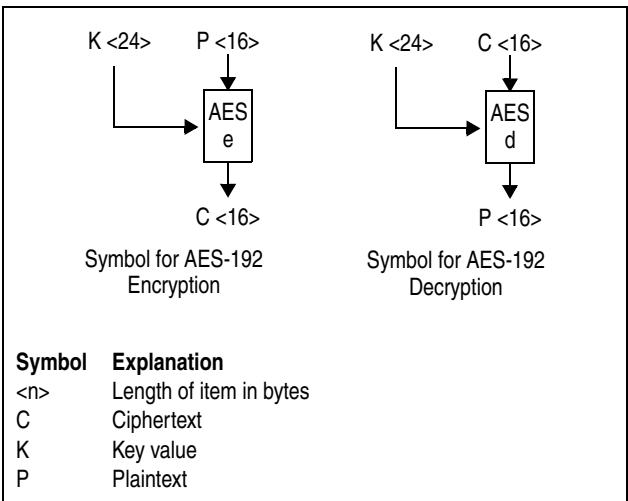


Figure 7-262. Symbols for AES-192 Encryption and Decryption

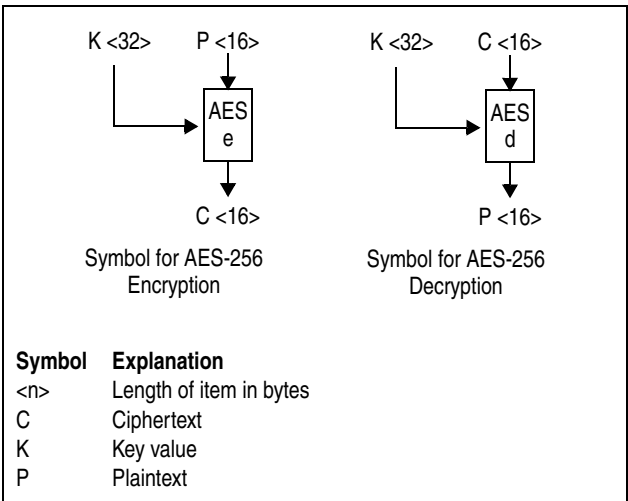


Figure 7-263. Symbols for AES-256 Encryption and Decryption

PCC-Query (Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the function has the following format:

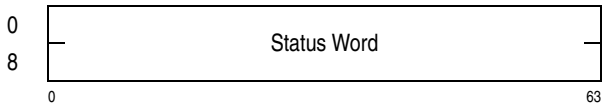


Figure 7-264. Parameter Block for PCC-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function

codes 0-127, respectively, of the PCC instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

PCC-Compute-Last-Block-CMAC-Using-DEA (Function Code 1)

PCC-Compute-Last-Block-CMAC-Using-Encrypted-DEA (Function Code 9)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-DEA function has the following format:

0	ML	Reserved
8	Message	
16	Initial Chaining Value (ICV)	
24	Cryptographic Key (K)	
0	63	

Figure 7-265. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-DEA

For the PCC-Compute-Last-Block-CMAC-Using-DEA function, the initial-chaining value is in byte offsets 16-23 of the parameter block and the cryptographic key is in byte offsets 24-31 of the parameter block.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-Encrypted-DEA function has the following format:

0	ML	Reserved
8	Message	
16	Initial Chaining Value (ICV)	
24	Encrypted Cryptographic Key ($WK_d(K)$)	
32	DEA Wrapping-Key	
40	Verification Pattern	
48	(WK_dVP)	
0	63	

Figure 7-266. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-Encrypted-DEA

For the PCC-Compute-Last-Block-CMAC-Using-Encrypted-DEA function, the contents of byte offsets 32-55 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is com-

pleted by setting condition code 1. If they match, byte offsets 16-23 of the parameter block contain the initial chaining value, and the contents of byte offsets 24-31 of the parameter block are deciphered using the DEA wrapping key to obtain the 64-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The description in the following paragraphs applies to both functions.

The bit length of the message in byte offsets 8-15 of the parameter block is specified by an 8-bit unsigned binary integer in the message-length (ML) field at byte offset 0. When the ML field contains a value greater than 64, the operation is completed by setting condition code 2. Otherwise, the ML field specifies the number of leftmost bits in byte offsets 8-15 that constitute the message. All other bits in byte offsets 8-15 are ignored.

When the ML field specifies a value of 0, a 64-bit message block is formed by setting the leftmost bit to one and all other bits to zero; when the ML field specifies a value of 64, the contents of byte offsets 8-15 form the 64-bit message block; when the ML field specifies a value of 63, the leftmost 63 bits in byte offsets 8-15 padded with a bit of one on the right form the 64-bit message block; when the ML field specifies a value in the range between 1 and 62, inclusively, the specified number of leftmost bits in byte offsets 8-15 padded on the right with a bit of one followed by the necessary number of bits of zero form a 64-bit message block.

When the ML field specifies the value of 64, a 64-bit subkey, K_x , is derived using the 64-bit cryptographic key, K. When the ML field specifies a value in the range between 0 and 63, inclusively, a different subkey, K_y , is derived using the 64-bit cryptographic key, K. The subkey generation algorithm is shown in the following figure.

Steps:

1. $L = CIPH_K(0)$
2. If $MSB(L) = 0$, then $K_x = L \ll 1$;
Else, $K_x = (L \ll 1) \text{ XOR } (R_{64})$
3. If $MSB(K_x) = 0$, then $K_y = K_x \ll 1$;
Else $K_y = (K_x \ll 1) \text{ XOR } (R_{64})$

Figure 7-267. Subkey Generation Algorithm

Explanation:

$CIPH_K(0)$	Encryption of the value zero using a 64-bit key and the DEA encryption algorithm.
MSB(A)	Most significant (leftmost) bit of A
$B \ll 1$	The bit string that results from discarding the leftmost bit of B and appending a '0' bit on the right.
XOR	Bit-wise exclusive OR.
R_{64}	A 64-bit value of 27.

Figure 7-267. Subkey Generation Algorithm (Continued)

When the ML field specifies the value of 64, the 64-bit message block is exclusive-ORed with the subkey K_x . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 64-bit cryptographic key, K, and the DEA encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 16-23 of the parameter block.

When the ML field specifies a value in the range between 0 and 63, inclusively, the 64-bit message block is exclusive-ORed with the subkey K_y . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 64-bit cryptographic key, K, and the DEA encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 16-23 of the parameter block.

PCC-Compute-Last-Block-CMAC-Using-TDEA-128 (Function Code 2)

PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-128 (Function Code 10)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-TDEA-128 function has the following format:

0	ML	Reserved
8		Message
16		Initial Chaining Value (ICV)
24		Cryptographic Key (K)
32		
0		63

Figure 7-268. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-TDEA-128

For the PCC-Compute-Last-Block-CMAC-Using-TDEA-128 function, the initial-chaining value is in byte offsets 16-23 of the parameter block and the cryptographic key is in byte offsets 24-39 of the parameter block.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-128 function has the following format:

0	ML	Reserved
8		Message
16		Initial Chaining Value (ICV)
24		Encrypted Cryptographic Key ($WK_0(K)$)
32		
40		DEA Wrapping-Key Verification Pattern ($WK_0(VP)$)
48		
56		
0		63

Figure 7-269. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-128

For the PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-128 function, the contents of byte offsets 40-63 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 16-23 of the parameter block contain the initial chaining value, and the contents of byte offsets 24-39 of the parameter block are deciphered using the DEA wrapping key to obtain the 128-bit cryptographic key, K. (See the section "Protection of Cryptographic Key" on page 7-339 for details.)

The description in the following paragraphs applies to both functions.

The bit length of the message in byte offsets 8-15 of the parameter block is specified by an 8-bit unsigned binary integer in the message-length (ML) field at byte offset 0. When the ML field contains a value greater than 64, the operation is completed by setting condition code 2. Otherwise, the ML field specifies the number of leftmost bits in byte offsets 8-15 that constitute the message. All other bits in byte offsets 8-15 are ignored.

When the ML field specifies a value of 0, a 64-bit message block is formed by setting the leftmost bit to one and all other bits to zero; when the ML field specifies a value of 64, the contents of byte offsets 8-15 form the 64-bit message block; when the ML field specifies a value of 63, the leftmost 63 bits in byte offsets 8-15 padded with a bit of one on the right form the 64-bit message block; when the ML field specifies a value in the range between 1 and 62, inclusively, the specified number of leftmost bits in byte offsets 8-15 padded on the right with a bit of one followed by the necessary number of bits of zero form a 64-bit message block.

When the ML field specifies the value of 64, a 64-bit subkey, K_x , is derived using the 128-bit cryptographic key, K . When the ML field specifies a value in the range between 0 and 63, inclusively, a different subkey, K_y , is derived using the 128-bit cryptographic key, K . The subkey generation algorithm is shown in the following figure.

Steps:

1. $L = CIPH_K(0)$
2. If $MSB(L) = 0$, then $K_x = L \ll 1$;
Else, $K_x = (L \ll 1) \text{ XOR } (R_{64})$
3. If $MSB(K_x) = 0$, then $K_y = K_x \ll 1$;
Else $K_y = (K_x \ll 1) \text{ XOR } (R_{64})$

Explanation:

$CIPH_K(0)$ Encryption of the value zero using a 128-bit key and the TDEA-128 encryption algorithm.

$MSB(A)$ Most significant (leftmost) bit of A

$B \ll 1$ The bit string that results from discarding the leftmost bit of B and appending a '0' bit on the right.

XOR Bit-wise exclusive OR.

R_{64} A 64-bit value of 27.

Figure 7-270. Subkey Generation Algorithm

When the ML field specifies the value of 64, the 64-bit message block is exclusive-ORed with the subkey K_x . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 128-bit cryptographic key, K , and the TDEA-128 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 16-23 of the parameter block.

When the ML field specifies a value in the range between 0 and 63, inclusively, the 64-bit message block is exclusive-ORed with the subkey K_y . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 128-bit cryptographic key, K , and the TDEA-128 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 16-23 of the parameter block.

PCC-Compute-Last-Block-CMAC-Using-TDEA-192 (Function Code 3)

PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-192 (Function Code 11)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-TDEA-192 function has the following format:

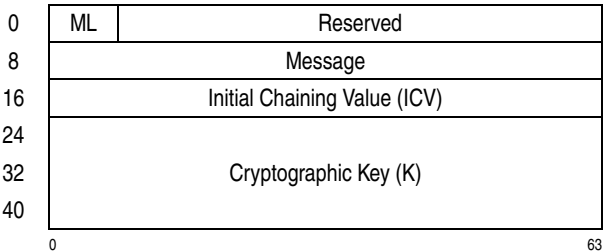


Figure 7-271. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-TDEA-192

For the PCC-Compute-Last-Block-CMAC-Using-TDEA-192 function, the initial-chaining value is in byte offsets 16-23 of the parameter block and the cryptographic key is in byte offsets 24-47 of the parameter block.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-192 function has the following format:

0	ML	Reserved
8	Message	
16	Initial Chaining Value (ICV)	
24	Encrypted Cryptographic Key ($WK_d(K)$)	
32		
40		
48	DEA Wrapping-Key Verification Pattern (WK_dVP)	
56		
64		
0		63

Figure 7-272. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-192

For the PCC-Compute-Last-Block-CMAC-Using-Encrypted-TDEA-192 function, the contents of byte offsets 48-71 of the parameter block are compared with the contents of the DEA wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 16-23 of the parameter block contain the initial chaining value, and the contents of byte offsets 24-47 of the parameter block are deciphered using the DEA wrapping key to obtain the 192-bit cryptographic key, K . (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The description in the following paragraphs applies to both functions.

The bit length of the message in byte offsets 8-15 of the parameter block is specified by an 8-bit unsigned binary integer in the message-length (ML) field at byte offset 0. When the ML field contains a value greater than 64, the operation is completed by setting condition code 2. Otherwise, the ML field specifies the number of leftmost bits in byte offsets 8-15 that constitute the message. All other bits in byte offsets 8-15 are ignored.

When the ML field specifies a value of 0, a 64-bit message block is formed by setting the leftmost bit to one and all other bits to zero; when the ML field specifies a value of 64, the contents of byte offsets 8-15 form the 64-bit message block; when the ML field specifies a value of 63, the leftmost 63 bits in byte offsets 8-15 padded with a bit of one on the right form the 64-bit message block; when the ML field speci-

fies a value in the range between 1 and 62, inclusively, the specified number of leftmost bits in byte offsets 8-15 padded on the right with a bit of one followed by the necessary number of bits of zero form a 64-bit message block.

When the ML field specifies the value of 64, a 64-bit subkey, K_x , is derived using the 192-bit cryptographic key, K . When the ML field specifies a value in the range between 0 and 63, inclusively, a different subkey, K_y , is derived using the 192-bit cryptographic key, K . The subkey generation algorithm is shown in the following figure.

Steps:	
1.	$L = CIPH_K(0)$
2.	If MSB (L) = 0, then $K_x = L \ll 1$; Else, $K_x = (L \ll 1) \text{ XOR } (R_{64})$
3.	If MSB (K_x) = 0, then $K_y = K_x \ll 1$; Else $K_y = (K_x \ll 1) \text{ XOR } (R_{64})$
Explanation:	
$CIPH_K(0)$	Encryption of the value zero using a 192-bit key and the TDEA-192 encryption algorithm.
MSB(A)	Most significant (leftmost) bit of A
$B \ll 1$	The bit string that results from discarding the leftmost bit of B and appending a '0' bit on the right.
XOR	Bit-wise exclusive OR.
R_{64}	A 64-bit value of 27.

Figure 7-273. Subkey Generation Algorithm

When the ML field specifies the value of 64, the 64-bit message block is exclusive-ORed with the subkey K_x . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then encrypted using the 192-bit cryptographic key, K , and the TDEA-192 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 16-23 of the parameter block.

When the ML field specifies a value in the range between 0 and 63, inclusively, the 64-bit message block is exclusive-ORed with the subkey K_y . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then encrypted using the 192-bit cryptographic key, K , and the TDEA-192 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 16-23 of the parameter block.

PCC-Compute-Last-Block-CMAC-Using-AES-128 (Function Code 18)

PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-128 (Function Code 26)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-AES-128 function has the following format:

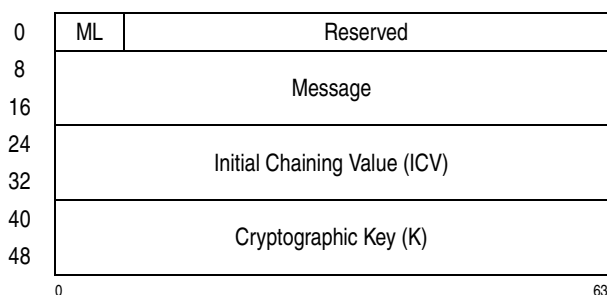


Figure 7-274. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-AES-128

For the PCC-Compute-Last-Block-CMAC-Using-AES-128 function, the initial-chaining value is in byte offsets 24-39 of the parameter block and the cryptographic key is in byte offsets 40-55 of the parameter block.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-128 function has the following format:

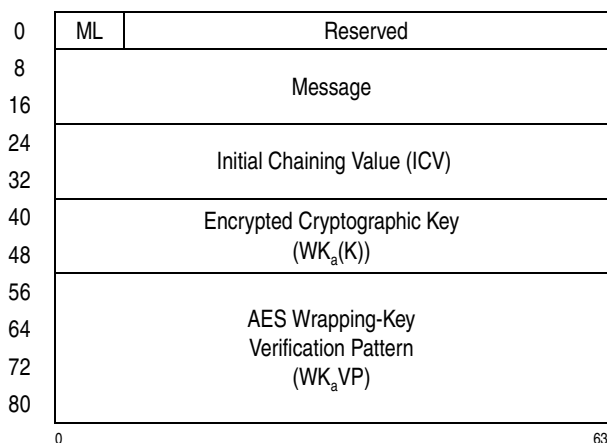


Figure 7-275. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-128

For the PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-128 function, the contents of byte offsets 56-87 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 24-39 of the parameter block contain the initial chaining value, and the contents of byte offsets 40-55 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The description in the following paragraphs applies to both functions.

The bit length of the message in byte offsets 8-23 of the parameter block is specified by an 8-bit unsigned binary integer in the message-length (ML) field at byte offset 0. When the ML field contains a value greater than 128, the operation is completed by setting condition code 2. Otherwise, the ML field specifies the number of leftmost bits in byte offsets 8-23 that constitute the message. All other bits in byte offsets 8-23 are ignored.

When the ML field specifies a value of 0, a 128-bit message block is formed by setting the leftmost bit to one and all other bits to zero; when the ML field specifies a value of 128, the contents of byte offsets 8-23 form the 128-bit message block; when the ML field specifies a value of 127, the leftmost 127 bits in byte offsets 8-23 padded with a bit of one on the right form the 128-bit message block; when the ML field specifies a value in the range between 1 and 126, inclusively, the specified number of leftmost bits in byte offsets 8-23 padded on the right with a bit of one followed by the necessary number of bits of zero form a 128-bit message block.

When the ML field specifies the value of 128, a 128-bit subkey, K_x , is derived using the 128-bit cryptographic key, K. When the ML field specifies a value in the range between 0 and 127, inclusively, a different subkey, K_y , is derived using the 128-bit cryptographic

key, K. The subkey generation algorithm is shown in the following figure.

Steps:	
1.	$L = \text{CIPH}_K(0)$
2.	If $\text{MSB}(L) = 0$, then $K_X = L \ll 1$; Else, $K_X = (L \ll 1) \text{ XOR } (R_{128})$
3.	If $\text{MSB}(K_X) = 0$, then $K_Y = K_X \ll 1$; Else $K_Y = (K_X \ll 1) \text{ XOR } (R_{128})$
Explanation:	
$\text{CIPH}_K(0)$	Encryption of the value zero using a 128-bit key and the AES-128 encryption algorithm.
$\text{MSB}(A)$	Most significant (leftmost) bit of A
$B \ll 1$	The bit string that results from discarding the leftmost bit of B and appending a '0' bit on the right.
XOR	Bit-wise exclusive OR.
R_{64}	A 128-bit value of 135.

Figure 7-276. Subkey Generation Algorithm

When the ML field specifies the value of 128, the 128-bit message block is exclusive-ORed with the subkey K_X . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 128-bit cryptographic key, K, and the AES-128 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 24-39 of the parameter block.

When the ML field specifies a value in the range between 0 and 127, inclusively, the 128-bit message block is exclusive-ORed with the subkey K_Y . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 128-bit cryptographic key, K, and the AES-128 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 24-39 of the parameter block.

PCC-Compute-Last-Block-CMAC-Using-AES-192 (Function Code 19)

PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-192 (Function Code 27)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-AES-192 function has the following format:

0	ML	Reserved
8		
16		Message
24		
32		Initial Chaining Value (ICV)
40		
48		Cryptographic Key (K)
56		
63		

Figure 7-277. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-AES-192

For the PCC-Compute-Last-Block-CMAC-Using-AES-192 function, the initial-chaining value is in byte offsets 24-39 of the parameter block and the cryptographic key is in byte offsets 40-63 of the parameter block.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-192 function has the following format:

0	ML	Reserved
8		
16		Message
24		
32		Initial Chaining Value (ICV)
40		
48		Encrypted Cryptographic Key ($WK_a(K)$)
56		
64		AES Wrapping-Key Verification Pattern ($WK_a(VP)$)
72		
80		
88		
96		

Figure 7-278. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-192

For the PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-192 function, the contents of byte offsets 64-95 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, byte offsets 24-39 of the parameter block contain the initial chaining value, and the contents of

byte offsets 40-63 of the parameter block are deciphered using the AES wrapping key to obtain the 192-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The description in the following paragraphs applies to both functions.

The bit length of the message in byte offsets 8-23 of the parameter block is specified by an 8-bit unsigned binary integer in the message-length (ML) field at byte offset 0. When the ML field contains a value greater than 128, the operation is completed by setting condition code 2. Otherwise, the ML field specifies the number of leftmost bits in byte offsets 8-23 that constitute the message. All other bits in byte offsets 8-23 are ignored.

When the ML field specifies a value of 0, a 128-bit message block is formed by setting the leftmost bit to one and all other bits to zero; when the ML field specifies a value of 128, the contents of byte offsets 8-23 form the 128-bit message block; when the ML field specifies a value of 127, the leftmost 127 bits in byte offsets 8-23 padded with a bit of one on the right form the 128-bit message block; when the ML field specifies a value in the range between 1 and 126, inclusively, the specified number of leftmost bits in byte offsets 8-23 padded on the right with a bit of one followed by the necessary number of bits of zero form a 128-bit message block.

When the ML field specifies the value of 128, a 128-bit subkey, K_x , is derived using the 192-bit cryptographic key, K. When the ML field specifies a value in the range between 0 and 127, inclusively, a different subkey, K_y , is derived using the 192-bit cryptographic key, K. The subkey generation algorithm is shown in the following figure.

Steps:

1. $L = CIPH_K(0)$

2. If $MSB(L) = 0$, then $K_x = L \ll 1$;
Else, $K_x = (L \ll 1) \text{ XOR } (R_{128})$

3. If $MSB(K_x) = 0$, then $K_y = K_x \ll 1$;
Else $K_y = (K_x \ll 1) \text{ XOR } (R_{128})$

Figure 7-279. Subkey Generation Algorithm

Explanation:

$CIPH_K(0)$

Encryption of the value zero using a 192-bit key and the AES-192 encryption algorithm.

$MSB(A)$

Most significant (leftmost) bit of A

$B \ll 1$

The bit string that results from discarding the leftmost bit of B and appending a '0' bit on the right.

XOR

Bit-wise exclusive OR.

R_{64}

A 128-bit value of 135.

Figure 7-279. Subkey Generation Algorithm (Continued)

When the ML field specifies the value of 128, the 128-bit message block is exclusive-ORed with the subkey K_x . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 192-bit cryptographic key, K, and the AES-192 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 24-39 of the parameter block.

When the ML field specifies a value in the range between 0 and 127, inclusively, the 128-bit message block is exclusive-ORed with the subkey K_y . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 192-bit cryptographic key, K, and the AES-192 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 24-39 of the parameter block.

PCC-Compute-Last-Block-CMAC-Using-AES-256 (Function Code 20)

PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-256(Function Code 28)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-AES-256 function has the following format:

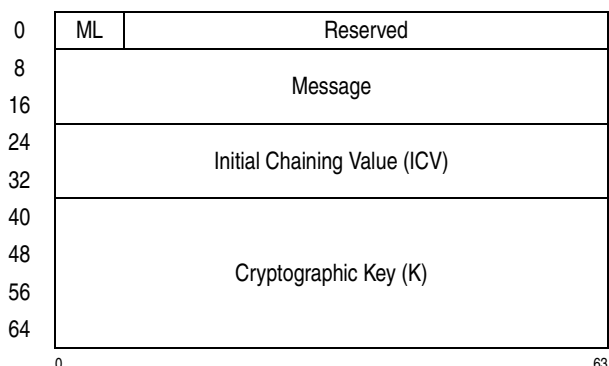


Figure 7-280. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-AES-256

For the PCC-Compute-Last-Block-CMAC-Using-AES-256 function, the initial-chaining value is in byte offsets 24-39 of the parameter block and the cryptographic key is in byte offsets 40-71 of the parameter block.

The parameter block used for the PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-256 function has the following format:

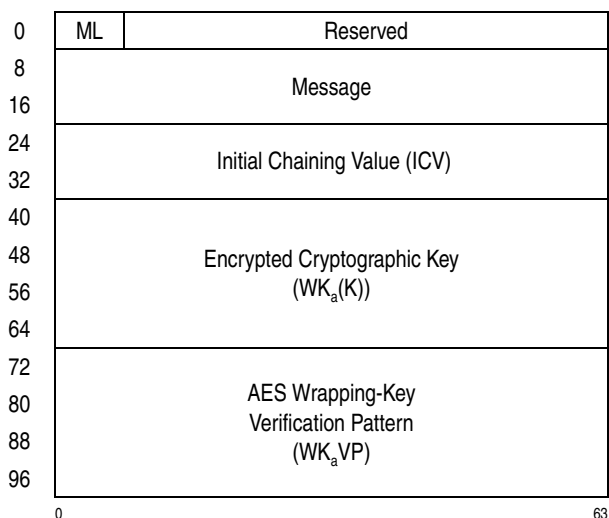


Figure 7-281. Parameter Block for PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-256

For the PCC-Compute-Last-Block-CMAC-Using-Encrypted-AES-256 function, the contents of byte offsets 72-103 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation

is completed by setting condition code 1. If they match, byte offsets 24-39 of the parameter block contain the initial chaining value, and the contents of byte offsets 40-71 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.)

The description in the following paragraphs applies to both functions.

The bit length of the message in byte offsets 8-23 of the parameter block is specified by an 8-bit unsigned binary integer in the message-length (ML) field at byte offset 0. When the ML field contains a value greater than 128, the operation is completed by setting condition code 2. Otherwise, the ML field specifies the number of leftmost bits in byte offsets 8-23 that constitute the message. All other bits in byte offsets 8-23 are ignored.

When the ML field specifies a value of 0, a 128-bit message block is formed by setting the leftmost bit to one and all other bits to zero; when the ML field specifies a value of 128, the contents of byte offsets 8-23 form the 128-bit message block; when the ML field specifies a value of 127, the leftmost 127 bits in byte offsets 8-23 padded with a bit of one on the right form the 128-bit message block; when the ML field specifies a value in the range between 1 and 126, inclusively, the specified number of leftmost bits in byte offsets 8-23 padded on the right with a bit of one followed by the necessary number of bits of zero form a 128-bit message block.

When the ML field specifies the value of 128, a 128-bit subkey, K_x , is derived using the 256-bit cryptographic key, K. When the ML field specifies a value in the range between 0 and 127, inclusively, a different subkey, K_y , is derived using the 256-bit cryptographic key, K. The subkey generation algorithm is shown in the following figure.

Steps:

1. $L = CIPH_K(0)$
2. If $MSB(L) = 0$, then $K_x = L \ll 1$;
Else, $K_x = (L \ll 1) \text{ XOR } (R_{128})$
3. If $MSB(K_x) = 0$, then $K_y = K_x \ll 1$;
Else $K_y = (K_x \ll 1) \text{ XOR } (R_{128})$

Figure 7-282. Subkey Generation Algorithm

Explanation:	
$CIPH_K(0)$	Encryption of the value zero using a 256-bit key and the AES-256 encryption algorithm.
$MSB(A)$	Most significant (leftmost) bit of A
$B \ll 1$	The bit string that results from discarding the leftmost bit of B and appending a '0' bit on the right.
XOR	Bit-wise exclusive OR.
R_{64}	A 128-bit value of 135.

Figure 7-282. Subkey Generation Algorithm (Continued)

When the ML field specifies the value of 128, the 128-bit message block is exclusive-ORed with the subkey K_x . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 256-bit cryptographic key, K, and the AES-256 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 24-39 of the parameter block.

When the ML field specifies a value in the range between 0 and 127, inclusively, the 128-bit message block is exclusive-ORed with the subkey K_y . The result of the exclusive-OR operation is then exclusive-ORed with initial-chaining value. The result of the second exclusive-OR operation is then enciphered using the 256-bit cryptographic key, K, and the AES-256 encryption algorithm. The result of the encryption operation is the CMAC and is placed in byte offsets 24-39 of the parameter block.

PCC-Compute-XTS-Parameter-Using-AES-128 (Function Code 50)

PCC-Compute-XTS-Parameter-Using-Encrypted-AES-128 (Function Code 58)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the PCC-Compute-XTS-Parameter-Using-AES-128 function has the following format:

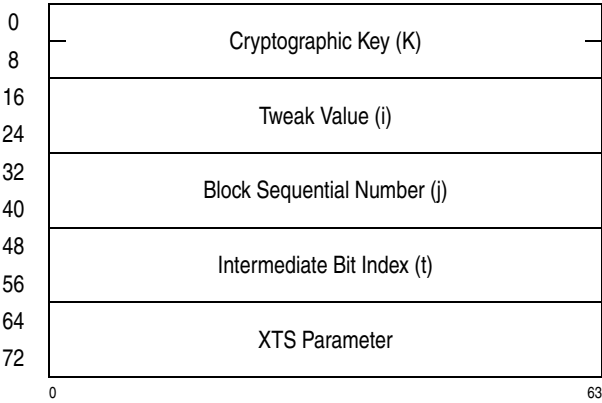


Figure 7-283. Parameter Block for PCC-Compute-XTS-Parameter-Using-AES-128

For the PCC-Compute-XTS-Parameter-Using-AES-128 function, the cryptographic key is in byte offsets 0-15 of the parameter block; the 128-bit tweak value (i) and the 128-bit block sequential number (j) are in byte offsets 16-31 and 32-47, respectively, of the parameter block. When the block sequential number is zero, the contents of byte offsets 48-79 of the parameter block are ignored, the encrypted tweak value is placed in byte offsets 64-79 of the parameter block, and the operation is completed by setting condition code 0. When the block sequential number is nonzero, byte offsets 48-63 of the parameter block contain the index of the next bit in j to be processed. In this case, if the contents of byte offsets 48-63 are zero, then it is the initial execution of the computation, and the contents of byte offsets 64-79 of the parameter block are ignored; if the contents of byte offsets 48-63 are in the range between 1 and 127, inclusively, then byte offsets 64-79 contain the partial XTS parameter computed by a previous iteration; if byte offsets 48-63 of the parameter block contain a value greater than 127, the operation is completed by setting condition code 2.

The parameter block used for the PCC-Compute-XTS-Parameter-Using-Encrypted-AES-128 function has the following format:

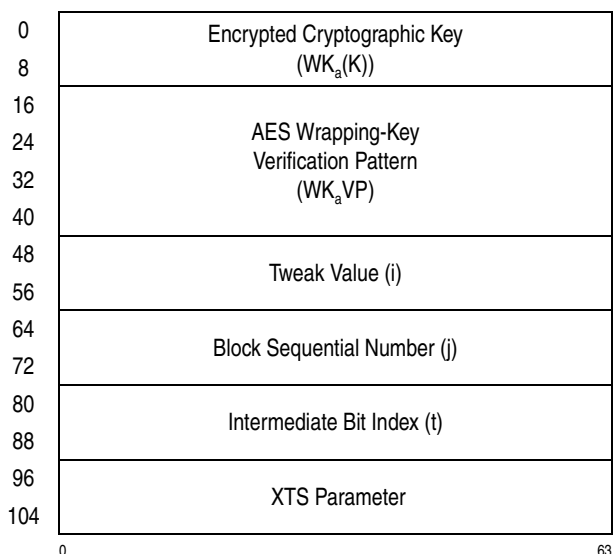


Figure 7-284. Parameter Block for PCC-Compute-XTS-Parameter-Using-Encrypted-AES-128

For the PCC-Compute-XTS-Parameter-Using-Encrypted-AES-128 function, the contents of byte offsets 16-47 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-15 of the parameter block are deciphered using the AES wrapping key to obtain the 128-bit cryptographic key, K . (See the section “Protection of Cryptographic Key” on page 7-339 for details.) The 128-bit tweak value (i) and the 128-bit block sequential number (j) are in byte offsets 48-63 and 64-79, respectively, of the parameter block. When the block sequential number is zero, the contents of byte offsets 80-111 of the parameter block are ignored, the encrypted tweak value is placed in byte offsets 96-111 of the parameter block, and the operation is completed by setting condition code 0. When the block sequential number is nonzero, byte offsets 80-95 of the parameter block contain the index of the next bit in j to be processed. In this case, if the contents of byte offsets 80-95 are zero, then it is the initial execution of the computation, and the contents of byte offsets 96-111 of the parameter block are ignored; if the contents of byte offsets 80-95 are in the range between 1 and 127, inclusively, then byte offsets 96-111 contain the partial XTS parameter computed by a previous iteration; if byte offsets 80-95 of the parameter block contain a

value greater than 127, the operation is completed by setting condition code 2.

The following description applies to both functions.

The 128-bit XTS parameter is the result of multiplying the encrypted tweak value by the value of 2 raised to the power of the block sequential number (j). Both the power and multiplication operations are performed over $GF(2^{128})$. The 128-bit tweak value (i) is encrypted using the 128-bit cryptographic key and the AES encryption algorithm. The operation is shown in Figure 7-285.

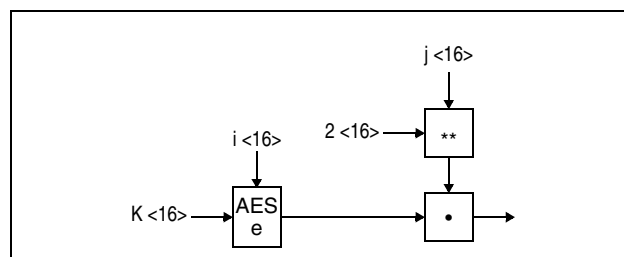


Figure 7-285. Compute XTS Parameter Using AES 128

The value of 2 raised to the power of j is computed as follows: For each bit in j , the value of 2 raised to the power of an exponent, that is a 128-bit unsigned binary integer with an one in that bit and zeros in all other bits, is pre-computed. To obtain the value of 2 raised to the power of j , the pre-computed values that correspond to a bit of one in j are multiplied together. The power and multiplication operations are performed over $GF(2^{128})$.

For a nonzero j , bits in j are processed from left to right, with an index of 0 for the leftmost bit and an index of 127 for the right most bit. The operation is ended when all bits in j have been processed (called normal completion) or when a CPU-determined number of bits that is less than the total number of bits in j have been processed (called partial completion). The CPU-determined number of bits depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of bits is usually nonzero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless reoccurrence of this no-progress case.

When the operation ends due to normal completion, condition code 0 is set and the intermediate bit index field (t) in the parameter block is set to 128. When the operation ends due to partial completion, condition

code 3 is set and the intermediate bit index field (t) in the parameter block is set to the index of the next bit in j to be processed. For the Compute-XTS-Parameter-Using-AES-128 function, the computed XTS parameter or partial XTS parameter is placed in byte offsets 64-79 of the parameter block. For the Compute-XTS-Parameter-Using-Encrypted-AES-128 function, the computed XTS parameter or partial XTS parameter is placed in byte offsets 96-111 of the parameter block.

PCC-Compute-XTS-Parameter-Using-AES-256 (Function Code 52)

PCC-Compute-XTS-Parameter-Using-Encrypted-AES-256 (Function Code 60)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-256 on page 7-256.

The parameter block used for the PCC-Compute-XTS-Parameter-Using-AES-256 function has the following format:

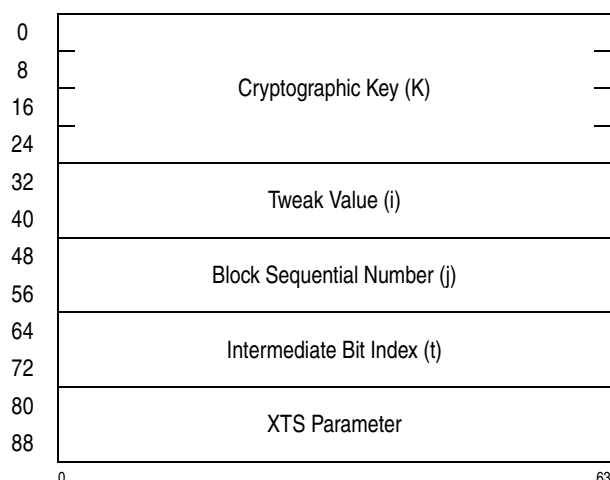


Figure 7-286. Parameter Block for PCC-Compute-XTS-Parameter-Using-AES-256

For the PCC-Compute-XTS-Parameter-Using-AES-256 function, the cryptographic key is in byte offsets 0-31 of the parameter block; the 128-bit tweak value (i) and the 128-bit block sequential number (j) are in byte offsets 32-47 and 48-63, respectively, of the parameter block. When the block sequential number is zero, the contents of byte offsets 64-95 of the parameter block are ignored, the encrypted tweak value is placed in byte offsets 80-95 of the parameter block, and the operation is completed by setting con-

dition code 0. When the block sequential number is nonzero, byte offsets 64-79 of the parameter block contain the index of the next bit in j to be processed. In this case, if the contents of byte offsets 64-79 are zero, then it is the initial execution of the computation, and the contents of byte offsets 80-95 of the parameter block are ignored; if the contents of byte offsets 64-79 are in the range between 1 and 127, inclusively, then byte offsets 80-95 contain the partial XTS parameter computed by a previous iteration; if byte offsets 64-79 of the parameter block contain a value greater than 127, the operation is completed by setting condition code 2.

The parameter block used for the PCC-Compute-XTS-Parameter-Using-Encrypted-AES-256 function has the following format:

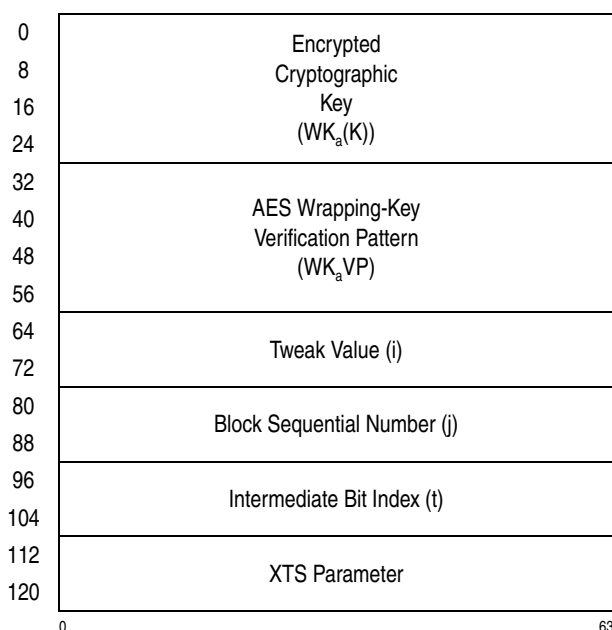


Figure 7-287. Parameter Block for PCC-Compute-XTS-Parameter-Using-Encrypted-AES-256

For the PCC-Compute-XTS-Parameter-Using-Encrypted-AES-256 function, the contents of byte offsets 32-63 of the parameter block are compared with the contents of the AES wrapping-key verification-pattern register. If they mismatch, the operation is completed by setting condition code 1. If they match, the contents of byte offsets 0-31 of the parameter block are deciphered using the AES wrapping key to obtain the 256-bit cryptographic key, K. (See the section “Protection of Cryptographic Key” on page 7-339 for details.) The 128-bit tweak value (i) and the 128-bit block sequential number (j) are in byte offsets 64-79 and 80-95, respectively, of the

parameter block. When the block sequential number is zero, the contents of byte offsets 96-127 of the parameter block are ignored, the encrypted tweak value is placed in byte offsets 112-127 of the parameter block, and the operation is completed by setting condition code 0. When the block sequential number is nonzero, byte offsets 96-111 of the parameter block contain the index of the next bit in j to be processed. In this case, if the contents of byte offsets 96-111 are zero, then it is the initial execution of the computation, and the contents of byte offsets 112-127 of the parameter block are ignored; if the contents of byte offsets 96-111 are in the range between 1 and 127, inclusively, then byte offsets 112-127 contain the partial XTS parameter computed by a previous iteration; if byte offsets 96-111 of the parameter block contain a value greater than 127, the operation is completed by setting condition code 2.

The following description applies to both functions.

The 128-bit XTS parameter is the result of multiplying the encrypted tweak value by the value of 2 raised to the power of the block sequential number (j). Both the power and multiplication operations are performed over $GF(2^{128})$. The 128-bit tweak value (i) is encrypted using the 256-bit cryptographic key and the AES encryption algorithm. The operation is shown in Figure 7-288.

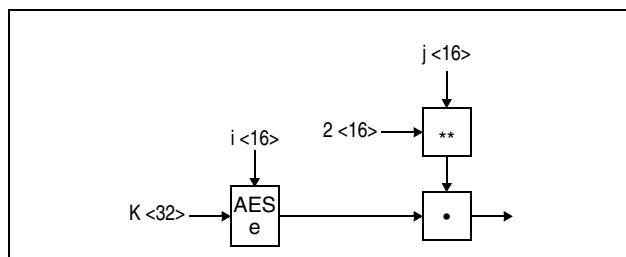


Figure 7-288. Compute XTS Parameter Using AES 256

The value of 2 raised to the power of j is computed as follows: For each bit in j , the value of 2 raised to the power of an exponent, that is a 128-bit unsigned binary integer with an one in that bit and zeros in all other bits, is pre-computed. To obtain the value of 2 raised to the power of j , the pre-computed values that correspond to a bit of one in j are multiplied together. The power and multiplication operations are performed over $GF(2^{128})$.

For a nonzero j , bits in j are processed from left to right, with an index of 0 for the leftmost bit and an index of 127 for the right most bit. The operation is

ended when all bits in j have been processed (called normal completion) or when a CPU-determined number of bits that is less than the total number of bits in j have been processed (called partial completion). The CPU-determined number of bits depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of bits is usually nonzero. In certain unusual situations, this number may be zero, and condition code 3 may be set with no progress. However, the CPU protects against endless reoccurrence of this no-progress case.

When the operation ends due to normal completion, condition code 0 is set and the intermediate bit index field (t) in the parameter block is set to 128. When the operation ends due to partial completion, condition code 3 is set and the intermediate bit index field (t) in the parameter block is set to the index of the next bit in j to be processed. For the Compute-XTS-Parameter-Using-AES-256 function, the computed XTS parameter or partial XTS parameter is placed in byte offsets 80-95 of the parameter block. For the Compute-XTS-Parameter-Using-Encrypted-AES-256 function, the computed XTS parameter or partial XTS parameter is placed in byte offsets 112-127 of the parameter block.

Special Conditions

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bit 56 of general register 0 is not zero.
2. Bits 57-63 of general register 0 specify an unassigned or uninstalled function code.

Resulting Condition Code:

- 0 Normal completion
- 1 Verification pattern mismatch
- 2 Invalid bit index or message length
- 3 Partial completion

Program Exceptions:

- Access (fetch, parameter block; fetch and store, intermediate bit index, XTS parameter, initial chaining value)
- Operation (if the message-security-assist extension 4 facility is not installed)
- Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Operation exception.
8.	Specification exception due to invalid function code or invalid register number.
9.	Specification exception due to invalid operand length.
10.	Condition code 0 due to second-operand length originally zero.
11.	Access exceptions for an access to the parameter block or second operand.
12.A	Condition code 1 due to verification pattern mismatch.
12.B	Condition code 2 due to invalid bit index or message length
13.	Condition code 0 due to normal completion (second-operand length originally nonzero, but stepped to zero).
14.	Condition code 3 due to partial completion (second-operand length still nonzero).

Figure 7-289. Priority of Execution: PCC

Programming Note: e For the initial execution of a PCC Compute-XTS-Parameter function, the contents of the intermediate-bit-index (t) field in the parameter block shall be set to zeros.

PERFORM LOCKED OPERATION

PLO	$R_1, D_2(B_2), R_3, D_4(B_4)$						[SS-e]
'EE'	R_1	R_3	B_2	D_2	B_4	D_4	
0	8	12	16	20	32	36	47

After the lock specified in general register 1 has been obtained, the operation specified by the function code in general register 0 is performed, and then the lock is released. However, as observed by other CPUs: (1) storage operands, including fields in a parameter list that may be used, may be fetched, and may be tested for store-type access exceptions if a store at a tested location is possible, before the lock

is obtained, and (2) operands may be stored in the parameter list after the lock has been released. If an operand not in the parameter list is fetched before the lock is obtained, it is fetched again after the lock has been obtained.

The function code can specify any of six operations: compare and load, compare and swap, double compare and swap, compare and swap and store, compare and swap and double store, or compare and swap and triple store.

A test bit in general register 0 specifies, when one, that a lock is not to be obtained and none of the six operations is to be performed but, instead, the validity of the function code is to be tested. This will be useful if additional function codes for additional operations are assigned in the future. This definition is written as if the test bit is zero except when stated otherwise.

If compare and load is specified, the first-operand comparison value and the second operand are compared. If they are equal, the fourth operand is placed in the third-operand location. If the comparison indicates inequality, the second operand is placed in the first-operand-comparison-value location as a new first-operand comparison value.

If compare and swap is specified, the first-operand comparison value and the second operand are compared. If they are equal, the first-operand replacement value is stored at the second-operand location. If the comparison indicates inequality, the second operand is placed in the first-operand-comparison-value location as a new first-operand comparison value.

If double compare and swap is specified, the first-operand comparison value and the second operand are compared. If they are equal, the third-operand comparison value and the fourth operand are compared. If both comparisons indicate equality, the first-operand and third-operand replacement values are stored at the second-operand location and fourth-operand location, respectively. If the first comparison indicates inequality, the second operand is placed in the first-operand-comparison-value location as a new first-operand comparison value. If the first comparison indicates equality but the second does not, the fourth operand is placed in the third-operand-comparison-value location as a new third-operand comparison value.

If compare and swap and store, double store, or triple store is specified, the first-operand comparison value and the second operand are compared. If they are equal, the first-operand replacement value is stored at the second-operand location, and the third operand is stored at the fourth-operand location. Then, if the operation is the double-store or triple-store operation, the fifth operand is stored at the sixth-operand location, and, if it is the triple-store operation, the seventh operand is stored at the eighth-operand location. If the first-operand comparison indicates inequality, the second operand is placed in the first-operand-comparison-value location as a new first-operand comparison value.

After any of the six operations, the result of the comparison or comparisons is indicated in the condition code.

The function code (FC) is in bit positions 56-63 of general register 0. The function code specifies not only the operation to be performed but also the length of the operands and whether the first-operand comparison and replacement values and the third operand or third-operand comparison and replacement values, which are referred to collectively simply as the first and third operands, are in general registers or a parameter list. The pattern of the function codes is as follows:

- A function code that is a multiple of 4 (including 0) specifies a 32-bit length with the first and third operands in bit positions 32-63 of general registers.
- A function code that is one more than a multiple of 4 specifies a 64-bit length with the first and third operands in a parameter list.
- A function code that is 2 more than a multiple of 4 specifies a 64-bit length with the first and third operands in bit positions 0-63 of general registers.
- A function code that is 3 more than a multiple of 4 specifies a 128-bit length with the first and third operands in a parameter list.

Figure 7-290 on page 7-271 shows the function codes, operation names, and operand lengths, and also symbols that may be used to refer to the operations in discussions. For example, PLO.DCS may be used to mean PERFORM LOCKED OPERATION with function code 8. In the symbols, the letter “G” indicates a 64-bit operand length, the letter “R” indi-

cates that some or all 64-bit operands are in general registers, and the letter “X” indicates a 128-bit operand length.

Function Code	Operation	Operand Length (Bits)	Function Symbol
0	Compare and load	32	CL
1	Same as 0	64	CLG
2	Same as 0	64	CLGR
3	Same as 0	128	CLX
4	Compare and swap	32	CS
5	Same as 4	64	CSG
6	Same as 4	64	CSGR
7	Same as 4	128	CSX
8	Double compare and swap	32	DCS
9	Same as 8	64	DCSG
10	Same as 8	64	DCSGR
11	Same as 8	128	DCSX
12	Compare and swap and store	32	CSST
13	Same as 12	64	CSSTG
14	Same as 12	64	CSSTGR
15	Same as 12	128	CSSTX
16	Compare and swap and double store	32	CSDST
17	Same as 16	64	CSDSTG
18	Same as 16	64	CSDSTGR
19	Same as 16	128	CSDSTX
20	Compare and swap and triple store	32	CSTST
21	Same as 20	64	CSTSTG
22	Same as 20	64	CSTSTGR
23	Same as 20	128	CSTSTX

Figure 7-290. PERFORM LOCKED OPERATION Function Codes and Operations

The CPU can perform all of the operations specified by the function codes listed in Figure 7-290 specifying operations that the CPU can perform are called valid. Function codes that have not been assigned to operations or that specify operations that the CPU cannot perform because the operations are not implemented (installed) are called invalid.

Bit 55 of general register 0 is the test bit (T). When bit 55 is zero, the function code in general register 0 must be valid; otherwise, a specification exception is recognized. When bit 55 is one, the condition code is set to 0 if the function code is valid or to 3 if the func-

tion code is invalid, and no other operation is performed.

Bits 32-54 of general register 0 must be all zeros; otherwise, a specification exception is recognized. When bit 55 of the register is one, this is the only exception that can be recognized. Bits 0-31 of general register 0 are ignored.

The lock to be used is represented by a program lock token (PLT) whose logical address is specified in

general register 1. In the 24-bit addressing mode, the PLT address is bits 40-63 of general register 1, and bits 0-39 of the register are ignored. In the 31-bit addressing mode, the PLT address is bits 33-63 of the register, and bits 0-32 of the register are ignored. In the 64-bit addressing mode, the PLT address is bits 0-63 of the register.

The contents of general registers 0 and 1 described above are shown in Figure 7-291.

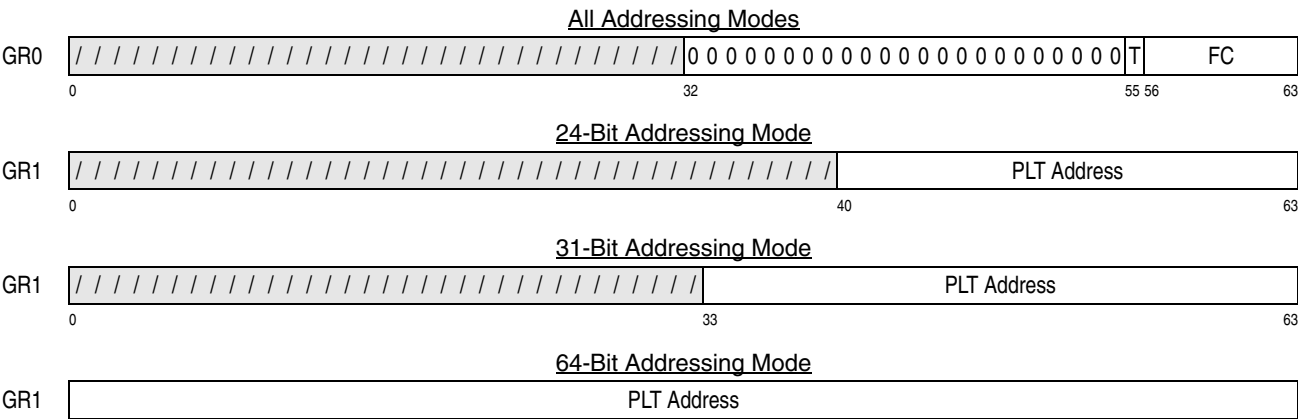


Figure 7-291. General Register Assignment for PLO

For the even-numbered function codes, including 0, the first-operand comparison value is in general register R_1 . For the even-numbered function codes beginning with 4, the first-operand replacement value is in general register $R_1 + 1$, and R_1 designates an even-odd pair of registers and must designate an even-numbered register; otherwise, a specification exception is recognized. For function codes 0 and 2, R_1 can be even or odd.

For function codes 0, 2, 12, and 14, the third operand is in general register R_3 , and R_3 can be even or odd.

For function codes 8 and 10, the third-operand comparison value is in general register R_3 , the third-operand replacement value is in general register $R_3 + 1$, and R_3 designates an even-odd pair of registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

For all function codes, the B_2 and D_2 fields of the instruction specify the second-operand address.

For function codes 0, 2, 8, 10, 12, and 14, the B_4 and D_4 fields of the instruction specify the fourth-operand address.

For function codes 1, 3, 5, 7, 9, 11, 13, 15, and 16-23, the B_4 and D_4 fields of the instruction specify the address of a parameter list that is used by the instruction, and this address is not called the fourth-operand address. The parameter list contains odd-numbered operands, including comparison and replacement values, and addresses of even-numbered operands other than the second operand. In the access-register mode, the parameter list also contains access-list-entry tokens (ALETs) associated with the even-numbered-operand addresses.

In the access-register mode, for function codes that cause use of a parameter list containing an ALET, R_3 must not be zero; otherwise, a specification exception is recognized.

The rules about R_1 and R_3 , and the use of the address specified by B_4 and D_4 , are summarized in Figure 7-292 on page 7-273.

Figure 7-293 on page 7-274 shows the locations of the operands (including operand comparison and replacement values), operand addresses, and parameter-list address used by the instruction.

Function Codes	Operation	R ₁	R ₃	D ₄ (B ₄)
0 and 2	Compare and load	EO	EO	Op4a
1 and 3	Compare and load	-	NZ	PLa
4 and 6	Compare and swap	E	-	-
5 and 7	Compare and swap	-	-	PLa
8 and 10	Double compare and swap	E	E	Op4a
9 and 11	Double compare and swap	-	NZ	PLa
12 and 14	Compare and swap and store	E	EO	Op4a
13 and 15	Compare and swap and store	-	NZ	PLa
16 and 18	Compare and swap and double store	E	NZ	PLa
17 and 19	Compare and swap and double store	-	NZ	PLa
20 and 22	Compare and swap and triple store	E	NZ	PLa
21 and 23	Compare and swap and triple store	-	NZ	PLa

Explanation:

- Ignored.

E Must be even.

EO Can be even or odd.

NZ Must be nonzero in the access-register mode. Ignored otherwise.

Op4a D4(B4) is operand-4 address.

PLa D4(B4) is parameter-list address.

Figure 7-292. Register Rules and D₄(B₄) Usage for PERFORM LOCKED OPERATION

Operand addresses in a parameter list, if used, are in doublewords in the list. In the 24-bit addressing mode, an operand address is bits 40-63 of a doubleword, and bits 0-39 of the doubleword are ignored. In the 31-bit addressing mode, an operand address is bits 33-63 of a doubleword, and bits 0-32 of the doubleword are ignored. In the 64-bit addressing mode, an operand address is bits 0-63 of a doubleword.

In the access-register mode, access register 1 specifies the address space containing the program lock token (PLT), access register B₂ specifies the address space containing the second operand, and access register B₄ specifies the address space containing a fourth operand or a parameter list as shown in Figure 7-293 on page 7-274. Also, for an operand whose address is in the parameter list, an access-

list-entry token (ALET) is in the list along with the address and is used in the access-register mode to specify the address space containing the operand.

In the access-register mode, if an access exception or PER storage-alteration event is recognized for an operand whose address is in the parameter list, the associated ALET in the parameter list is loaded into access register R₃ when the exception or event is recognized. Then, during the resulting program interruption, if a value is due to be stored as the exception access identification at real location 160 or the PER access identification at real location 161, R₃ is stored. If the instruction execution is completed without the recognition of an exception or event, the contents of access register R₃ are unpredictable. When not in the access-register mode, or when a parameter list containing an ALET is not used, the contents of access register R₃ remain unchanged.

The even-numbered (2, 4, 6, and 8) storage operands must be designated on an integral boundary, which is a word boundary for function codes that are a multiple of 4, a doubleword boundary for function codes that are one or 2 more than a multiple of 4, or a quadword boundary for function codes that are 3 more than a multiple of 4. A parameter list, if used, must be designated on a doubleword boundary. Otherwise, a specification exception is recognized. The program-lock-token (PLT) address in general register 1 does not have a boundary-alignment requirement.

All unused fields in a parameter list should contain all zeros; otherwise, the program may not operate compatibly in the future.

A serialization function is performed immediately after the lock is obtained and again immediately before it is released. However, values fetched from the parameter list before the lock is obtained are not necessarily refetched. A serialization function is not performed if the test bit, bit 55 of general register 0, is one.

In the following figures showing the parameter lists for the different function codes, the offsets shown on the left are byte values.

Function Codes 0-3 (Compare and Load)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-293 on page 7-274.

Function Codes ¹	Operation				Op3 or			Op5 and Op6a	Op7 and Op8a	
		Op1c	Op1r	Op2a	Op3c	Op3r	Op4a			PLa
0 and 2	Compare and load	R ₁	-	D ₂ (B ₂)		R ₃	D ₄ (B ₄)	-	-	-
1 and 3	Compare and load	PL	-	D ₂ (B ₂)	PL		PL	-	-	D ₄ (B ₄)
4 and 6	Compare and swap	R ₁	R ₁ +1	D ₂ (B ₂)	-		-	-	-	-
5 and 7	Compare and swap	PL	PL	D ₂ (B ₂)	-		-	-	-	D ₄ (B ₄)
8 and 10	Double compare and swap	R ₁	R ₁ +1	D ₂ (B ₂)	R ₃	R ₃ +1	D ₄ (B ₄)	-	-	-
9 and 11	Double compare and swap	PL	PL	D ₂ (B ₂)	PL	PL	PL	-	-	D ₄ (B ₄)
12 and 14	Compare and swap and store	R ₁	R ₁ +1	D ₂ (B ₂)		R ₃	D ₄ (B ₄)	-	-	-
13 and 15	Compare and swap and store	PL	PL	D ₂ (B ₂)	PL		PL	-	-	D ₄ (B ₄)
16 and 18	Compare and swap and double store	R ₁	R ₁ +1	D ₂ (B ₂)	PL		PL	PL	-	D ₄ (B ₄)
17 and 19	Compare and swap and double store	PL	PL	D ₂ (B ₂)	PL		PL	PL	-	D ₄ (B ₄)
20 and 22	Compare and swap and triple store	R ₁	R ₁ +1	D ₂ (B ₂)	PL		PL	PL	PL	D ₄ (B ₄)
21 and 23	Compare and swap and triple store	PL	PL	D ₂ (B ₂)	PL		PL	PL	PL	D ₄ (B ₄)
Explanation:										
¹	For function codes that are a multiple of 4 (including 0) or one more than a multiple of 4, operands in general registers are in bit positions 32-63 of the registers, and bits 0-31 of the registers are ignored and remain unchanged. For function codes that are two more than a multiple of 4, operands in general registers are in bit positions 0-63 of the registers.									
-	Operand, value, or address is not used in the operation.									
OpNc	Operand-N comparison value.									
OpNr	Operand-N replacement value.									
OpNa	Operand-N address.									
PL	Operand, value, or address is in the parameter list.									
PLa	Parameter-list address.									

Figure 7-293. Operand and Address Locations for PERFORM LOCKED OPERATION

The parameter list used for function code 1 has the following format:

Parameter List for Function Code 1	
0	
8	Operand-1 Comparison Value
16	
24	
32	
40	Operand 3
48	
56	
64	Operand-4 ALET
72	Operand-4 Address

The parameter list used for function code 3 has the following format:

Parameter List for Function Code 3	
0	Operand-1 Comparison Value
8	Operand-1 Comparison Value (continued)
16	
24	
32	Operand 3
40	Operand 3 (continued)
48	
56	
64	Operand-4 ALET
72	Operand-4 Address

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the third operand is replaced by the fourth operand, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand com-

parison value is replaced by the second operand, and condition code 1 is set.

Function Codes 4-7 (Compare and Swap)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-293 on page 7-274.

The parameter list used for function code 5 has the following format:

Parameter List for Function Code 5	
0	
8	Operand-1 Comparison Value
16	
24	Operand-1 Replacement Value

The parameter list used for function code 7 has the following format:

Parameter List for Function Code 7	
0	Operand-1 Comparison Value
8	Operand-1 Comparison Value (continued)
16	Operand-1 Replacement Value
24	Operand-1 Replacement Value (continued)

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the first-operand replacement value is stored at the second-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

Function Codes 8-11 (Double Compare and Swap)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-293 on page 7-274.

The parameter list used for function code 9 has the following format:

Parameter List for Function Code 9	
0	
8	Operand-1 Comparison Value
16	
24	Operand-1 Replacement Value
32	
40	Operand-3 Comparison Value
48	
56	Operand-3 Replacement Value
64	Operand-4 ALET
72	Operand-4 Address

The parameter list used for function code 11 has the following format:

Parameter List for Function Code 11	
0	Operand-1 Comparison Value
8	Operand-1 Comparison Value (continued)
16	Operand-1 Replacement Value
24	Operand-1 Replacement Value (continued)
32	Operand-3 Comparison Value
40	Operand-3 Comparison Value (continued)
48	Operand-3 Replacement Value
56	Operand-3 Replacement Value (continued)
64	Operand-4 ALET
72	Operand-4 Address

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the third-operand comparison value is compared to the fourth operand. When the third-operand comparison value is equal to the fourth operand (after the first-operand comparison value has been found equal to the second operand), the first-operand replacement value is stored at the second-operand location, the third-operand replacement value is stored at the fourth-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

When the third-operand comparison value is not equal to the fourth operand (after the first-operand comparison value has been found equal to the second operand), the third-operand comparison value is replaced by the fourth operand, and condition code 2 is set.

Function Codes 12-15 (Compare and Swap and Store)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-293 on page 7-274.

The parameter list used for function code 13 has the following format:

Parameter List for Function Code 13	
0	
8	Operand-1 Comparison Value
16	
24	Operand-1 Replacement Value
32	
40	
48	
56	Operand 3
64	Operand-4 ALET
72	Operand-4 Address

The parameter list used for function code 15 has the following format:

Parameter List for Function Code 15	
0	Operand-1 Comparison Value
8	Operand-1 Comparison Value (continued)
16	Operand-1 Replacement Value
24	Operand-1 Replacement Value (continued)
32	
40	
48	Operand 3
56	Operand 3 (continued)
64	Operand-4 ALET
72	Operand-4 Address

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the first-operand replacement value is stored at the second-

operand location, the third operand is stored at the fourth-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

Function Codes 16-19 (Compare and Swap and Double Store)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-293 on page 7-274.

The parameter list used for function code 16 has the following format:

Parameter List for Function Code 16	
0	
8	
16	
24	
32	
40	
48	
56	Operand 3
64	Operand-4 ALET
72	Operand-4 Address
80	
88	Operand 5
96	Operand-6 ALET
104	Operand-6 Address

The parameter list used for function code 17 has the following format:

Parameter List for Function Code 17	
0	
8	Operand-1 Comparison Value
16	
24	Operand-1 Replacement Value
32	
40	
48	
56	Operand 3
64	Operand-4 ALET
72	Operand-4 Address
80	
88	Operand 5
96	Operand-6 ALET
104	Operand-6 Address

The parameter list used for function code 18 has the following format:

Parameter List for Function Code 18	
0	
8	
16	
24	
32	
40	
48	
56	Operand 3
64	Operand-4 ALET
72	Operand-4 Address
80	
88	Operand 5
96	Operand-6 ALET
104	Operand-6 Address

The parameter list used for function code 19 has the following format:

Parameter List for Function Code 19	
0	Operand-1 Comparison Value
8	Operand-1 Comparison Value (continued)
16	Operand-1 Replacement Value
24	Operand-1 Replacement Value (continued)
32	
40	
48	Operand 3
56	Operand 3 (continued)
64	Operand-4 ALET
72	Operand-4 Address
80	Operand 5
88	Operand 5 (continued)
96	Operand-6 ALET
104	Operand-6 Address

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the first-operand replacement value is stored at the second-operand location, the third operand is stored at the fourth-operand location, the fifth operand is stored at the sixth-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

Function Codes 20-23 (Compare and Swap and Triple Store)

The locations of the operands and addresses used by the instruction are as shown in Figure 7-293 on page 7-274.

The parameter list used for function code 20 has the following format:

Parameter List for Function Code 20	
0	
8	
16	
24	
32	
40	
48	
56	Operand 3
64	Operand-4 ALET
72	Operand-4 Address
80	
88	Operand 5
96	Operand-6 ALET
104	Operand-6 Address
112	
120	Operand 7
128	Operand-8 ALET
136	Operand-8 Address

The parameter list used for function code 21 has the following format:

Parameter List for Function Code 21	
0	
8	Operand-1 Comparison Value
16	
24	Operand-1 Replacement Value
32	
40	
48	
56	Operand 3
64	Operand-4 ALET
72	Operand-4 Address
80	
88	Operand 5
96	Operand-6 ALET
104	Operand-6 Address
112	
120	Operand 7
128	Operand-8 ALET
136	Operand-8 Address

The parameter list used for function code 22 has the following format:

Parameter List for Function Code 22	
0	
8	
16	
24	
32	
40	
48	
56	Operand 3
64	Operand-4 ALET
72	Operand-4 Address
80	
88	Operand 5
96	Operand-6 ALET
104	Operand-6 Address
112	
120	Operand 7
128	Operand-8 ALET
136	Operand-8 Address

The parameter list used for function code 23 has the following format:

Parameter List for Function Code 23	
0	Operand-1 Comparison Value
8	Operand-1 Comparison Value (continued)
16	Operand-1 Replacement Value
24	Operand-1 Replacement Value (continued)
32	
40	
48	Operand 3
56	Operand 3 (continued)
64	Operand-4 ALET
72	Operand-4 Address
80	Operand 5
88	Operand 5 (continued)
96	Operand-6 ALET
104	Operand-6 Address
112	Operand 7
120	Operand 7 (continued)
128	Operand-8 ALET
136	Operand-8 Address

The first-operand comparison value is compared to the second operand. When the first-operand comparison value is equal to the second operand, the first-operand replacement value is stored at the second-operand location, the third operand is stored at the fourth-operand location, the fifth operand is stored at the sixth-operand location, the seventh operand is stored at the eighth-operand location, and condition code 0 is set.

When the first-operand comparison value is not equal to the second operand, the first-operand comparison value is replaced by the second operand, and condition code 1 is set.

Locking

A lock is obtained at the beginning of the operation and released at the end of the operation. The lock obtained is represented by a program lock token (PLT) whose logical address is specified in general register 1 as already described.

A PLT is a value produced by a model-dependent transformation of the PLT logical address. Depending on the model, the PLT may be derived directly from

the PLT logical address or, when DAT is on, from the real address that results from transformation of the PLT logical address by DAT. If DAT is used, access-register translation (ART) precedes DAT in the access-register mode.

A PLT selects one of a model-dependent number of locks within the configuration. Programs being executed by different CPUs can be assured of specifying the same lock only by specifying PLT logical addresses that are the same and that can be transformed to the same real address by the different CPUs.

Since a model may or may not use ART and DAT when forming a PLT, access-exception conditions that can be encountered during ART and DAT may or may not be recognized as exceptions. There is no accessing of a location designated by a PLT, but an addressing exception may be recognized for the location. A protection exception is not recognized for any reason during processing of a PLT logical address.

The CPU can hold one lock at a time.

When PERFORM LOCKED OPERATION is executed by this CPU and is to use a lock that is already held by another CPU due to the execution of a PERFORM LOCKED OPERATION instruction by the other CPU, the execution by this CPU is delayed until the lock is no longer held. An excessive delay can be caused only by a machine malfunction and is a machine-check condition.

The order in which multiple requests for the same lock are satisfied is undefined.

A nonrecoverable failure of a CPU while holding a lock may result in a machine check, entry into the check-stop state, or system check stop. The machine check is processing backup if all operands are undamaged or processing damage if register operands are damaged. If a machine check or the check-stop state is the result, either no storage operands have been changed or else all storage operands that were due to be changed have been correctly changed, and, in either case, the lock has been released. If the storage operands are not in either their correct original state or their correct final state, the result is system check stop.

Storage-Operand References

The accesses to the even-numbered storage operands appear to be word concurrent, as observed by other CPUs, for function codes that are a multiple of 4 or doubleword concurrent for function codes that are one, 2, or 3 more than a multiple of 4. The accesses to the doublewords in the parameter list appear to be doubleword concurrent, as observed by other CPUs, regardless of the function code.

As observed by other CPUs, all storage operands may be tested for access exceptions before a lock is obtained. (A channel program cannot observe a lock.)

As observed by other CPUs, in all operations except the compare-and-swap operation (which does not have a fourth operand), the fourth operand is accessed while the lock is held only if a comparison of the first-operand comparison value to the second operand while the lock is held has indicated equality. In these operations, the fourth operand is accessed before the lock is held only if a comparison of the first-operand comparison value to the second operand has indicated equality and only if, when DAT is on, an INVALIDATE PAGE TABLE ENTRY instruction executed by another CPU after the fetch of the second operand will not be the cause of a page-translation exception recognized for the fourth operand, which it will if it sets to one the page-invalid bit in the page-table entry for the fourth operand when this CPU does not have a TLB entry corresponding to that page-table entry. In the compare-and-swap-and-double-store and compare-and-swap-and-triple-store operations, the sixth operand, and also the eighth operand in the triple-store operation, are treated the same as the fourth operand described above. The reason for this specification about INVALIDATE PAGE TABLE ENTRY is given in programming note 6.

Provided that accessing of an operand is not prohibited as described in the preceding paragraph, store-type access exceptions may be recognized for the operand even when a store does not occur because of the results of a comparison. A storage-alteration PER event is recognized, and a change bit is set, only if a store occurs.

When a comparison is made between an operand comparison value and an operand before the lock is obtained and indicates inequality, the lock still is obtained. The condition code is set only as a result of a comparison made while the lock is held. When condition code 1 or 2 is set, the first-operand comparison

value or third-operand comparison value is replaced only by means of a fetch of the second operand or fourth operand, respectively, made while the lock is held, as observed by other CPUs.

In those cases when a store is performed to the second-operand location and one or more of the fourth-, sixth-, and eighth-operand locations, the store to the second-operand location is always performed last, as observed by other CPUs and by channel programs.

Stores into the parameter list may be performed while the lock is held or after it has been released.

Access exceptions may be recognized for parameter-list locations even when the locations are not required in the operation. The locations are those beginning at offset 0 and extending up through the last location defined for the function code used.

For the compare-and-load and compare-and-swap operations, the operation is suppressed on all addressing and protection exceptions.

When a nonrecoverable failure of a CPU while holding a lock results in a machine check or entry into the check-stop state, either no storage operands have been changed or else all storage operands that were due to be changed have been correctly changed. The latter may be accomplished by repeating stores that were performed successfully before the failure. Therefore, there may be two single-access store references (possibly the store part of an update reference and then a store reference) to the store-type operands, with the first value stored equal to the second value stored.

Resulting Condition Code:

When test bit is zero:

- 0 All comparisons equal; replacement value or values stored or loaded
- 1 First-operand comparison not equal; first-operand and comparison value replaced
- 2 -- (all operations except double compare and swap)
- 2 First-operand comparison equal but third-operand and comparison not equal; third-operand comparison value replaced (double compare and swap)
- 3 --

When test bit is one:

- 0 Function code valid
- 1 --
- 2 --
- 3 Function code invalid

Program Exceptions:

- Access (for all function codes, fetch, except addressing and protection for PLT location, program lock token, model-dependent; for all function codes, fetch and store, operand 2; for odd-numbered function codes, fetch and store, parameter list; for function codes 16, 18, 20, and 22, fetch, parameter list; for function codes 0-3, fetch, operand 4; for function codes 8-11, fetch and store, operand 4; for function codes 12-23, store, operand 4; for function codes 16-23, store, operand 6; for function codes 20-23, store, operand 8)
- Specification

Programming Notes:

1. An example of the use of the PERFORM LOCKED OPERATION instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When the contents of storage locations are changed by PERFORM LOCKED OPERATION instructions that are executed concurrently by different CPUs and that use the same lock, the changes to operands not in the parameter list will be completed by one of the CPUs before they are begun by the other CPU, depending on which CPU first obtains the lock.
3. The compare-and-swap functions of PERFORM LOCKED OPERATION are not performed by means of interlocked-update references. Concurrent store references by another CPU to the storage operands, even if they are interlocked-update references, will interfere unpredictably, in terms of the resulting register and storage contents, with the intended operation of PERFORM LOCKED OPERATION. All changes to the contents of the storage locations must be made by PERFORM LOCKED OPERATION instructions that use the same lock, if predictable storage results are to be obtained.
4. Because a nonrecoverable failure of a CPU while executing PERFORM LOCKED OPERATION may cause two stores of the same value to a store-type operand, a concurrent store made by another CPU to the same operand but not by executing PERFORM LOCKED OPERATION may be lost.
5. When programs in different address spaces are using the same lock when DAT is on, the programs must ensure that they are using PLT logical addresses that are the same and that will be translated to the same real address regardless of the address space in which a translation occurs. Otherwise, the programs may in fact use different locks.
6. The section "Storage-Operand References" on page 7-280 contains a specification concerning the INVALIDATE PAGE TABLE ENTRY (IPTE) instruction. The need for the specification is shown by the following example that is possible without the specification.
 - a. CPU 1 begins to execute a PERFORM LOCKED OPERATION instruction with function code 8, which is referred to as PLO.DCS. Operand 2 is a location, Qtail, containing the address (the first-operand comparison value) of the last element, element X, on a queue, and operand 4 is a location in that element containing the address (0, the third-operand comparison value) of the next (nonexisting) element on the queue. The purpose of the PLO instruction is to enqueue an element by placing the address of the element (the first-operand and third-operand replacement values) in both operand 2 and operand 4. With the lock not held, the PLO instruction fetches operand 2 and compares it, with an equal result, to the first-operand comparison value.
 - b. CPU 2 completely executes a PLO.DCS instruction to dequeue element X, which is the only element on the queue, from the queue. The PLO instruction stores 0 in Qtail and also in Qhead, which is a location containing the address of the first element on the queue. The program on CPU 2 processes the dequeued element and then invokes the freemain service of the control program to deallocate the storage containing the element. The freemain service uses IPTE to set the page-invalid bit to one in the page-table entry for the page containing element X. The IPTE instruction immediately sets the page-invalid bit to one, and then it waits for the sig-

nal that all other CPUs have cleared their TLBs of entries corresponding to the page.

- c. CPU 1 attempts to fetch operand 4. CPU 1 does not have a TLB entry for the operand-4 page. CPU 1 signals CPU 2 that the CPU 2 IPTE instruction may proceed.
- d. CPU 2 completes its IPTE instruction. The program on CPU 2 sets a software bit in the page-table entry to one to indicate that the page has been freemained and that, therefore, a reference to the page should result in presentation by the control program of an addressing exception to the program making the reference.
- e. CPU 1 attempts to do DAT for operand 4 and sees that the page-invalid bit is one. CPU 1 performs a program interruption indicating a page-translation exception. The exception handler sees that the software bit indicating freemained is one, and it presents an

addressing exception to the CPU 1 program, which causes an abend of the program.

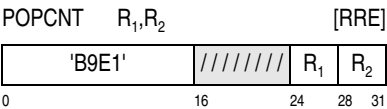
If CPU 1 had had a TLB entry for the page, its PLO instruction would not have been interrupted, and the comparison of the first-operand comparison value to the second operand while the lock was held would indicate that CPU 2 had changed the second operand. The PLO instruction would set condition code 1. If CPU 1 did not have a TLB entry but IPTE could not set the page-invalid bit to one while CPU 1 was executing an instruction, CPU 1 could successfully translate the operand-4 address and, again, discover while the lock was held that operand 2 had changed. The case when operand 2 points to element X but the freemained bit for the element-X page is one is a programming error.

7. "Summary of PERFORM LOCKED OPERATION Results" on page 7-284 summarizes the results of the operation.

Op1c=Op2	Op3c=Op4	Cond Code	Action
Function Codes 0-3 (Compare and Load)			
No	-	1	Op2 → Op1c
Yes	-	0	Op4 → Op3
Function Codes 4-7 (Compare and Swap)			
No	-	1	Op2 → Op1c
Yes	-	0	Op1r → Op2
Function Codes 8-11 (Double Compare and Swap)			
No	-	1	Op2 → Op1c
Yes	No	2	Op4 → Op3c
Yes	Yes	0	Op1r → Op2 Op3r → Op4
Function Codes 12-15 (Compare and Swap and Store)			
No	-	1	Op2 → Op1c
Yes	-	0	Op1r → Op2 Op3 → Op4
Function Codes 16-19 (Compare and Swap and Double Store)			
No	-	1	Op2 → Op1c
Yes	-	0	Op1r → Op2 Op3 → Op4 Op5 → Op6
Function Codes 20-23 (Compare and Swap and Triple Store)			
No	-	1	Op2 → Op1c
Yes	-	0	Op1r → Op2 Op3 → Op4 Op5 → Op6 Op7 → Op8
Explanation: - Not applicable. OpNc Operand-N comparison value. OpNr Operand-N replacement value.			

Figure 7-294. Summary of PERFORM LOCKED OPERATION Results

POPULATION COUNT



A count of the number of one bits in each of the eight bytes of general register R₂ is placed into the corresponding byte of general register R₁. Each byte of general register R₁ is an 8-bit binary integer in the range of 0-8.

Resulting Condition Code:

- 0 Result zero
- 1 Result not zero
- 2 --
- 3 --

Program Exceptions:

- Operation (if the population-count facility is not installed)

Programming Notes:

1. The condition code is set based on all 64 bits of general register R₁.
2. The total number of one bits in a general register can be computed as shown below. In this example, general register 15 contains the bits to be counted; the result containing the total number of one bits in general register 15 is placed in general register 8. (General register 9 is used as a work register and contains residual values on completion.)

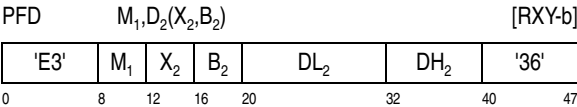
```
POPCNT 8,15
AHHLR  8,8,8
```

SLLG	9,8,16
ALGR	8,9
SLLG	9,8,8
ALGR	8,9
SRLG	8,8,56

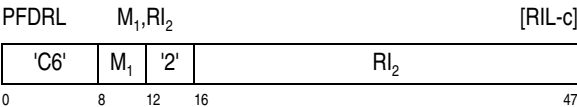
If there is a high probability that the results of the POPCNT instruction are zero, a conditional branch instruction may be inserted to skip the adding and shifting operations based on the condition code set by POPCNT.

- Using techniques similar to that shown in programming note 2, the number of one bits in a word, halfword, or discontinuous bytes of the second operand may be determined.

PREFETCH DATA



PREFETCH DATA RELATIVE LONG



Note: In this instruction definition, the name “PREFETCH DATA (RELATIVE LONG)” refers to both the PREFETCH DATA instruction and the PREFETCH DATA RELATIVE LONG instruction.

Subject to the controls specified in the M_1 field, the CPU is signalled to prefetch the second operand into a cache line, or it is signalled to release a cache line containing the second operand. The second operand designates a logical address.

The M_1 field contains a 4-bit unsigned binary integer that is used as a code to signal the CPU as to the

intended use of the second operand. The codes are as follows:

Code	Function Performed
1	Prefetch the data at the second-operand address for fetch access.
2	Prefetch the data at the second-operand address for store access.
6	Release the cache line containing the second operand from store access; retain the data in the cache line for fetch access.
7	Release the cache line containing the second operand from all accesses

All other codes are reserved. Depending on the model, the CPU may not implement all of the prefetch functions. For functions that are not implemented by the CPU, and for reserved functions, the instruction acts as a no-op. Code 0 always acts as a no-op.

No access exceptions or PER storage-alteration events are recognized for the second operand.

For code 2, it is unpredictable whether the change bit is set for the second operand. For all codes, it is model dependent whether a TLB entry is formed for the data that is prefetched.

For PREFETCH DATA, the displacement is treated as a 20-bit signed binary integer.

For PREFETCH DATA RELATIVE LONG, the contents of the I_2 field are signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

Condition Code: The code remains unchanged.

Program Exceptions:

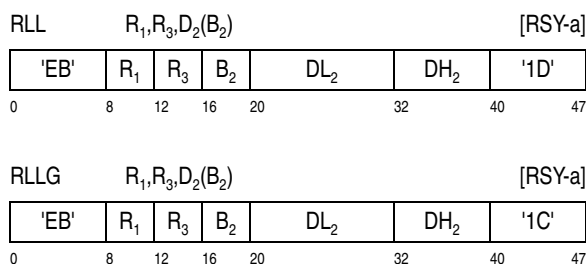
- Operation (if the general-instructions-extension facility is not installed)

Programming Notes:

1. PREFETCH DATA (RELATIVE LONG) signals the CPU to perform the specified operation, but it does not guarantee that the CPU will necessarily honor the request.
2. There is no guarantee that storage location will still be in the cache when a subsequent instruction references the location. Likewise, there is no guarantee that when a cache line is released, that the CPU will not subsequently refetch it (independent of any prefetching operations). Rather, PREFETCH DATA (RELATIVE LONG) simply provides hints as to the program's anticipated use of storage areas.
3. If an exception condition would otherwise be recognized when accessing the second operand, PREFETCH DATA (RELATIVE LONG) is completed with no indication of the exception provided to the program, however, the performance of PREFETCH DATA (RELATIVE LONG) may be significantly slower than if the exception condition did not exist.
4. A significant delay may be experienced if a storage location has been prefetched and then released, and then a subsequent instruction references the same storage location. Similarly, a delay may be experienced if a storage location has been prefetched for fetch access, and then a subsequent instruction references the same location for storing.
5. On models that implement a unified data and instruction cache, the function performed may affect both subsequent operand accesses and instruction fetches from the second-operand location. On such models, a significant delay may be experienced if the code 7 is used to release data in a cache line from which an instruction is subsequently fetched.
6. On models that implement separate data and instruction caches, codes 1 and 2 cause the second operand to be prefetched into the data cache. Similarly, on such models, codes 6 and 7 cause the second operand to be released from the data cache. However, on certain models with separate data and instruction caches, code 7 may also cause the second operand to be released from the instruction cache, in addition to being released from the data cache.

7. On models that implement separate data and instruction caches, the use of code 2 to prefetch (for storing) a cache line from which instructions will subsequently be fetched may cause significant delays. Similar delays may be experienced for any store operation into a cache line from which instructions are subsequently fetched.
8. The use of PREFETCH DATA (RELATIVE LONG) to prefetch operands that are frequently updated in a multiprocessing environment may actually degrade performance by causing unnecessary contention for the cache line.
9. A prefetch operation consists of fetching a cache line on an integral boundary. The cache line size (and corresponding integral boundary) may be determined by executing EXTRACT CACHE ATTRIBUTE.
10. The second operand is fetched into the cache line in model-dependent units, on an integral boundary, the minimum size of which is a double-word. Thus, at least the rightmost three bits of the second-operand address are assumed to contain zeros, regardless of what is specified by the program.
11. On most models, the unit directly addressed by the second-operand address is prefetched first. The order in which the remaining units of the cache line are prefetched is also model dependent.
12. When PREFETCH DATA RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.

ROTATE LEFT SINGLE LOGICAL



The 32-bit or 64-bit third operand is rotated left the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Except for when the R_1 and R_3 fields design-

- 0 Result is zero
- 1 Result is not zero
- 2 --
- 3 --

Program Exceptions:

- Operation (if the general-instructions-extension facility is not installed)

Programming Notes:

1. Although the bits 2-7 of the I₅ field are defined to contain an unsigned binary integer specifying the number of bits that the second operand is rotated to the left, a negative value may be coded which effectively specifies a rotate-right amount.
2. The first operand is always used in its unrotated form. When the R₁ and R₂ fields designate the same register, the contents of the register are first rotated, and then the selected bits of the rotated value logically operate upon the corresponding bits of the unrotated register contents.
3. Examples of the use of ROTATE THEN EXCLUSIVE OR SELECTED BITS, and ROTATE THEN OR SELECTED BITS are given in “Number Representation and Instruction-Use Examples” on page A-1.
4. In the assembler syntax, the I₅ operand containing the rotate amount is considered to be

optional. When the I₅ field is not coded, a rotate amount of zero is implied.

5. The I₃ field contains both the test-results control (in bit 0) and the starting bit position value (in bits 2-7). For example, to AND bits 40-43 of register 5 with the corresponding bits of register 7 (no rotation) and simply test the results, the programmer might code:

```
RNSBG R5,R7,X'80'+40,43,0
```

The X'80' represents the test-results control which is added to the starting-bit position to form the I₃ field.

The high-level assembler (HLASM) provides alternative mnemonics for the test versions of these instructions, as shown below:

```
RNSBGT R5,R7,40,43,0
```

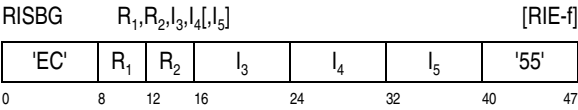
The “T” suffix to the mnemonic indicates that the specified I₃ field is ORed with a value of X'80' when generating the object code. The T mnemonic suffix also applies to ROSBG and RXSBG.

6. The High Level Assembler implements various high-word logical operations by providing extended-mnemonics for the RNSBG, ROSBG, and RXSBG instructions, as shown in Figure 7-295.

Instruction Name	Extended-Mnemonic Syntax	RxSBG Equivalent
AND HIGH (HIGH ← HIGH)	NHHR R ₁ ,R ₂	RNSBG R ₁ ,R ₂ ,0,31
AND HIGH (HIGH ← LOW)	NHLR R ₁ ,R ₂	RNSBG R ₁ ,R ₂ ,0,31,32
AND HIGH (LOW ← HIGH)	NLHR R ₁ ,R ₂	RNSBG R ₁ ,R ₂ ,32,63,32
EXCLUSIVE OR HIGH (HIGH ← HIGH)	XHHR R ₁ ,R ₂	RXSBG R ₁ ,R ₂ ,0,31
EXCLUSIVE OR HIGH (HIGH ← LOW)	XHLR R ₁ ,R ₂	RXSBG R ₁ ,R ₂ ,0,31,32
EXCLUSIVE OR HIGH (LOW ← HIGH)	XLHR R ₁ ,R ₂	RXSBG R ₁ ,R ₂ ,32,63,32
OR HIGH (HIGH ← HIGH)	OHHR R ₁ ,R ₂	ROSBG R ₁ ,R ₂ ,0,31
OR HIGH (HIGH ← LOW)	OHLR R ₁ ,R ₂	ROSBG R ₁ ,R ₂ ,0,31,32
OR HIGH (LOW ← HIGH)	OLHR R ₁ ,R ₂	ROSBG R ₁ ,R ₂ ,32,63,32

Figure 7-295. Extended-Mnemonics Formed Using RNSBG, RXSBG, and ROSBG

ROTATE THEN INSERT SELECTED BITS



The second operand is rotated left by the number of bits specified in the fifth operand. Each bit shifted out of the leftmost bit position of the operand reenters in the rightmost bit position of the operand. The selected bits of the rotated second operand replace the contents of the corresponding bit positions of the

first operand. The result is indicated by the condition code.

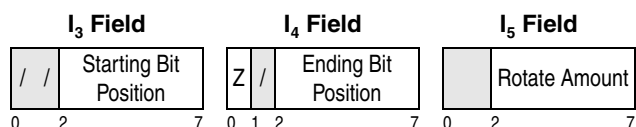
The second operand remains unchanged in general register R_2 .

Bits 2-7 of the I_3 field (bits 18-23 of the instruction) contain an unsigned binary integer specifying the starting bit position of the selected range of bits in the first operand and in the second operand after rotation. Bits 2-7 of the I_4 field (bits 26-31 of the instruction) contain an unsigned binary integer specifying the ending bit position (inclusive) of the selected range of bits. When the ending bit position is less than the starting bit position, the range of bits wraps around from bit 63 to bit 0.

Bits 2-7 of the I_5 field (bits 34-39 of the instruction) contain an unsigned binary integer specifying the number of bits that the second operand is rotated to the left.

Bit 0 of the I_4 field (bit 24 of the instruction) contains the zero-remaining-bits control (Z). The Z bit controls how the remaining bits of the first operand are set (that is, those bits, if any, that are outside of the specified range). When the Z bit is zero, the remaining bits of the first operand are unchanged. When the Z bit is one, the remaining bits of the first operand are set to zeros.

The immediate fields just described are as follows:



Bits 0-1 of the I_3 field and bit 1 of the I_4 field (bits 16-17 and 25 of the instruction) are reserved and should contain zeros; otherwise, the program may not operate compatibly in the future. Bits 0-1 of the I_5 field (bits 32-33 of the instruction) are ignored.

For the purposes of setting the condition code, the result in general register R_1 is treated as being a 64-bit signed binary integer.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero

3 --

Program Exceptions:

- Operation (if the general-instructions-extension facility is not installed)

Programming Notes:

1. Although the bits 2-7 of the I_5 field are defined to contain an unsigned binary integer specifying the number of bits that the second operand is rotated to the left, a negative value may be coded which effectively specifies a rotate-right amount.
2. When used with the zero-remaining-bits control, ROTATE THEN INSERT SELECTED BITS provides a means of effecting a shift operation.
3. When the R_1 and R_2 fields designate the same register and the Z bit is one, ROTATE THEN INSERT SELECTED BITS may be used to zero a selected range of bits in a register. Note, this technique cannot be used to zero all 64 bits of the register.
4. The first operand is always used in its unrotated form. When the R_1 and R_2 fields designate the same register, the contents of the register are first rotated, and then the selected bits of the rotated value are inserted into the corresponding bits of the unrotated register contents.
5. Examples of the use of ROTATE THEN INSERT SELECTED BITS are given in "Number Representation and Instruction-Use Examples" on page A-1.
6. In the assembler syntax, the I_5 operand containing the rotate amount is considered to be optional. When the I_5 field is not coded, a rotate amount of zero is implied.
7. The I_4 field contains both the zero-remaining-bits control (in bit 0) and the ending bit position value (in bits 2-7). For example, to insert bits 40-43 of register 7 into the corresponding bits of register 5 (no rotation) and zero the remaining bits in register 5, the programmer might code:

```
RISBG R5,R7,40,X'80'+43,0
```

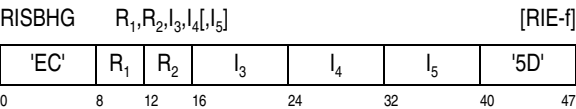
The X'80' represents the zero-remaining-bits control which is added to the ending-bit position to form the I_4 field.

The high-level assembler (HLASM) provides an alternative mnemonic for the zero-remaining-bits version of the instruction, as shown below:

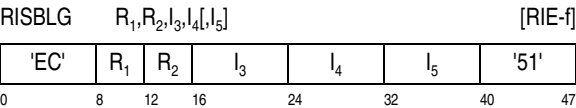
```
RISBGZ R5,R7,40,43,0
```

The “Z” suffix to the mnemonic indicates that the specified I₄ field is ORed with a value of X’80’ when generating the object code.

ROTATE THEN INSERT SELECTED BITS HIGH



ROTATE THEN INSERT SELECTED BITS LOW



The 64-bit second operand is rotated left by the number of bits specified in the fifth operand. Each bit shifted out of the leftmost bit position of the operand reenters in the rightmost bit position of the operand. The selected bits of the rotated second operand replace the contents of the corresponding bit positions of the first operand.

For ROTATE THEN INSERT SELECTED BITS HIGH, the first operand is in bits 0-31 of general register R₁, and bits 32-63 of the register are unchanged. For ROTATE THEN INSERT SELECTED BITS LOW, the first operand is in bits 32-63 of general register R₁, and bits 0-31 of the register are unchanged.

The second operand remains unchanged in general register R₂.

For ROTATE THEN INSERT SELECTED BITS HIGH, bits 3-7 of the I₃ and I₄ fields (bits 19-23 and 27-31 of the instruction, respectively), with a binary zero appended on the left of each, form six-bit unsigned binary integers specifying the starting and ending bit positions (inclusive) of the selected range of bits in

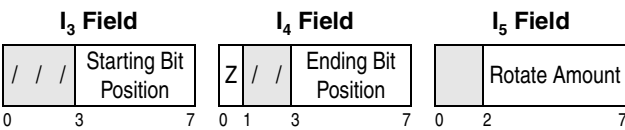
the first operand and in the second operand after rotation. When the ending bit position is less than the starting bit position, the range of selected bits wraps around from bit 31 to bit 0. Thus, the starting and ending bit positions of the selected range of bits are always between 0 and 31.

For ROTATE THEN INSERT SELECTED BITS LOW, bits 3-7 of the I₃ and I₄ fields, with a binary one appended on the left of each, form six-bit unsigned binary integers specifying the starting and ending bit positions (inclusive) of the selected range of bits in the first operand and in the second operand after rotation. When the ending bit position is less than the starting bit position, the range of selected bits wraps around from bit 63 to bit 32. Thus, the starting and ending bit positions of the selected range of bits are always between 32 and 63.

Bits 2-7 of the I₅ field (bits 34-39 of the instruction) contain an unsigned binary integer specifying the number of bits that the second operand is rotated to the left.

Bit 0 of the I₄ field (bit 24 of the instruction) contains the zero-remaining-bits control (Z). The Z bit controls how the remaining bits of the first operand are set (that is, those bits, if any, that are outside of the specified range). When the Z bit is zero, the remaining bits of the first operand are unchanged. When the Z bit is one, the remaining bits of the first operand are set to zeros.

The immediate fields just described are as follows:



Bits 0-2 of the I₃ field and bits 1-2 of the I₄ field (bits 16-19 and 25-26 of the instruction) are reserved and should contain zeros; otherwise, the program may not operate compatibly in the future. Bits 0-1 of the I₅ field (bits 32-33 of the instruction) are ignored.

Condition Code: The code remains unchanged

Program Exceptions:

- Operation (if the high-word facility is not installed)

Programming Notes:

1. Although the bits 2-7 of the I_5 field are defined to contain an unsigned binary integer specifying the number of bits that the second operand is rotated to the left, a negative value may be coded which effectively specifies a rotate-right amount.
2. The first operand is always used in its unrotated form. When the R_1 and R_2 fields designate the same register, the value contained in the register is first rotated, and then the selected bits of the rotated value are inserted into the corresponding bits of the unrotated register contents.
3. In the assembler syntax, the I_5 operand containing the rotate amount is considered to be optional. When the I_5 field is not coded, a rotate amount of zero is implied.
4. The I_4 field contains both the zero-remaining-bits control (in bit 0) and the ending bit position value (in bits 2-7). For example, to insert bits 40-43 of register 7 into the corresponding bits of register 5 (no rotation) and zero the remaining bits in the right half of register 5, the programmer might code:

```
RISBLG R5,R7,40,X'80'+43,0
```

The X'80' represents the zero-remaining-bits control which is added to the ending-bit position to form the I_4 field.

The high-level assembler (HLASM) provides alternative mnemonics for the zero-remaining-bits versions of RISBHG and RISBLG in the form of RISBHGZ and RISBLHZ, respectively. The "Z" suffix to the mnemonic indicates that the specified I_4 field is ORed with a value of X'80' when generating the object code.

An equivalent to the example shown above using the Z-suffixed mnemonic is as follows:

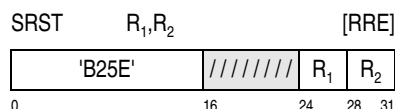
```
RISBLGZ R5,R7,40,43,0
```

5. On some models, improved performance of RISBHG and RISBLG may be realized by setting the zero-remaining-bits control to one (or using the Z mnemonic suffix).
6. Unlike ROTATE THEN INSERT SELECTED BITS which sets the condition code, ROTATE THEN INSERT SELECTED BITS HIGH and ROTATE THEN INSERT SELECTED BITS LOW do not set the condition code.
7. The High Level Assembler implements various high-word logical operations by providing extended-mnemonics for the RISBHG and RISBLG instructions, as shown in Figure 7-296.

Instruction Name	Extended-Mnemonic Syntax	RISBHG / RISBLG Equivalent
LOAD (HIGH ← HIGH)	LHHR R_1, R_2	RISBHGZ $R_1, R_2, 0, 31$
LOAD (HIGH ← LOW)	LHLR R_1, R_2	RISBHGZ $R_1, R_2, 0, 31, 32$
LOAD (LOW ← HIGH)	LLHFR R_1, R_2	RISBLGZ $R_1, R_2, 0, 31, 32$
LOAD LOGICAL HALFWORD (HIGH ← HIGH)	LLHHR R_1, R_2	RISBHGZ $R_1, R_2, 16, 31$
LOAD LOGICAL HALFWORD (HIGH ← LOW)	LLHLR R_1, R_2	RISBHGZ $R_1, R_2, 16, 31, 32$
LOAD LOGICAL HALFWORD (LOW ← HIGH)	LLLHR R_1, R_2	RISBLGZ $R_1, R_2, 16, 31, 32$
LOAD LOGICAL CHARACTER (HIGH ← HIGH)	LLCHHR R_1, R_2	RISBHGZ $R_1, R_2, 24, 31$
LOAD LOGICAL CHARACTER (HIGH ← LOW)	LLCHLR R_1, R_2	RISBHGZ $R_1, R_2, 24, 31, 32$
LOAD LOGICAL CHARACTER (LOW ← HIGH)	LLCLHR R_1, R_2	RISBLGZ $R_1, R_2, 24, 31, 32$

Figure 7-296. Extended Mnemonics Formed Using RISBHG and RISBLG

SEARCH STRING



The second operand is searched until a specified character is found, the end of the second operand is

reached, or a CPU-determined number of bytes have been searched, whichever occurs first. The CPU-determined number is at least 256. The result is indicated in the condition code.

The location of the first byte of the second operand is designated by the contents of general register R_2 . The location of the first byte after the second operand is designated by the contents of general register R_1 .

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

In the access-register mode, the address space containing the second operand is specified only by means of access register R_2 . The contents of access register R_1 are ignored.

The character for which the search occurs is specified in bit positions 56-63 of general register 0. Bit positions 32-55 of general register 0 are reserved for possible future extensions and must contain all zeros; otherwise, a specification exception is recognized.

The operation proceeds left to right and ends as soon as the specified character has been found in the second operand, the address of the next second-operand byte to be examined equals the address in general register R_1 , or a CPU-determined number of second-operand bytes have been examined, whichever occurs first. The CPU-determined number is at least 256. When the specified character is found, condition code 1 is set. When the address of the next second-operand byte to be examined equals the address in general register R_1 , condition code 2 is set. When a CPU-determined number of second-operand bytes have been examined, condition code 3 is set. When the CPU-determined number of second-operand bytes have been examined and the address of the next second-operand byte is in general register R_1 , it is unpredictable whether condition code 2 or 3 is set.

When condition code 1 is set, the address of the specified character found in the second operand is placed in general register R_1 , and the contents of general register R_2 remain unchanged. When condition code 3 is set, the address of the next byte to be processed in the second operand is placed in general register R_2 , and the contents of general register R_1 remain unchanged. When condition code 2 is set, the contents of general registers R_1 and R_2 remain unchanged. Whenever an address is placed in a general register, bits 32-39 of the register, in the 24-bit addressing mode, or bit 32, in the 31-bit addressing

mode, are set to zeros. Bits 0-31 of the R_1 and R_2 registers always remain unchanged in the 24-bit or 31-bit mode.

When the address in general register R_1 equals the address in general register R_2 , condition code 2 is set immediately, and access exceptions are not recognized. When the address in general register R_1 is less than the address in general register R_2 , condition code 2 can be set only if the operand wraps around from the top of storage to location 0.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

Access exceptions for the second operand are recognized only for that portion of the operand that is necessarily examined.

The storage-operand-consistency rules are described in the section "Storage-Operand Consistency" on page 5-95.

Resulting Condition Code:

- 0 --
- 1 Specified character found; general register R_1 updated with address of character; general register R_2 unchanged
- 2 Specified character not found in entire second operand; general registers R_1 and R_2 unchanged
- 3 CPU-determined number of bytes searched; general register R_1 unchanged; general register R_2 updated with address of next byte

Program Exceptions:

- Access (fetch, operand 2)
- Specification

Programming Notes:

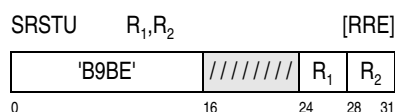
1. Examples of the use of the SEARCH STRING instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the search. The program need not determine the number of bytes that were searched.

3. The R_1 or R_2 fields may designate general register 0, in which case general register 0 is treated as containing an address and also the specified character.

However, if the program branches back to the SEARCH STRING instruction following condition code 3, specifying R_2 as register 0 is impractical. This is because bits 32-55 of the register will have been altered to a nonzero value, thus resulting in a specification exception on the subsequent execution. The search character in bit positions 56-63 of the register also may have been altered.

4. When it is desired to search a string of unknown length for its ending character, and assuming that the specified character in general register 0 need not be preserved, then the R_1 field can designate general register 0 in order to have SEARCH STRING use only two general registers instead of three. In this case, the rightmost portion of general register 0 containing the required zeros and the 8-bit search character is also used to form the address of the first byte after the second operand.
5. If the program branches back to the SEARCH STRING instruction when condition code 3 is set, and a subsequent execution results in condition code 1 or 2, general register R_2 will have changed from its initial value, even though the definition states that the register is unchanged when the character is found.
6. If the length of the string (as determined by the difference between the addresses in general registers R_1 and R_2) is 255 characters or less, condition code 3 will never be set, and branching on that condition is not necessary. However, if the length of the string is 256 or more characters, condition code 3 may be set – even if condition code 2 also applies.

SEARCH STRING UNICODE



The second operand is searched until a specified two-byte character is found, the end of the second operand is reached, or a CPU-determined number of

two-byte characters have been searched, whichever occurs first. The CPU-determined number is at least 256 two-byte characters. The result is indicated in the condition code.

The location of the first two-byte character of the second operand is designated by the contents of general register R_2 . When the contents of bit position 63 of general registers R_1 and R_2 are identical (that is, when both addresses are even or both addresses are odd), the location of the first two-byte character after the second operand is designated by the contents of general register R_1 . When the contents of bit position 63 of general registers R_1 and R_2 differ (that is, when one address is even and the other is odd), the location of the first two-byte character after the second operand is one more than the contents of general register R_1 .

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

In the access-register mode, the address space containing the second operand is specified only by means of access register R_2 . The contents of access register R_1 are ignored.

The two-byte character for which the search occurs is specified in bit positions 48-63 of general register 0. Bit positions 32-47 of general register 0 are reserved for possible future extensions and must contain all zeros; otherwise, a specification exception is recognized.

Except for the case when the address in general register R_1 equals the address in general register R_2 , the operation proceeds from left to right in steps of two bytes and ends as soon as the specified two-byte character has been found in the second operand, the address of the next second-operand two-byte character to be examined (excluding the first two-byte character) is equal to the address of the first two-byte character after the second operand (as designated by general register R_1), or a CPU-determined number of second-operand two-byte characters have been examined, whichever occurs first. The CPU-

determined number is at least 256. When the specified two-byte character is found, condition code 1 is set. When the address of the next second-operand two-byte character to be examined (excluding the first two-byte character) is equal to the address of the first two-byte character after the second operand, condition code 2 is set. When a CPU-determined number of second-operand two-byte characters have been examined, condition code 3 is set, except that condition code 2 is set if the conditions for setting it also apply.

When condition code 1 is set, the address of the specified two-byte character found in the second operand is placed in general register R_1 , and the contents of general register R_2 remain unchanged. When condition code 3 is set, the address of the next two-byte character to be processed in the second operand is placed in general register R_2 , and the contents of general register R_1 remain unchanged. When condition code 2 is set, the contents of general registers R_1 and R_2 remain unchanged. Whenever an address is placed in a general register, bits 32-39 of the register, in the 24-bit addressing mode, or bit 32, in the 31-bit addressing mode, are set to zeros. Bits 0-31 of the R_1 and R_2 registers always remain unchanged in the 24-bit or 31-bit mode.

When the address in general register R_1 equals the address in general register R_2 , condition code 2 is set immediately, and access exceptions are not recognized. When the address in general register R_1 is less than the address in general register R_2 , condition code 2 can be set only if the operand wraps around from the top of storage to location 0.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

Access exceptions for the second operand are recognized only for that portion of the operand that is necessarily examined.

The storage-operand-consistency rules are described in the section "Storage-Operand Consistency" on page 5-95.

Resulting Condition Code:

- 0 --
- 1 Specified two-byte character found; general register R_1 updated with address of character; general register R_2 unchanged
- 2 Specified two-byte character not found in entire second operand; general registers R_1 and R_2 unchanged
- 3 CPU-determined number of two-byte characters searched; general register R_1 unchanged; general register R_2 updated with address of next two-byte character

Program Exceptions:

- Access (fetch, operand 2)
- Operation (if the extended-translation facility 3 is not installed)
- Specification

Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the search. The program need not determine the number of two-byte characters that were searched.
2. The R_1 or R_2 fields may designate general register 0, in which case general register 0 is treated as containing an address and also the specified character.

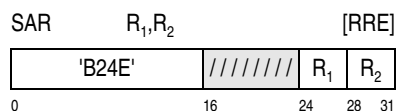
However, if the program branches back to the SEARCH STRING UNICODE instruction following condition code 3, specifying R_2 as register 0 is impractical. This is because bits 32-47 of the register may have been altered to a nonzero value, thus resulting in a specification exception on the subsequent execution. The search character in bit positions 48-63 of the register also may have been altered.

3. When it is desired to search a string of unknown length for its ending two-byte character, and assuming that the specified two-byte character in general register 0 need not be preserved, then the R_1 field can designate general register 0 in order to have SEARCH STRING UNICODE use only two general registers instead of three. In this case, the rightmost portion of general register 0 containing the required zeros and the 16-bit search character is also used to form the

address of the first byte after the second operand.

4. If the program branches back to the SEARCH STRING UNICODE instruction when condition code 3 is set, and a subsequent execution results in condition code 1 or 2, general register R_2 will have changed from its initial value, even though the definition states that the register is unchanged when the two-byte character is found.
5. If the length of the string (as determined by the difference between the addresses in general registers R_1 and R_2) is 256 two-byte characters or less, condition code 3 will never be set, and branching on that condition is not necessary. However, if the length of the string is 257 or more two-byte characters, condition code 3 may be set.

SET ACCESS

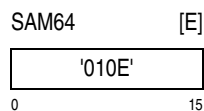
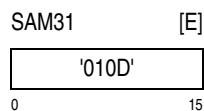
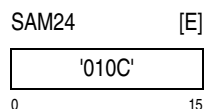


The contents of bit positions 32-63 of general register R_2 are placed in access register R_1 .

Condition Code: The code remains unchanged.

Program Exceptions: None.

SET ADDRESSING MODE



The addressing mode is set by setting the extended-addressing-mode bit, bit 31 of the current PSW, and

the basic-addressing-mode bit, bit 32 of the current PSW, as follows:

Instruction	PSW Bit 31	PSW Bit 32	Resulting Addressing Mode
SAM24	0	0	24-bit
SAM31	0	1	31-bit
SAM64	1	1	64-bit

The instruction address in the PSW is updated under the control of the new addressing mode, as follows. The value 2 (the instruction length) is added to the contents of bit positions 64-127 of the PSW, or the value 4 is added if the instruction is the target of EXECUTE, or the value 6 is added if the instruction is the target of EXECUTE RELATIVE LONG. In any case, a carry out of bit position 0 is ignored. Then bits 64-103 of the PSW are set to zeros if the new addressing mode is the 24-bit mode, or bits 64-96 are set to zeros if the new addressing mode is the 31-bit mode.

The instruction is completed only if the new addressing mode and the unupdated instruction address in the PSW are a valid combination. When the new addressing mode is to be the 24-bit mode, bits 64-103 of the unupdated PSW must be all zeros, or, when the new addressing mode is to be the 31-bit mode, bits 64-96 of the unupdated PSW must be all zeros; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

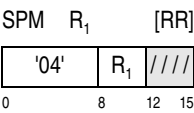
Program Exceptions:

- Specification (SAM24 and SAM31 only)
- Trace

Programming Note: Checking the unupdated instruction address prevents completion in two major cases: the instruction is located at address 2^{24} or above and the new addressing mode is to be the 24-bit mode, or the instruction is located at address 2^{31} or above and the new addressing mode is to be the 24-bit or 31-bit mode. In these cases, if the instruction were completed, the updating of the instruction address under the control of the new addressing mode would cause one or more leftmost bits of the address to be set to zeros, which would cause the next instruction to be fetched from other than the next sequential location. This action is sometimes called a “wild branch.” A wild branch still

can occur if the instruction is located at $2^{24} - 2$ or $2^{31} - 2$, or at $2^{24} - 4$ or $2^{31} - 4$ if EXECUTE or EXECUTE RELATIVE LONG is used, or at $2^{24} - 6$ or $2^{31} - 6$ if EXECUTE RELATIVE LONG is used.

SET PROGRAM MASK



The first operand is used to set the condition code and the program mask of the current PSW.

Bits 34 and 35 of general register R₁ replace the condition code, and bits 36-39 replace the program mask. Bits 0-33 and 40-63 of general register R₁ are ignored.

Resulting Condition Code:

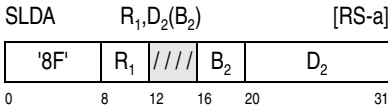
The code is set as specified by bits 34 and 35 of general register R₁.

Program Exceptions: None.

Programming Notes:

- 1. Bits 34-39 of the general register may have been loaded from the PSW by execution of BRANCH AND LINK in the 24-bit addressing mode or by execution of INSERT PROGRAM MASK in any addressing mode.
- 2. SET PROGRAM MASK permits setting of the condition code and the mask bits in either the problem state or the supervisor state.
- 3. The program should take into consideration that the setting of the program mask can have a significant effect on subsequent execution of the program. Not only do the four mask bits control whether the corresponding interruptions occur, but the HFP-exponent-underflow and HFP-significance masks also determine the result which is obtained.

SHIFT LEFT DOUBLE



The 63-bit numeric part of the signed first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location. The first operand consists of bits 32-63 of general register R₁ followed on the right by bits 32-63 of general register R₁ + 1.

The R₁ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 64-bit signed binary integer. The sign bit of the first operand, bit 32 of the even-numbered register, remains unchanged. Bit position 32 of the odd-numbered register contains a numeric bit, which participates in the shift in the same manner as the other numeric bits. Zeros are supplied to the vacated bit positions on the right. Bits 0-31 of general registers R₁ and R₁ + 1 remain unchanged.

If one or more bits unlike the sign bit are shifted out of bit position 33 of the even-numbered register, an overflow occurs, and condition code 3 is set. If the fixed-point-overflow mask bit is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

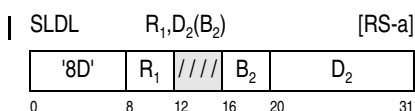
- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

- Fixed-point overflow
- Specification

Programming Notes:

1. An example of the use of the SHIFT LEFT DOUBLE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The eight shift instructions that are in both ESA/390 and z/Architecture provide the following three pairs of alternatives for 32 bits in one general register or, for double, in each of two general registers: left or right, single or double, and signed or logical. The four additional shift instructions in z/Architecture provide left or right, signed or logical shifts of 64 bits in one general register. The signed shifts differ from the logical shifts in that, in the signed shifts, overflow is recognized, the condition code is set, and the leftmost bit participates as a sign.
3. A zero shift amount in the two signed double-shift operations provides a double-length sign and magnitude test.
4. The base register participating in the generation of the second-operand address permits indirect specification of the shift amount by means of placement of the shift amount in the base register. A zero in the B_2 field indicates the absence of indirect shift specification.

SHIFT LEFT DOUBLE LOGICAL

The 64-bit first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location. The first operand consists of bits 32-63 of general register R_1 followed on the right by bits 32-63 of general register $R_1 + 1$.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

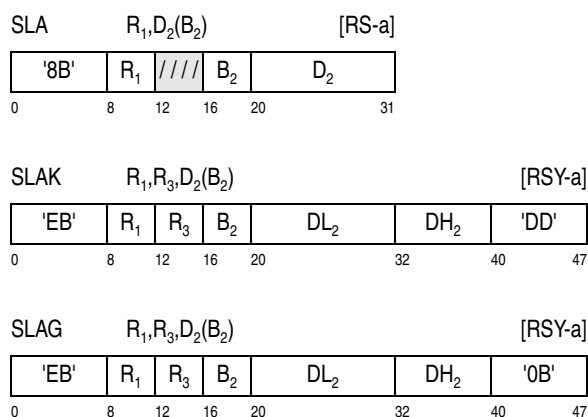
The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 32 of the even-numbered register are not inspected and are lost. Zeros are supplied to the vacated bit positions on the right. Bits 0-31 of general registers R_1 and $R_1 + 1$ remain unchanged.

Condition Code: The code remains unchanged.

Program Exceptions:

- Specification

SHIFT LEFT SINGLE

For SLA, the 31-bit numeric part of the signed first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Bits 0-31 of general register R_1 remain unchanged.

For SLAK, the 31-bit numeric part of the signed third operand is shifted left the number of bits specified by the second-operand address, and the result, with the sign bit of the third operand appended on its left, is placed at the first-operand location. Bits 0-31 of general register R_1 remain unchanged. Except for when the R_1 and R_3 fields designate the same register, the third operand remains unchanged in general register R_3 .

For SLAG, the 63-bit numeric part of the signed third operand is shifted left the number of bits specified by the second-operand address, and the result, with the sign bit of the third operand appended on its left, is placed at the first-operand location. Except for when the R_1 and R_3 fields designate the same register, the third operand remains unchanged in general register R_3 .

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

For SLA, the first operand is treated as a 32-bit signed binary integer in bit positions 32-63 of general register R_1 . The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the left shift.

For SLAK, the first and third operands are treated as 32-bit signed binary integers in bit positions 32-63 of general registers R_1 and R_3 , respectively. The sign of the first operand is set equal to the sign of the third operand. All 31 numeric bits of the third operand participate in the left shift.

For SLAG, the first and third operands are treated as 64-bit signed binary integers in bit positions 0-63 of general registers R_1 and R_3 , respectively. The sign of the first operand is set equal to the sign of the third operand. All 63 numeric bits of the third operand participate in the left shift.

For SLA, SLAG, or SLAK, zeros are supplied to the vacated bit positions on the right.

If one or more bits unlike the sign bit are shifted out of bit position 33, for SLA or SLAK, or 1, for SLAG, an overflow occurs, and condition code 3 is set. If the fixed-point-overflow mask bit is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

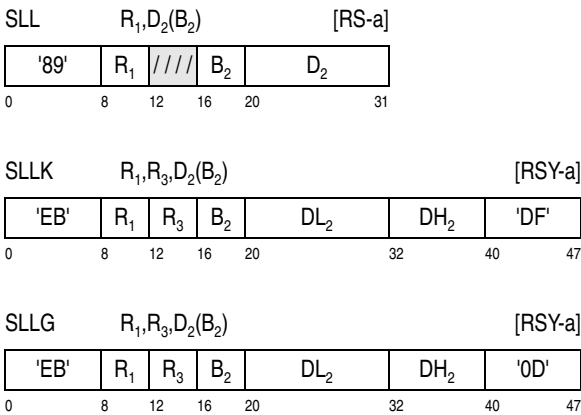
- Fixed-point overflow
- Operation (SLAK, if the distinct-operands facility is not installed)

Programming Notes:

- 1. An example of the use of the SHIFT LEFT SINGLE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

- 2. For SHIFT LEFT SINGLE (SLA and SLAK), for numbers with a value greater than or equal to -2^{30} and less than 2^{30} , a left shift of one bit position is equivalent to multiplying the number by 2. For SHIFT LEFT SINGLE (SLAG), the comparable values are -2^{62} and 2^{62} .
- 3. For SHIFT LEFT SINGLE (SLA and SLAK), shift amounts from 31 to 63 cause the entire numeric part to be shifted out of the register, leaving a result of the maximum negative number or zero, depending on whether or not the initial contents were negative. For SHIFT LEFT SINGLE (SLAG), a shift amount of 63 causes the same effect.

SHIFT LEFT SINGLE LOGICAL



For SLL, the 32-bit first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Bits 0-31 of general register R_1 remain unchanged.

For SLLK, the 32-bit third operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Bits 0-31 of general register R_1 remain unchanged. Except for when the R_1 and R_3 fields designate the same register, the third operand remains unchanged in general register R_3 .

For SLLG, the 64-bit third operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Except for when the R_1 and R_3 fields designate the same register, the third operand remains unchanged in general register R_3 .

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

For SLL, the first operand is in bit positions 32-63 of general register R_1 . All 32 bits of the operand participate in the left shift.

For SLLK, the first and third operands are in bit positions 32-63 of general registers R_1 and R_3 , respectively. All 32 bits of the third operand participate in the left shift.

For SLLG, the first and third operands are in bit positions 0-63 of general registers R_1 and R_3 , respectively. All 64 bits of the third operand participate in the left shift.

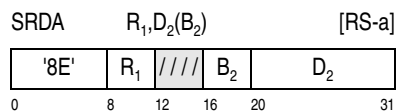
For SLL, SLLG, or SLLK, zeros are supplied to the vacated bit positions on the right.

Condition Code: The code remains unchanged.

Program Exceptions:

- Operation (SLLK, if the distinct-operands facility is not installed)

SHIFT RIGHT DOUBLE



The 63-bit numeric part of the signed first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location. The first operand consists of bits 32-63 of general register R_1 followed on the right by bits 32-63 of general register $R_1 + 1$.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 64-bit signed binary integer. The sign bit of the first operand, bit 32 of the even-numbered register, remains unchanged. Bit position 32 of the odd-numbered register contains a numeric bit, which participates in the shift in the same manner as the other numeric bits. Bits shifted out of bit position 63 of the odd-numbered register are not inspected and are lost. Bits equal to the sign are supplied to the vacated bit positions on the left. Bits 0-31 of general registers R_1 and $R_1 + 1$ remain unchanged.

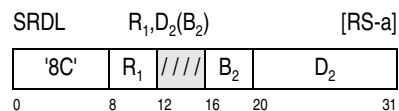
Resulting Condition Code:

- 0 Result zero
- 1 Result less than zero
- 2 Result greater than zero
- 3 --

Program Exceptions:

- Specification

SHIFT RIGHT DOUBLE LOGICAL



The 64-bit first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location. The first operand consists of bits 32-63 of general register R_1 followed on the right by bits 32-63 of general register $R_1 + 1$.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

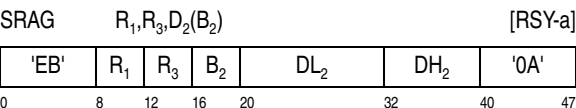
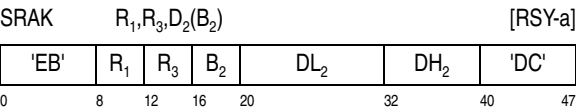
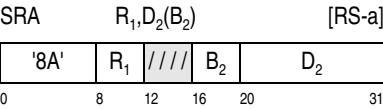
All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 63 of the odd-numbered register are not inspected and are lost. Zeros are supplied to the vacated bit positions on the left. Bits 0-31 of general registers R_1 and $R_1 + 1$ remain unchanged.

Condition Code: The code remains unchanged.

Program Exceptions:

- Specification

SHIFT RIGHT SINGLE



For SRA, The 31-bit numeric part of the signed first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Bits 0-32 of general register R₁ remain unchanged.

For SRAK, the 31-bit numeric part of the signed third operand is shifted right the number of bits specified by the second-operand address, and the result, with the sign bit of the third operand appended on its left, is placed at the first-operand location. Bits 0-31 of general register R₁ remain unchanged. Except for when the R₁ and R₃ fields designate the same register, the third operand remains unchanged in general register R₃.

For SHIFT RIGHT SINGLE (SRAG), the 63-bit numeric part of the signed third operand is shifted right the number of bits specified by the second-operand address, and the result, with the sign bit of the third operand appended on its left, is placed at the first-operand location. Except for when the R₁ and R₃ fields designate the same register, the third operand remains unchanged in general register R₃.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

For SRA, The first operand is treated as a 32-bit signed binary integer in bit positions 32-63 of general register R₁. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the right shift.

For SRAK, the first and third operands are treated as 32-bit signed binary integers in bit positions 32-63 of general registers R₁ and R₃, respectively. The sign of the first operand is set equal to the sign of the third operand. All 31 numeric bits of the third operand participate in the right shift.

For SRAG, the first and third operands are treated as 64-bit signed binary integers in bit positions 0-63 of general registers R₁ and R₃, respectively. The sign of the first operand is set equal to the sign of the third operand. All 63 numeric bits of the third operand participate in the right shift.

For SRA, SRAG, or SRAK, bits shifted out of bit position 63 are not inspected and are lost. Bits equal to the sign are supplied to the vacated bit positions on the left.

Resulting Condition Code:

- 0 Result zero
- 1 Result less than zero
- 2 Result greater than zero
- 3 --

Program Exceptions:

- Operation (SRAK, if the distinct-operands facility is not installed)

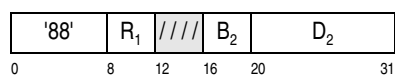
Programming Notes:

1. A right shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the result is equivalent to dividing the number by 2. When an odd number is shifted right one position, the result is equivalent to dividing the *next lower* number by 2. For example, +5 shifted right by one bit position yields +2, whereas -5 yields -3.
2. For SRA and SRAK, shift amounts from 31 to 63 cause the entire numeric part to be shifted out of the register, leaving a result of -1 or zero, depending on whether or not the initial contents were negative. For SHIFT RIGHT SINGLE

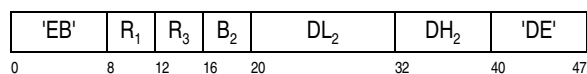
(SRAG), a shift amount of 63 causes the same effect.

SHIFT RIGHT SINGLE LOGICAL

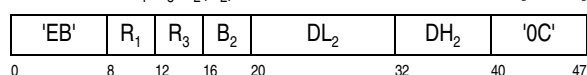
SRL $R_1, D_2(B_2)$ [RS-a]



SRLK $R_1, R_3, D_2(B_2)$ [RSY-a]



SRLG $R_1, R_3, D_2(B_2)$ [RSY-a]



For SRL, the 32-bit first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Bits 0-31 of general register R_1 remain unchanged.

For SRLK, the 32-bit third operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Bits 0-31 of general register R_1 remain unchanged. Except for when the R_1 and R_3 fields designate the same register, the third operand remains unchanged in general register R_3 .

For SRLG, the 64-bit third operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location. Except for when the R_1 and R_3 fields designate the same register, the third operand remains unchanged in general register R_3 .

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

For SRL, the first operand is in bit positions 32-63 of general register R_1 . All 32 bits of the operand participate in the right shift.

For SRLK, the first and third operands are in bit positions 32-63 of general registers R_1 and R_3 , respectively. All 32 bits of the third operand participate in the right shift.

For SRLG, the first and third operands are in bit positions 0-63 of general registers R_1 and R_3 , respectively. All 64 bits of the third operand participate in the right shift.

For SRL, SRLG, or SRLK, bits shifted out of bit position 63 are not inspected and are lost. Zeros are supplied to the vacated bit positions on the left.

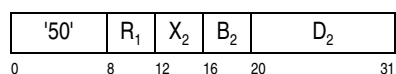
Condition Code: The code remains unchanged.

Program Exceptions:

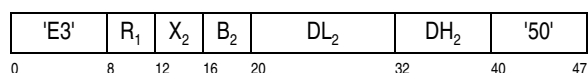
- Operation (SRLK, if the distinct-operands facility is not installed)

STORE

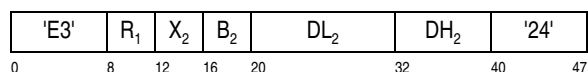
ST $R_1, D_2(X_2, B_2)$ [RX-a]



STY $R_1, D_2(X_2, B_2)$ [RXY-a]

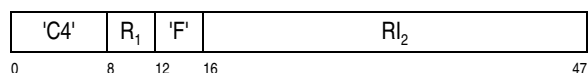


STG $R_1, D_2(X_2, B_2)$ [RXY-a]

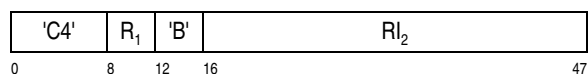


STORE RELATIVE LONG

STRL R_1, Rl_2 [RIL-b]



STGRL R_1, Rl_2 [RIL-b]



The first operand is placed unchanged at the second-operand location.

For STORE (ST, STY) and STORE RELATIVE LONG (STRL), the operands are 32 bits, and, for STORE (STG) and STORE RELATIVE LONG (STGRL), the operands are 64 bits.

The displacement for ST is treated as a 12-bit unsigned binary integer. The displacement for STY and STG is treated as a 20-bit signed binary integer.

For STORE RELATIVE LONG, the contents of the I₂ field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

For STORE RELATIVE LONG (STRL), the second operand must be aligned on a word boundary, and for STORE RELATIVE LONG (STGRL), the second operand must be aligned on a doubleword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

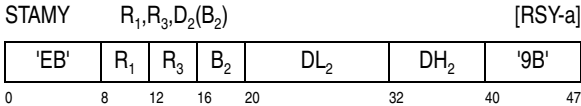
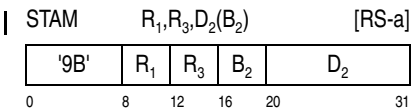
Program Exceptions:

- Access (store, operand 2)
- Operation (STY, if the long-displacement facility is not installed; STRL and STGRL, if the general-instructions-extension facility is not installed)
- Specification (STRL, STGRL only)

Programming Notes:

1. For STORE RELATIVE LONG, the second operand must be aligned on an integral boundary corresponding to the operand's size.
2. When STORE RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.
3. Significant delay may be incurred if the program stores into the same cache line as that containing the storing instruction or into the cache line from which a subsequent instruction may be fetched. The EXTRACT CACHE ATTRIBUTE instruction may be used to determine the cache-line size.

STORE ACCESS MULTIPLE



The contents of the set of access registers starting with access register R₁ and ending with access register R₃ are stored at the locations designated by the second-operand address.

The storage area where the contents of the access registers are placed starts at the location designated by the second-operand address and continues through as many storage words as the number of access registers specified. The contents of the access registers are stored in ascending order of their register numbers, starting with access register R₁ and continuing up to and including access register R₃, with access register 0 following access register 15. The contents of the access registers remain unchanged.

The displacement for STAM is treated as a 12-bit unsigned binary integer. The displacement for STAMY is treated as a 20-bit signed binary integer.

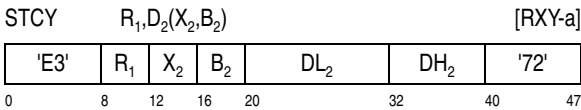
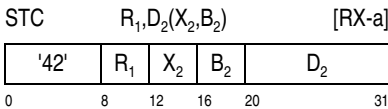
The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (STAMY, if the long-displacement facility is not installed)
- Specification

STORE CHARACTER



Bits 56-63 of general register R₁ are placed unchanged at the second-operand location. The second operand is one byte in length.

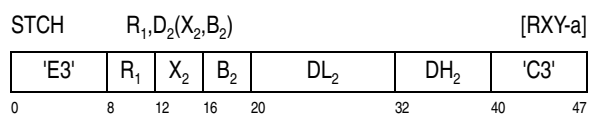
The displacement for STC is treated as a 12-bit unsigned binary integer. The displacement for STCY is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (STCY, if the long-displacement facility is not installed)

STORE CHARACTER HIGH



Bits 24-31 of general register R₁ are placed unchanged at the second-operand location. The second operand is one byte in length.

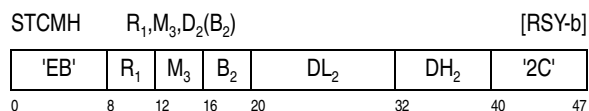
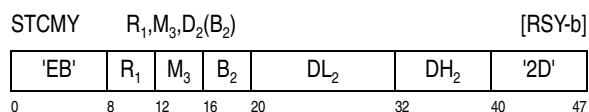
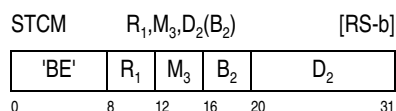
The displacement is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (if the high-word facility is not installed)

STORE CHARACTERS UNDER MASK



Bytes selected from general register R₁ under control of a mask are placed at contiguous byte locations beginning at the second-operand address.

The contents of the M₃ field are used as a mask. These four bits, left to right, correspond one for one with four bytes, left to right, of general register R₁. For STORE CHARACTERS UNDER MASK (STCM, STCMY), the four bytes to which the mask bits correspond are in bit positions 32-63 of general register R₁. For STORE CHARACTERS UNDER MASK (STCMH), the four bytes are in the high-order half, bit positions 0-31, of the register. The bytes corresponding to ones in the mask are placed in the same order at successive and contiguous storage locations beginning at the second-operand address. When the mask is not zero, the length of the second operand is equal to the number of ones in the mask. The contents of the general register remain unchanged.

When the mask is not zero, exceptions associated with storage-operand accesses are recognized only for the number of bytes specified by the mask.

When the mask is zero, the single byte designated by the second-operand address remains unchanged; however, on some models, the contents may be fetched and subsequently stored back unchanged at the same storage location. This update appears to be an interlocked-update reference as observed by other CPUs.

The displacement for STCM is treated as a 12-bit unsigned binary integer. The displacement for STCMY and STCMH is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (STCMY, if the long-displacement facility is not installed)

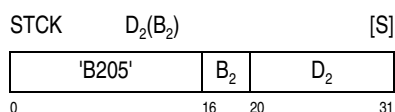
Programming Notes:

1. An example of the use of the STORE CHARACTERS UNDER MASK instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. STORE CHARACTERS UNDER MASK (STCM, STCMY), with a mask of 0111 binary may be

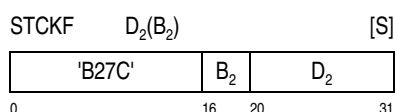
used to store a three-byte address, for example, in modifying the address in a format-0 CCW.

3. STORE CHARACTERS UNDER MASK (STCM, STCMY) with a mask of 1111, 0011, or 0001 binary performs the same function as STORE (ST), STORE HALFWORD, or STORE CHARACTER, respectively.
4. Using STORE CHARACTERS UNDER MASK with a zero mask may cause any of the following to occur for the byte designated by the second-operand address: a PER storage-alteration event may be recognized; access exceptions may be recognized; and, provided no access exceptions exist, the change bit may be set to one. Because the contents of storage remain unchanged, the change bit may or may not be one when a PER storage-alteration event is recognized.
5. On certain models, STORE CHARACTERS UNDER MASK with mask field that specifies discontinuous bytes may perform slower than when the mask specifies contiguous bytes.

STORE CLOCK



STORE CLOCK FAST



The current value of bits 0-63 of the TOD clock is stored in the eight-byte field designated by the second-operand address, provided the clock is in the set, stopped, or not-set state.

When the clock is stopped, zeros are stored in positions to the right of the rightmost bit position that is incremented when the clock is running. For STORE CLOCK, when the value of a running clock is stored, nonzero values may be stored in positions to the right of the rightmost incremented bit; this is to ensure that a unique value is stored. For STORE CLOCK FAST, when the value of a running clock is stored, bits to the right of the rightmost bit that is incremented are stored as zeros.

Zeros are stored at the operand location when the clock is in the error state or the not-operational state.

The quality of the clock value stored by the instruction is indicated by the resultant condition-code setting.

For STORE CLOCK, a serialization function is performed before the value of the clock is fetched and again after the value is placed in storage.

Resulting Condition Code:

- 0 Clock in set state
- 1 Clock in not-set state
- 2 Clock in error state
- 3 Clock in stopped state or not-operational state

Program Exceptions:

- Access (store, operand 2)
- Operation (STCKF, if the store-clock-fast facility is not installed)

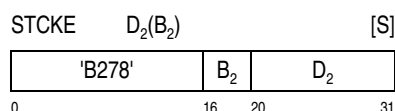
Programming Notes:

1. Bit position 31 of the clock is incremented every 1.048576 seconds; hence, for timing applications involving human responses, the leftmost clock word may provide sufficient resolution.
2. Condition code 0 normally indicates that the clock has been set by the control program. Accordingly, the value may be used in elapsed-time measurements and as a valid time-of-day and calendar indication. Condition code 1 indicates that the clock value is the elapsed time since the power for the clock was turned on. In this case, the value may be used in elapsed-time measurements but is not a valid time-of-day indication. Condition codes 2 and 3 mean that the value provided by STORE CLOCK cannot be used for time measurement or indication.
3. Condition code 3 indicates that the clock is in either the stopped state or the not-operational state. These two states can normally be distinguished because an all-zero value is stored when the clock is in the not-operational state.
4. If a problem program written for z/Architecture is to be executed also on a system in the System/370 mode, then the program should take into account that, in the System/370 mode, the value

stored when the condition code is 2 is not necessarily zero.

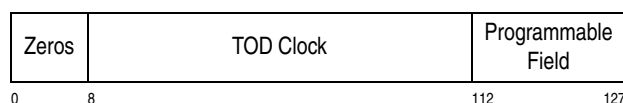
5. Two executions of STORE CLOCK FAST, or an execution of STORE CLOCK FAST and STORE CLOCK, either on the same or different CPUs, do not necessarily return different values of the clock if the clock is running. Thus, the values returned by STORE CLOCK FAST do not necessarily indicate the correct sequence of execution of the instruction by one or more CPUs.
6. When the TOD-clock-steering facility is installed, and assuming a valid operating system, then, for the problem program, the TOD clock is always in the set state and there is no need to test the condition code after issuing STORE CLOCK or STORE CLOCK FAST.

STORE CLOCK EXTENDED



The current value of bits 0-103 of the TOD clock is stored in byte positions 1-13 of the sixteen-byte field designated by the second-operand address, provided the clock is in the set, stopped, or not-set state. Zeros are stored in byte position 0. The TOD programmable field, bits 16-31 of the TOD programmable register, is stored in byte positions 14 and 15.

The operand just described has the following format:



When the clock is stopped, zeros are stored in the clock value in positions to the right of the rightmost bit position that is incremented when the clock is running. The programmable field still is stored.

When the value of a running clock is stored, the value in bit positions 64-103 of the clock (bit positions 72-111 of the storage operand) is always nonzero; this ensures that values stored by STORE CLOCK EXTENDED are unique when compared with values stored by STORE CLOCK and extended with zeros.

Zeros are stored at the operand location when the clock is in the error state or the not-operational state.

The quality of the clock value stored by the instruction is indicated by the resultant condition-code setting.

A serialization function is performed before the value of the clock is fetched and again after the value is placed in storage.

Resulting Condition Code:

- 0 Clock in set state
- 1 Clock in not-set state
- 2 Clock in error state
- 3 Clock in stopped state or not-operational state

Program Exceptions:

- Access (store, operand 2)

1. Condition code 0 normally indicates that the clock has been set by the control program. Accordingly, the value may be used in elapsed-time measurements and as a valid time-of-day and calendar indication. Condition code 1 indicates that the clock value is the elapsed time since the power for the clock was turned on. In this case, the value may be used in elapsed-time measurements but is not a valid time-of-day indication. Condition codes 2 and 3 mean that the value provided by STORE CLOCK EXTENDED cannot be used for time measurement or indication.
2. When the TOD-clock-steering facility is installed, and assuming a valid operating system, then, for the problem program, the TOD clock is always in the set state and there is no need to test the condition code after issuing STORE CLOCK EXTENDED.
3. Programming notes beginning on page 4-41 show hex values related to the value of the TOD clock as it is stored by the STORE CLOCK instruction. Notes 4-6, below, are repetitions of those notes except with the text and hex values adjusted so they apply to bits 0-71 of the value stored by STORE CLOCK EXTENDED.
4. The following chart shows the time interval between instants at which various bits of the TOD-clock value stored by STORE CLOCK EXTENDED are stepped. This time value may

also be considered as the weighted time value that the bit, when one, represents. The bit numbers are those of the STORE CLOCK EXTENDED operand.

STCKE Bit	Stepping Interval			
	Days	Hours	Min.	Seconds
59				0.000 001
55				0.000 016
51				0.000 256
47				0.004 096
43				0.065 536
39				1.048 576
35				16.777 216
31			4	28.435 456
27		1	11	34.967 296
23		19	5	19.476 736
19	12	17	25	11.627 776
15	203	14	43	6.044 416
11	3257	19	29	36.710 656

5. The following chart shows the setting of bits 0-63 of the STORE CLOCK EXTENDED operand for 00:00:00 (0 am), UTC time, for several dates: January 1, 1900, January 1, 1972, and for that instant in time just after each of the 24 leap seconds that have occurred through January, 2009. Each of these leap seconds was inserted in the UTC time scale beginning at 23:59:60 UTC of the day previous to the one listed and ending at 00:00:00 UTC of the day listed.

Year	Month	Day	Leap Sec	STCKE Value (Hex) Bits 0-63
1900	1	1		0000 0000 0000 0000
1972	1	1		0081 26D6 0E46 0000
1972	7	1	1	0082 0BA9 811E 2400
1973	1	1	2	0082 F300 AEE2 4800
1974	1	1	3	0084 BDE9 7114 6C00
1975	1	1	4	0086 88D2 3346 9000
1976	1	1	5	0088 53BA F578 B400
1977	1	1	6	008A 1FE5 9520 D800
1978	1	1	7	008B EACE 5752 FC00
1979	1	1	8	008D B5B7 1985 2000
1980	1	1	9	008F 809F DBB7 4400
1981	7	1	10	0092 305C 0FCD 6800
1982	7	1	11	0093 FB44 D1FF 8C00
1983	7	1	12	0095 C62D 9431 B000

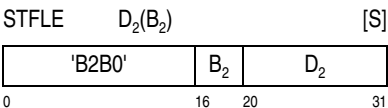
Year	Month	Day	Leap Sec	STCKE Value (Hex) Bits 0-63
1985	7	1	13	0099 5D40 F517 D400
1988	1	1	14	009D DA69 A557 F800
1990	1	1	15	00A1 717D 063E 1C00
1991	1	1	16	00A3 3C65 C870 4000
1992	7	1	17	00A5 EC21 FC86 6400
1993	7	1	18	00A7 B70A BEB8 8800
1994	7	1	19	00A9 81F3 80EA AC00
1996	1	1	20	00AC 3433 6FEC D000
1997	7	1	21	00AE E3EF A402 F400
1999	1	1	22	00B1 962F 9305 1800
2006	1	1	23	00BE 2510 9797 3C00
2009	1	1	24	00C3 870C B9BB 6000

6. The stepping value of TOD-clock bit position 63, if implemented, is 2^{-12} microseconds, or approximately 244 picoseconds. This value is called a clock unit.

The following chart shows various time intervals in clock units expressed in hexadecimal notation. The chart shows the values stored in bit positions 0-71 of the STORE CLOCK EXTENDED operand. Bit 71 of the operand represents a clock unit.

Interval	Clock Units (Hex) Bits 0-71
1 microsecond	0010 00
1 millisecond	3E80 00
1 second	00F4 2400 00
1 minute	3938 7000 00
1 hour	000D 693A 4000 00
1 day	0141 DD76 0000 00
365 days	0001 CAE8 C13E 0000 00
366 days	0001 CC2A 9EB4 0000 00
1,461 days*	0007 2CE4 E26E 0000 00
* Number of days in four years, including a leap year. Note that the year 1900 was not a leap year. Thus, the four-year span starting in 1900 has only 1,460 days.	

STORE FACILITY LIST EXTENDED



A list of bits providing information about facilities is stored beginning at the doubleword specified by the second operand address.

For the leftmost doublewords in which facility bits are assigned, the reserved bits are stored as zeros. Doublewords to the right of the doubleword in which the highest-numbered facility bit is assigned for a model may or may not be stored. Access exceptions and PER events are not recognized for doublewords that are not stored.

The size of the second operand, in doublewords, is one more than the value specified in bits 56-63 of general register 0. The remaining bits of general register 0 are unassigned and should contain zeros; otherwise, the program may not operate compatibly in the future.

When the size of the second operand is large enough to contain all of the facility bits assigned for a model, then the complete facility list is stored in the second operand location, bits 56-63 of general register 0 are updated to contain one less than the number of doublewords needed to contain all of the facility bits assigned for the model, and condition code 0 is set.

When the size of the second operand is not large enough to contain all of the facility bits assigned for a model, then only the number of doublewords specified by the second-operand size are stored, bits 56-63 of general register 0 are updated to contain one less than the number of doublewords needed to contain all of the facility bits assigned for the model, and condition code 3 is set.

Figure 4-19, “Assigned Facility Bits” on page 4-67 shows the meanings of the assigned facility bits.

Special Conditions

The second operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Complete facility list stored
- 1 --
- 2 --
- 3 Incomplete facility list stored

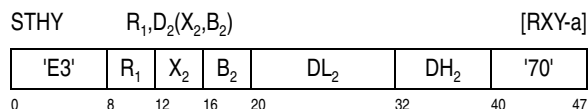
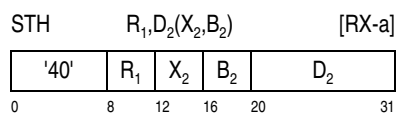
Program Exceptions:

- Access (store, second operand)
- Operation (if the store-facility-list-extended facility is not installed)
- Specification

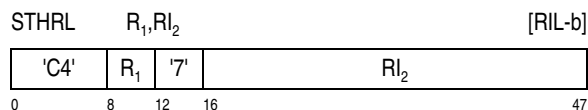
Programming Notes:

1. The performance of STORE FACILITY LIST EXTENDED may be significantly slower than that of simply testing a byte in storage. Programs that need to frequently test for the presence of a facility — for example, dual-path code in which the facility is used in one path but not another — should execute the STORE FACILITY LIST EXTENDED instruction once during initialization. Subsequently, the program may test for the presence of the facility by examining the stored result, using an instruction such as TEST UNDER MASK.
2. When condition code 0 is set, bits 56-63 of general register 0 are updated to contain a value that is one less than the number of doublewords stored. If the program chooses to ignore the results in general register 0, then it should ensure that the entire second operand in storage is set to zero prior to executing STORE FACILITY LIST EXTENDED.

STORE HALFWORD



STORE HALFWORD RELATIVE LONG



Bits 48-63 of general register R_1 are placed unchanged at the second-operand location. The second operand is two bytes in length.

The displacement for STH is treated as a 12-bit unsigned binary integer. The displacement for STHY is treated as a 20-bit signed binary integer.

For STORE HALFWORD RELATIVE LONG, the contents of the I_2 field are a signed binary integer specifying the number of halfwords that is added to the address of the instruction to generate the address of the second operand in storage. When DAT is on, the second operand is accessed using the same addressing-space mode as that used to access the instruction. When DAT is off, the second operand is accessed using a real address.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

Operation (STHY if the long-displacement facility is not installed; STHRL, if the general-instructions-extension facility is not installed.)

Programming Notes:

- 1. For STORE HALFWORD RELATIVE LONG, the second operand is necessarily aligned on an integral boundary corresponding to the operand's size.
- 2. When STORE HALFWORD RELATIVE LONG is the target of an execute-type instruction, the second-operand address is relative to the target address.
- 3. Significant delay may be incurred if the program stores into the same cache line as that containing the storing instruction. The EXTRACT CACHE ATTRIBUTE instruction may be used to determine the cache-line size.

STORE HALFWORD HIGH



Bits 16-31 of general register R_1 are placed unchanged at the second-operand location. The second operand is two bytes in length.

The displacement is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (if the high-word facility is not installed)

STORE HIGH



The first operand is placed unchanged at the second-operand location. The first operand is in bits 0-31 of general register R_1 , and the second operand is 32 bits in storage.

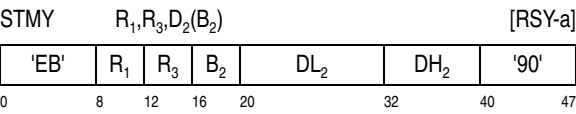
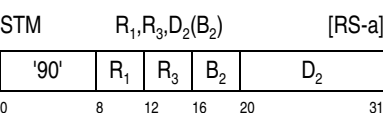
The displacement is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (if the high-word facility is not installed)

STORE MULTIPLE



The contents of bit positions of the set of general registers starting with general register R_1 and ending with general register R_3 are placed in the storage area beginning at the location designated by the second-operand address and continuing through as many locations as needed.

For STORE MULTIPLE (STM, STMY), the contents of bit positions 32-63 of the general registers are

stored in successive four-byte fields beginning at the second-operand address. For STORE MULTIPLE (STMG), the contents of bit positions 0-63 of the general registers are stored in successive eight-byte fields beginning at the second-operand address.

The general registers are stored in the ascending order of their register numbers, starting with general register R_1 and continuing up to and including general register R_3 , with general register 0 following general register 15.

The displacement for STM is treated as a 12-bit unsigned binary integer. The displacement for STMY and STMG is treated as a 20-bit signed binary integer.

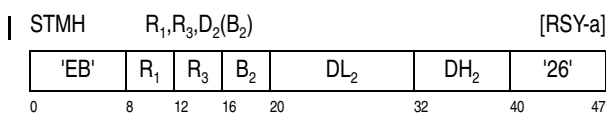
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (STMY, if the long-displacement facility is not installed)

Programming Note: An example of the use of the STORE MULTIPLE instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

STORE MULTIPLE HIGH



The contents of the high-order halves, bit positions 0-31, of the set of general registers starting with general register R_1 and ending with general register R_3 are placed in the storage area beginning at the location designated by the second-operand address and continuing through as many locations as needed, that is, the contents of bit positions 0-31 are stored in successive four-byte fields beginning at the second-operand address. Bits 32-63 of the registers are ignored.

The general registers are stored in the ascending order of their register numbers, starting with general register R_1 and continuing up to and including general register R_3 , with general register 0 following general register 15.

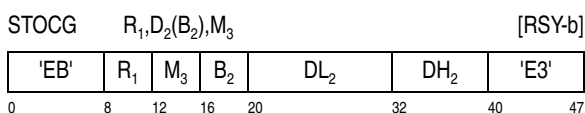
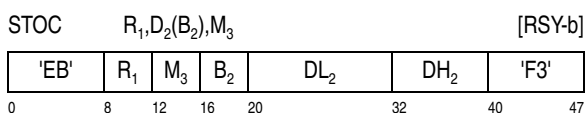
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)

Programming Note: All combinations of register numbers specified by R_1 and R_3 are valid. When the register numbers are equal, only four bytes are transmitted. When the number specified by R_3 is less than the number specified by R_1 , the register numbers wrap around from 15 to 0.

STORE ON CONDITION



The first operand is placed unchanged at the second-operand location if the condition code has one of the values specified by M_3 ; otherwise, the second operand remains unchanged.

For STOC, the first and second operands are 32 bits, and for STOCG, the first and second operands are 64 bits.

The M_3 field is used as a four-bit mask. The four condition codes (0, 1, 2, and 3) correspond, left to right, with the four bits of the mask, as follows:

Condition Code	Mask Position Value
0	8
1	4
2	2
3	1

The current condition code is used to select the corresponding mask bit. If the mask bit selected by the condition code is one, the store is performed. If the mask bit selected is zero, the store is not performed.

The displacement is treated as a 20-bit signed binary integer.

When the condition specified by the M_3 field is not met (that is, store operation is not performed), it is model dependent whether any or all of the following occur for the second operand: (a) an access exception is recognized, (b) a PER storage-alteration event is recognized, (c) a PER zero-address-detection event is recognized, or (d) the change bit is set.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (if the load/store-on-condition facility is not installed)

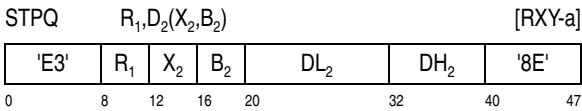
Programming Notes:

1. When the M_3 field contain all zeros and no exception condition exists, the instruction acts as a NOP. When the M_3 field contains all ones and no exception condition exists, the store operation is always performed. However, these are not the preferred means of implementing a NOP or unconditional store, respectively.
2. When the condition specified by the M_3 field is not met, it is model dependent whether the second operand is brought into the cache.
3. STORE ON CONDITION provides a function similar to that of a separate BRANCH ON CONDITION instruction followed by a STORE instruction, except that STORE ON CONDITION does not provide an index register. For example, the following two instruction sequences are equivalent.

```
STOCG 15,256(7),8      BC 7,SKIP
                               STG 15,256(0,7)
                               SKIP DS 0H
```

On models that implement predictive branching, the combination of the BRANCH ON CONDITION and STORE instructions may perform somewhat better than the STORE ON CONDITION instruction when the CPU is able to successfully predict the branch condition. However, on models where the CPU is not able to successfully predict the branch condition, such as when the condition is more random, the STORE ON CONDITION instruction may provide significant performance improvement.

STORE PAIR TO QUADWORD



The quadword first operand is stored at the second-operand location. The store at the second-operand location appears to be quadword concurrent as observed by other CPUs. The left doubleword of the first operand is in general register R_1 , and the right doubleword is in general register $R_1 + 1$.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register. The second operand must be designated on a quadword boundary. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

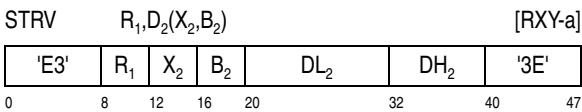
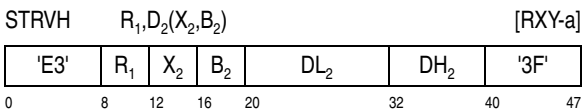
Program Exceptions:

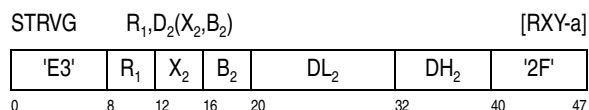
- Access (store, operand 2)
- Specification

Programming Notes:

1. The STORE MULTIPLE (STM or STMG) instruction does not necessarily provide quadword-concurrent access.
2. The performance of STORE PAIR TO QUADWORD on some models may be significantly slower than that of STORE MULTIPLE (STMG). Unless quadword consistency is required, STMG should be used instead of STPQ.

STORE REVERSED





The first operand is placed at the second-operand location with the left-to-right sequence of the bytes reversed.

For STORE REVERSED (STRVH), the first operand is two bytes in bit positions 48-63 of general register R_1 . For STORE REVERSED (STRV), the first operand is four bytes in bit positions 32-63 of general register R_1 . For STORE REVERSED (STRVG), the first operand is eight bytes in bit positions 0-63 of general register R_1 .

Condition Code: The code remains unchanged.

Program Exceptions:

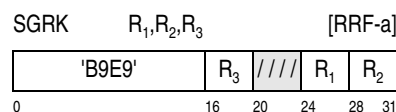
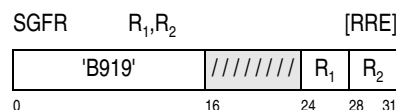
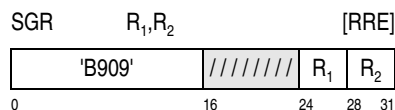
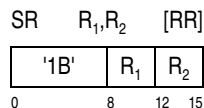
- Access (store, operand 2)

Programming Notes:

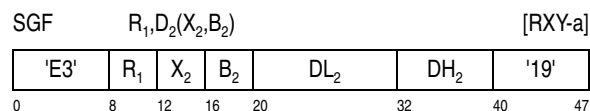
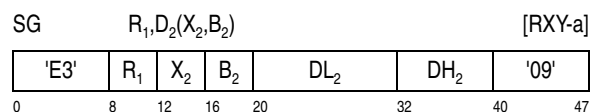
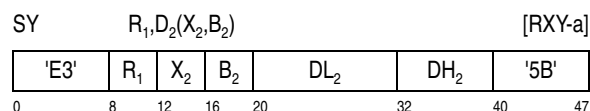
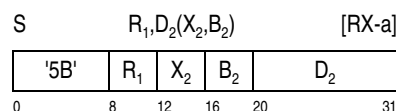
1. The instruction can be used to convert two, four, or eight bytes from a “little-endian” format to a “big-endian” format, or vice versa. In the big-endian format, the bytes in a left-to-right sequence are in the order most significant to least significant. In the little-endian format, the bytes are in the order least significant to most significant. For example, the bytes ABCD in the big-endian format are DCBA in the little-endian format.
2. The storage-operand references of STORE REVERSED may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

SUBTRACT

Register-and-register formats:



Register-and-storage formats:



For S, SG, SGF, SGFR, SGR, SR, and SY, the second operand is subtracted from the first operand, and the difference is placed at the first-operand location. For SGRK and SRK, the third operand is subtracted from the second operand, and the difference is placed at the first-operand location.

For S, SR, SRK, and SY, the operands and the difference are treated as 32-bit signed binary integers. For SG, SGR, and SGRK, they are treated as 64-bit signed binary integers. For SGFR and SGF, the second operand is treated as a 32-bit signed binary integer, and the first operand and the difference are treated as 64-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is

the third operand is in bits 32-63 of general register R_3 ; bits 0-31 of the register are ignored.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

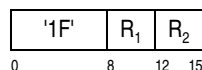
Program Exceptions:

- Fixed-point overflow
- Operation (if the high-word facility is not installed)

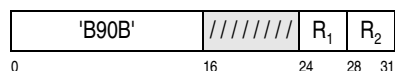
SUBTRACT LOGICAL

Register-and-register formats:

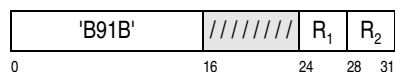
SLR R_1, R_2 [RR]



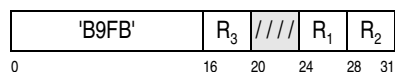
SLGR R_1, R_2 [RRE]



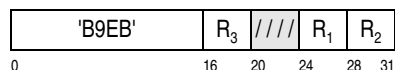
SLGFR R_1, R_2 [RRE]



SLRK R_1, R_2, R_3 [RRF-a]

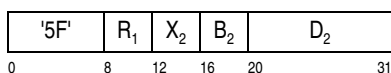


SLGRK R_1, R_2, R_3 [RRF-a]

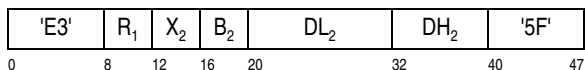


Register-and-storage formats:

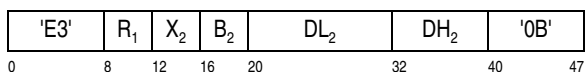
SL $R_1, D_2(X_2, B_2)$ [RX-a]



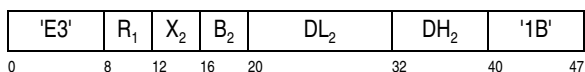
SLY $R_1, D_2(X_2, B_2)$ [RXY-a]



SLG $R_1, D_2(X_2, B_2)$ [RXY-a]

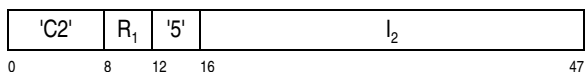


SLGF $R_1, D_2(X_2, B_2)$ [RXY-a]

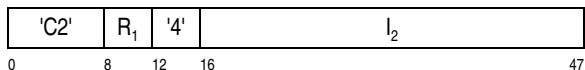


SUBTRACT LOGICAL IMMEDIATE

SLFI R_1, I_2 [RIL-a]



SLGFI R_1, I_2 [RIL-a]



For SUBTRACT LOGICAL (SL, SLG, SLGF, SLGFR, SLGR, SLR, SLY) and for SUBTRACT LOGICAL IMMEDIATE, the second operand is subtracted from the first operand, and the difference is placed at the first-operand location. For SUBTRACT LOGICAL (SLGRK and SLRK), the third operand is subtracted from the second operand, and the difference is placed at the first-operand location.

For SUBTRACT LOGICAL (SL, SLR, SLRK, and SLY) and for SUBTRACT LOGICAL IMMEDIATE (SLFI), the operands and the difference are treated as 32-bit unsigned binary integers. For SUBTRACT LOGICAL (SLG, SLGR, and SLGRK), they are treated as 64-bit unsigned binary integers. For SUBTRACT LOGICAL (SLGFR, SLGF) and for SUBTRACT LOGICAL IMMEDIATE (SLGFI), the second operand is treated as a 32-bit unsigned binary integer, and the first operand and the difference are treated as 64-bit unsigned binary integers.

The displacement for SL is treated as a 12-bit unsigned binary integer. The displacement for SLY, SLG, and SLGF is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 --
- 1 Result not zero; borrow
- 2 Result zero; no borrow
- 3 Result not zero; no borrow

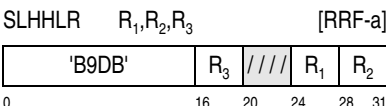
Program Exceptions:

- Access (fetch, operand 2 of SL, SLY, SLG, and SLGF only)
- Operation (SLY, if the long-displacement facility is not installed; SLFI and SLGFI, if the extended-immediate facility is not installed; SLRK and SLGRK, if the distinct-operands facility is not installed)

Programming Notes:

- Logical subtraction is performed by adding the one's complement of the second operand and a value of one to the first operand. The use of the one's complement and the value of one instead of the two's complement of the second operand results in a carry when the second operand is zero.
- SUBTRACT LOGICAL differs from SUBTRACT only in the meaning of the condition code and in the absence of the interruption for overflow.
- A zero difference is always accompanied by a carry out of bit position 0 for 64-bit results or bit position 32 for 32-bit results, and, therefore, no borrow.
- The condition-code setting for SUBTRACT LOGICAL can also be interpreted as indicating the presence or absence of a carry, as follows:
 - 1 Result not zero; no carry
 - 2 Result zero; carry
 - 3 Result not zero; carry

SUBTRACT LOGICAL HIGH



The third operand is subtracted from the second operand, and the difference is placed at the first-operand location. The operands and the difference are treated as 32-bit unsigned binary integers.

The first and second operands are in bits 0-31 of general registers R₁ and R₂, respectively; bits 32-63 of general register R₁ are unchanged, and bits 32-63 of general register R₂ are ignored. For SLHHHR, the third operand is in bits 0-31 of general register R₃; bits 32-63 of the register are ignored. For SLHHLR, the third operand is in bits 32-63 of general register R₃; bits 0-31 of the register are ignored.

Resulting Condition Code:

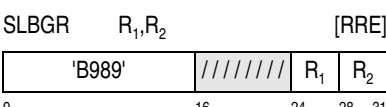
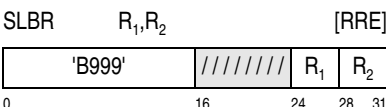
- 0 --
- 1 Result not zero; borrow
- 2 Result zero; no borrow
- 3 Result not zero; no borrow

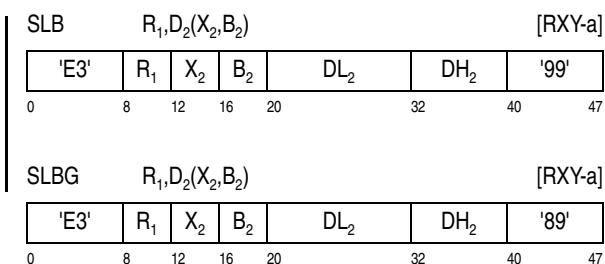
Program Exceptions:

- Operation (if the high-word facility is not installed)

SUBTRACT LOGICAL WITH BORROW

Register-and-register formats:



Register-and-storage formats:

The second operand and the borrow are subtracted from the first operand, and the difference is placed at the first-operand location. For SUBTRACT LOGICAL WITH BORROW (SLBR, SLB), the operands, the borrow, and the difference are treated as 32-bit unsigned binary integers. For SUBTRACT LOGICAL WITH BORROW (SLBGR, SLBG), they are treated as 64-bit unsigned binary integers.

Resulting Condition Code:

- 0 Result zero; borrow
- 1 Result not zero; borrow
- 2 Result zero; no borrow
- 3 Result not zero; no borrow

Program Exceptions:

- Access (fetch, operand 2 of SLB and SLBG only)

Programming Notes:

1. A borrow is represented by a zero value of bit 18 of the current PSW. Bit 18 is the leftmost bit of the two-bit condition code in the PSW. Bit 18 is set to zero by an execution of a SUBTRACT LOGICAL or SUBTRACT LOGICAL WITH BORROW instruction that produces a borrow into the leftmost bit position of the 32-bit or 64-bit result.
2. Logical subtraction with borrow is performed by adding the one's complement of the second operand and bit 18 of the current PSW to the first operand. Therefore, when bit 18 is one, indicating no borrow, the addition is the same as for SUBTRACT LOGICAL.
3. Condition code zero is set for SUBTRACT LOGICAL WITH BORROW (SLBR, SLB), when the maximum 32-bit unsigned binary integer, $2^{32} - 1$,

is subtracted from zero when PSW bit 18 indicates a borrow. For SUBTRACT LOGICAL WITH BORROW (SLBGR, SLBG) condition code zero is set when the maximum 64-bit unsigned binary integer, $2^{64} - 1$, is subtracted from zero when PSW bit 18 indicates a borrow.

4. SUBTRACT and SUBTRACT LOGICAL may provide better performance than SUBTRACT LOGICAL WITH BORROW, depending on the model.

SUPERVISOR CALL

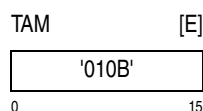
The instruction causes a supervisor-call interruption, with the I field of the instruction providing the rightmost byte of the interruption code.

Bits 8-15 of the instruction, with eight zeros appended on the left, are placed in the supervisor-call interruption code that is stored in the course of the interruption. See "Supervisor-Call Interruption" on page 6-50.

A serialization and checkpoint-synchronization function is performed.

Condition Code: The code remains unchanged and is saved as part of the old PSW. A new condition code is loaded as part of the supervisor-call interruption.

Program Exceptions: None.

TEST ADDRESSING MODE

The extended-addressing-mode bit and basic-addressing-mode bit, bits 31 and 32 of the current PSW, respectively, are tested, and the result is indicated in the condition code.

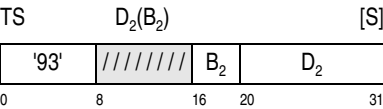
Resulting Condition Code:

- 0 PSW bits 31 and 32 zeros (indicating 24-bit addressing mode)
- 1 PSW bit 31 zero and bit 32 one (indicating 31-bit addressing mode)
- 2 --
- 3 PSW bits 31 and 32 ones (indicating 64-bit addressing mode)

Program Exceptions: None.

Programming Note: The case when PSW bit 31 is one and bit 32 is zero causes an early PSW specification exception to be recognized.

TEST AND SET



The leftmost bit (bit position 0) of the byte located at the second-operand address is used to set the condition code, and then the byte is set to all ones.

Bits 8-15 of the instruction are ignored.

The byte in storage is set to all ones as it is fetched for the testing of bit 0. This update appears to be an interlocked-update reference as observed by other CPUs.

A serialization function is performed before the byte is fetched and again after the storing of all ones.

Resulting Condition Code:

- 0 Leftmost bit zero
- 1 Leftmost bit one
- 2 --
- 3 --

Program Exceptions:

- Access (fetch and store, operand 2)

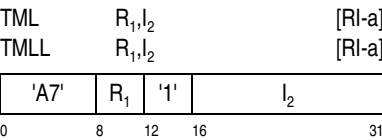
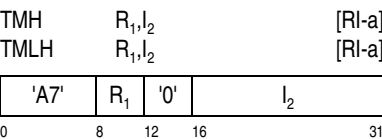
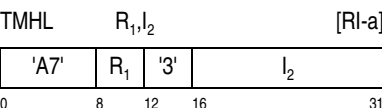
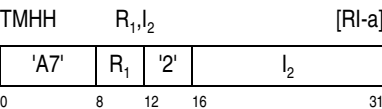
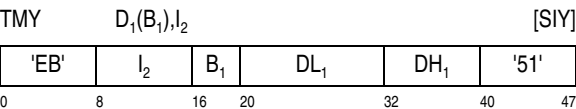
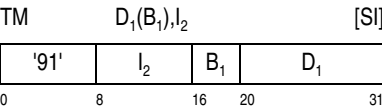
Programming Notes:

- 1. TEST AND SET may be used for controlled sharing of a common storage area by programs operating on different CPUs. This instruction is

provided primarily for compatibility with programs written for System/360. The instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP provide functions which are more suitable for sharing among programs on a single CPU or for programs that may be interrupted. See the description of these instructions and the associated programming notes for details.

- 2. TEST AND SET does not interlock against storage accesses by channel programs. Therefore, the instruction should not be used to update a location into which a channel program may store, since the channel-program data may be lost.

TEST UNDER MASK (TEST UNDER MASK HIGH, TEST UNDER MASK LOW)



A mask is used to select bits of the first operand, and the result is indicated in the condition code.

TEST UNDER MASK is a new name of, and TMLH and TMLL are new mnemonics for, the ESA/390 instructions TEST UNDER MASK HIGH (TMH) and TEST UNDER MASK LOW (TML), respectively.

In TEST UNDER MASK (TM, TMY), the byte of immediate data, I_2 , is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the byte in storage designated by the first-operand address.

A mask bit of one indicates that the storage bit is to be tested. When the mask bit is zero, the storage bit is ignored. When all storage bits thus selected are zero, condition code 0 is set. Condition code 0 is also set when the mask is all zeros. When the selected bits are all ones, condition code 3 is set; otherwise, condition code 1 is set.

Access exceptions associated with the storage operand are recognized for one byte even when the mask is all zeros.

In TEST UNDER MASK (TMHH, TMHL, TMLH, TMLL), The contents of the I_2 field are used as a 16-bit mask. For each instruction, the bits of the mask are made to correspond one for one with 16 bits of the first operand as follows:

Instruction	Bits Tested
TMHH	0-15
TMHL	16-31
TMLH (or TMH)	32-47
TMLL (or TML)	48-63

A mask bit of one indicates that the first-operand bit is to be tested. When the mask bit is zero, the first-operand bit is ignored. When all first-operand bits thus selected are zero, condition code 0 is set. Condition code 0 is also set when the mask is all zeros. When the selected bits are mixed zeros and ones, condition code 1 is set if the leftmost selected bit is zero, or condition code 2 is set if the leftmost selected bit is one. When the selected bits are all ones, condition code 3 is set.

The displacement for TM is treated as a 12-bit unsigned binary integer. The displacement for TMY is treated as a 20-bit signed binary integer.

Resulting Condition Code:

- 0 Selected bits all zeros; or mask bits all zeros
- 1 Selected bits mixed zeros and ones (TM and TMY only)
- 1 Selected bits mixed zeros and ones, and leftmost is zero (TMHH, TMHL, TMLH, TMLL)
- 2 -- (TM and TMY only)
- 2 Selected bits mixed zeros and ones, and leftmost is one (TMHH, TMHL, TMLH, TMLL)
- 3 Selected bits all ones

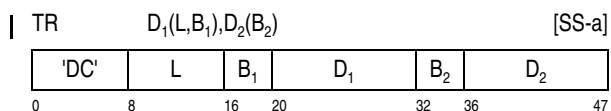
Program Exceptions:

- Access (fetch, operand 1, TM and TMY only)
- Operation (TMY, if the long-displacement facility is not installed)

Programming Notes:

1. An example of the use of the TEST UNDER MASK (TM) instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. When the mask for TMHH, TMHL, TMLH, or TMLL selects exactly two bits, the two selected bits effectively are loaded into the condition code.

TRANSLATE



The bytes of the first operand are used as eight-bit arguments to reference a list designated by the second-operand address. Each function byte selected from the list replaces the corresponding argument in the first operand.

The L field specifies the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding left to right. Each argument byte is added to the initial second-operand address. The addition is performed following the rules for address arithmetic, with the argument byte treated as an eight-bit unsigned binary integer and extended with zeros on the left. The sum is used as

the address of the function byte, which then replaces the original argument byte.

The operation proceeds until the first-operand field is exhausted. The list is not altered unless an overlap occurs.

When the operands overlap, the result is obtained as if each result byte were stored immediately after fetching the corresponding function byte.

Access exceptions are recognized only for those bytes in the second operand which are actually required.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2; fetch and store, operand 1)

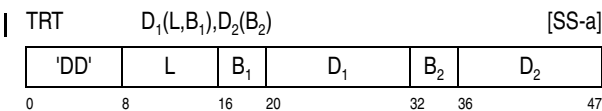
Programming Notes:

1. An example of the use of the TRANSLATE instruction is given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. TRANSLATE may be used to convert data from one code to another code.
3. The instruction may also be used to rearrange data. This may be accomplished by placing a pattern in the destination area, by designating the pattern as the first operand of TRANSLATE, and by designating the data that is to be rearranged as the second operand. Each byte of the pattern contains an eight-bit number specifying the byte destined for this position. Thus, when the instruction is executed, the pattern selects the bytes of the second operand in the desired order.
4. Because each eight-bit argument byte is added to the initial second-operand address to obtain the address of a function byte, the list may contain 256 bytes. In cases where it is known that not all eight-bit argument values will occur, it is possible to reduce the size of the list.
5. Significant performance degradation is possible when the second-operand address of TRANSLATE designates a location that is less than 256 bytes to the left of a 4K-byte boundary. This is because the machine may perform a trial execu-

tion of the instruction to determine if the second operand actually crosses the boundary.

6. The fetch and subsequent store accesses to a particular byte in the first-operand field do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel program may also be updating the location. An example of this effect is shown for OR (OI) in “Multiprogramming and Multiprocessing Examples” in Appendix A, “Number Representation and Instruction-Use Examples.”
7. The storage-operand references of TRANSLATE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

TRANSLATE AND TEST



The bytes of the first operand are used as eight-bit arguments to select function bytes from a list designated by the second-operand address. The first non-zero function byte is inserted in general register 2, and the related argument address in general register 1.

The L field specifies the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding left to right. The first operand remains unchanged in storage. Calculation of the address of the function byte is performed as in the TRANSLATE instruction. The function byte retrieved from the list is inspected for a value of zero.

When the function byte is zero, the operation proceeds with the next byte of the first operand. When the first-operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting condition code 0. The contents of general registers 1 and 2 remain unchanged.

When the function byte is nonzero, the operation is completed by inserting the function byte in general register 2 and the related argument address in general register 1. This address points to the argument

byte last translated. The function byte replaces bits 56-63 of general register 2, and bits 0-55 of this register remain unchanged. In the 24-bit addressing mode, the address replaces bits 40-63 of general register 1, and bits 0-39 of this register remain unchanged. In the 31-bit addressing mode, the address replaces bits 33-63 of general register 1, bit 32 of this register is set to zero, and bits 0-31 of the register remain unchanged. In the 64-bit addressing mode, the address replaces bits 0-63 of general register 1.

When the function byte is nonzero, either condition code 1 or 2 is set, depending on whether the argument byte is the rightmost byte of the first operand. Condition code 1 is set if one or more argument bytes remain to be translated. Condition code 2 is set if no more argument bytes remain.

The contents of access register 1 always remain unchanged.

Access exceptions are recognized only for those bytes in the second operand which are actually required. Access exceptions are not recognized for those bytes in the first operand which are to the right of the first byte for which a nonzero function byte is obtained.

Resulting Condition Code:

- 0 All function bytes zero
- 1 Nonzero function byte; first-operand field not exhausted
- 2 Nonzero function byte; first-operand field exhausted
- 3 --

Program Exceptions:

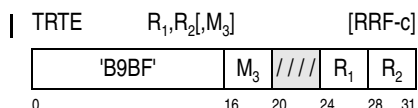
- Access (fetch, operands 1 and 2)

Programming Notes:

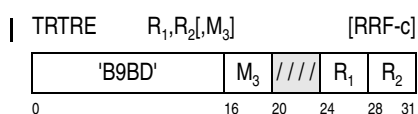
1. An example of the use of the TRANSLATE AND TEST instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. TRANSLATE AND TEST may be used to scan the first operand for characters with special meaning. The second operand, or list, is set up with all-zero function bytes for those characters

to be skipped over and with nonzero function bytes for the characters to be detected.

TRANSLATE AND TEST EXTENDED



TRANSLATE AND TEST REVERSE EXTENDED



The argument characters of the first operand are used to select function codes from a function-code table designated by general register 1. For TRANSLATE AND TEST EXTENDED, the argument characters are processed in a left-to-right direction; for TRANSLATE AND TEST REVERSE EXTENDED, the argument characters are processed in a right-to-left direction. When a nonzero function code is selected, it is inserted in general register R_2 , the related argument address is placed in general register R_1 , and the first-operand length in general register $R_1 + 1$ is decremented by the number of bytes processed. The operation proceeds until a nonzero function code is encountered, the end of the first operand is reached, or a CPU-determined number of characters have been processed, whichever occurs first. The result is indicated in the condition code.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

General register $R_1 + 1$ contains the length of the first operand in bytes.

Operation of the instruction is subject to controls specified in the M_3 field, bits 16-19 of the instruction. The M_3 field has the following format:



The bits of the M field are defined as follows:

- **Argument-Character Control (A):** The A bit, bit 0 of the M_3 field, controls the size of the argument characters in the first operand. When the A bit is zero, the argument characters are one byte in length. When the A bit is one, the argument characters are two bytes in length.

When the A bit is one, the first-operand length in general register $R_1 + 1$ must specify an even number of bytes; otherwise, a specification exception is recognized.

- **Function-Code Control (F):** The F bit, bit 1 of the M_3 field, controls the size of the function codes in the function-code table designated by general register 1. When the F bit is zero, a function code is one byte in length. When the F bit is one, a function code is two bytes in length.
- **Argument-Character Limit (L):** The L bit, bit 2 of the M_3 field, controls whether argument characters with a value greater than 255 are used to select function codes. When the L bit is zero, argument character values are unlimited. When the L bit is one, an argument character with a value greater than 255 is not used to select a function code; rather, the function code is assumed to contain zeros.

When the A bit of the M_3 field is zero, the L bit is ignored.

- **Unassigned:** Bit 3 of the M_3 field is unassigned and should contain zero; otherwise, the program may not operate compatibly in the future.

Figure 7-297 summarizes the size of the function-code table based on the A, F, and L bits.

			Table Size
A bit	F bit	L bit	(bytes)
0	0	–	256
0	1	–	512
1	0	0	65,536
1	1	0	131,072
1	0	1	256
1	1	1	512

Explanation:

– Not applicable

Figure 7-297. Function-Code Table Size

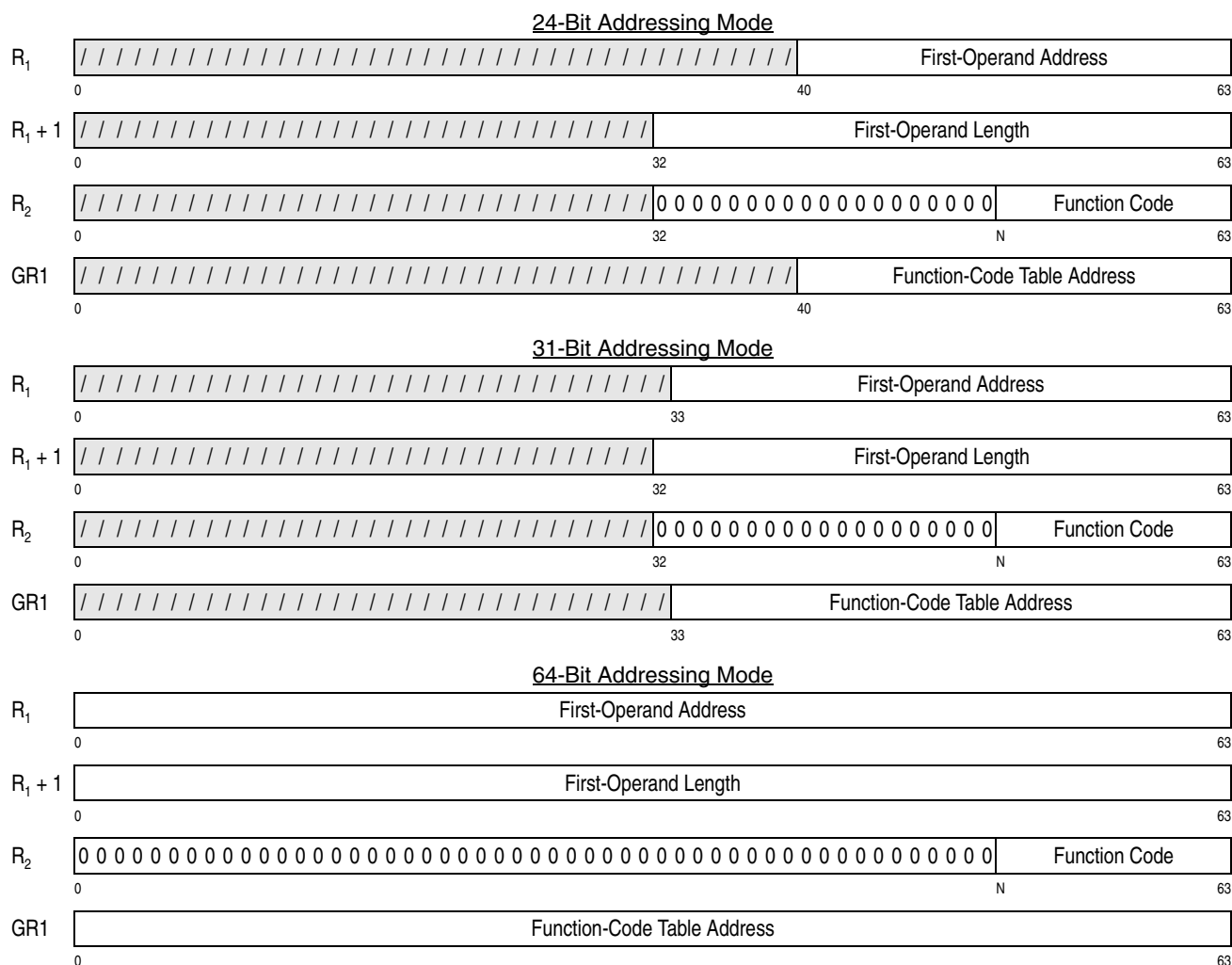
The location of the first argument character in the first operand is designated by the contents of general register R_1 . The location of the leftmost byte of the function-code table is designated by the contents of general register 1. In the 24-bit or 31-bit addressing mode, the number of bytes in the first-operand location is specified by the contents of bit positions 32-63 of general register $R_1 + 1$, and those contents are treated as a 32-bit unsigned binary integer. In the 64-bit addressing mode, the number of bytes in the first-operand location is specified by the entire contents of general register $R_1 + 1$, and those contents are treated as a 64-bit unsigned binary integer.

The handling of the argument-character address in general register R_1 is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-63 of the register constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the register constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The function-code table is treated as being on a doubleword boundary. The handling of the function-code-table address in general register 1 is dependent on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-60 of the register constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-60 of the register constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-60 of the register constitute the address. In all addressing modes, bit positions 61-63 are reserved and should contain zeros; otherwise, the program may not operate compatibly in the future.

When a nonzero function code is selected, it is inserted into either bits 56-63 or bits 48-63 of general register R_2 , depending on whether the F bit of the M_3 field is zero or one, respectively. In the 24-bit or 31-bit addressing mode, bits 0-31 of general register R_2 are unchanged, and bits 32-55 (F=0) or 32-47 (F=1) of the register are set to zero. In the 64-bit addressing mode, bits 0-55 (F=0) or 0-47 (F=1) are set to zero.

The contents of the registers just described are shown in Figure 7-298.



Explanation:

N 56 when the F-bit = 0; 48 when the F-bit = 1

Figure 7-298. Register Contents for TRANSLATE AND TEST EXTENDED and TRANSLATE AND TEST REVERSE EXTENDED

The argument characters of the first operand are selected one by one for processing, proceeding in a left-to-right direction for TRANSLATE AND TEST EXTENDED, or in a right-to-left direction for TRANSLATE AND TEST REVERSE EXTENDED. Depending on the A bit of the M₃ field, the argument characters are treated as either eight-bit or sixteen-bit unsigned binary integers, extended with zeros on the left.

When the F bit of the M_3 field is zero, the argument character is added to the function-code-table address in general register 1 to form the address of the selected 8-bit function code. When the F bit is one, the argument character, extended on the right

with a binary 0, is added to the function-code-table address in general register 1 to form the address of the selected 16-bit function code. These additions follow the rules for address arithmetic.

When both the A and L bits are one, and the value of the argument character is greater than 255, then the function-code table is not accessed. The function code is assumed to contain zero in this case.

When the selected function code contains zero, or when the function code is assumed to contain zero, processing continues with the next argument character in the first operand. The operation proceeds until a nonzero function code is selected. the first-operand

location is exhausted, or a CPU-determined number of first-operand bytes have been processed.

When the first-operand location is exhausted without having selected a nonzero function code, general register R_1 is either incremented or decremented by the first operand length in general register $R_1 + 1$; general register $R_1 + 1$ is set to zero; general register R_2 is set as though a function code of zero was selected; and condition code 0 is set. For TRANSLATE AND TEST EXTENDED, general register R_1 is incremented by the first operand length; for TRANSLATE AND TEST REVERSE EXTENDED, general register R_1 is decremented by the first operand length. In the 24-bit or 31-bit addressing mode, bits 0-31 of general register R_2 are unchanged, and bits 32-63 of the register are set to zeros; in the 64-bit addressing mode, bits 0-63 of general register R_2 are set to zeros.

When a nonzero function code is selected, the function code replaces bits 56-63 or bits 48-63 of general register R_2 , depending on whether the F bit is zero or one, respectively; depending on the addressing mode, the remaining bits in general register R_2 are set to zeros; the address of the argument character used to select the nonzero function code is placed in general register R_1 ; general register $R_1 + 1$ is decremented by the number of first-operand bytes processed prior to selecting the nonzero function byte; and condition code 1 is set. In the 24-bit or 31-bit addressing mode, bits 0-31 of general register R_2 are unchanged, and bits 32-55 ($F=0$) or 32-47 ($F=1$) are set to zero; in the 64-bit addressing mode, bits 0-55 ($F=0$) or 0-47 ($F=1$) are set to zeros.

When a CPU-determined number of bytes have been processed, general register R_1 is either incremented or decremented by the number of bytes in the first operand that were processed, general register $R_1 + 1$ is decremented by this number, and condition code 3 is set. For TRANSLATE AND TEST EXTENDED, general register R_1 is incremented by the number of bytes processed; for TRANSLATE AND TEST REVERSE EXTENDED, general register R_1 is decremented by the number of bytes processed. Condition code 3 may be set even when the first-operand location is exhausted or when the next argument character to be processed selects a nonzero function byte. In these cases, condition code 0 or 1 will be set when the instruction is executed again.

When the R_2 field designates the same register as register R_1 , the updated first-operand address is

placed in the register. When the R_2 field designates the same register as register $R_1 + 1$, the updated first-operand length is placed in the register.

When general register R_1 is updated in the 24-bit or 31-bit addressing mode, bits 32-39, in the 24-bit mode, or bit 32, in the 31-bit mode, may be set to zeros or may remain unchanged from their original values.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, and R_2 always remain unchanged.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

Access exceptions for the portion of the first operand beyond the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed. When the length of the first operand is zero, no access exceptions for the first operand are recognized.

Access exceptions for any byte of the function-code table specified by general register 1 may be recognized, even if not all bytes are used.

Special Conditions

A specification exception is recognized for any of the following conditions:

1. The R_1 field designates an odd-numbered register.
2. The A bit of the M_3 field is one and the first operand length in general register $R_1 + 1$ is odd.

Resulting Condition Code:

- | | |
|---|--|
| 0 | Entire first operand processed without selecting a nonzero function code |
| 1 | Nonzero function code selected |
| 2 | -- |
| 3 | CPU-determined number of bytes processed |

Program Exceptions:

- Access (fetch, operand 1 and the function-code table)
- Operation (if the parsing-enhancement facility is not installed)
- Specification

Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue processing. The program need not determine the number of bytes that were processed.
2. The storage-operand references of TRANSLATE AND TEST EXTENDED and TRANSLATE AND TEST REVERSE EXTENDED may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
3. The following example illustrates the use of TRANSLATE AND TEST EXTENDED to parse a buffer of 16-bit Unicode argument characters for delimiters such as those used in common markup languages. Because the Unicode representation of such delimiters is in the range of 0000 - 00FF hex, a limited function-code table is used (the L bit of the M₃ field is one).

In this example, the variable BUFFER is used to hold the 16-bit characters to be parsed. If a delimiting character is not found, control passes to the label NOTFND. If a delimiting character is found, the function code placed in general register R₂ is used as an index to the branch table that passes control to a specific processing routine (not shown).

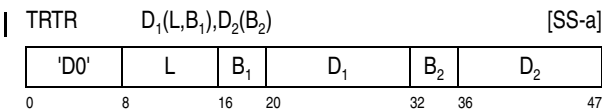
	LARL	1, TABLE	Point to func-code table.
	LARL	2, BUFFER	Point to buffer.
	LAY	3, L'BUFFER	Load length of buffer.
RETRY	TRTE	2, 15, B'1010'	Parse the buffer.
	BO	RETRY	Retry on CC3.
	B	BR_TBL(15)	Br. based on func. code:

BR_TBL	B	NOT_FND	- 00 (CC0, not found).
	B	NULL	- 04
	B	DOUBLE_QUOTE	- 08
	B	AMPERSAND	- 0C
	B	SINGLE_QUOTE	- 10
	B	SLASH	- 14
	B	LESS_THAN	- 18
	B	EQUALS	- 1C
	B	GREATER_THAN	- 20
	B	QUESTION_MARK	- 24
...			
NOT_FND	DS	0H	No delimiters found.
...			
DC		X'04,00,00,00,00,00,00,00'	00-07
DC		X'00,00,00,00,00,00,00,00'	08-0F
DC		X'00,00,00,00,00,00,00,00'	10-17
DC		X'00,00,00,00,00,00,00,00'	18-1F
DC		X'00,00,08,00,00,00,0C,10'	20-27
DC		X'00,00,00,00,00,00,00,14'	28-2F
DC		X'00,00,00,00,00,00,00,00'	30-37
DC		X'00,00,00,00,00,18,1C,20,24'	38-3F
DC		192X'00'	40-FF
BUFFER	DS	CL8192	Buffer of 16-bit chars.

Note, the branch to label NOT_FND does not result from a function-code-table entry of zeros. Rather, the branch is taken when the first operand is exhausted without selecting a nonzero function-code-table entry.

4. As an alternative to treating the selected function-code value as a table index (as shown in programming note 3, above), the function-code value may be used directly as a branch offset.
5. The performance of TRANSLATE AND TEST EXTENDED and TRANSLATE AND TEST REVERSE EXTENDED may be significantly slower if the program modifies the function-code table prior to execution of the instruction.
6. In the assembler syntax, the M₃ operand is considered to be optional. When the M₃ field is not coded, it is considered to contain zeros by the assembler.

TRANSLATE AND TEST REVERSE



The bytes of the first operand are used as eight-bit arguments to select function bytes from a list designated by the second-operand address. The first-operand address designates the rightmost byte of the first operand. The first nonzero function byte is inserted in general register 2, and the related argument address in general register 1.

The L field specifies the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding from right to left. The first operand remains unchanged in storage. Calculation of the address of the function byte is performed as in the TRANSLATE instruction. The function byte retrieved from the list is inspected for a value of zero.

When the function byte is zero, the operation proceeds with the preceding byte of the first operand. When the first-operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting condition code 0. The contents of general registers 1 and 2 remain unchanged.

When the function byte is nonzero, the operation is completed by inserting the function byte in general register 2 and the related argument address in general register 1. This address points to the argument byte last translated. The function byte replaces bits 56-63 of general register 2, and bits 0-55 of this register remain unchanged. In the 24-bit addressing mode, the address replaces bits 40-63 of general register 1, and bits 0-39 of this register remain unchanged. In the 31-bit addressing mode, the address replaces bits 33-63 of general register 1, and bits 0-32 of this register remain unchanged. In the 64-bit addressing mode, the address replaces bits 0-63 of general register 1.

When the function byte is nonzero, either condition code 1 or 2 is set, depending on whether the argument byte is the leftmost byte of the first operand. Condition code 1 is set if one or more argument bytes remain to be translated. Condition code 2 is set if no more argument bytes remain.

The contents of access register 1 always remain unchanged.

Access exceptions can be recognized for any byte in either the first or in the second operand.

Resulting Condition Code:

- 0 All function bytes zero
- 1 Nonzero function byte; first-operand field not exhausted
- 2 Nonzero function byte; first-operand field exhausted
- 3 --

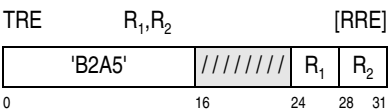
Program Exceptions:

- Access (fetch, operands 1 and 2)
- Operation (if the extended-translation facility 3 is not installed)

Programming Note:

TRANSLATE AND TEST REVERSE may be used to scan the first operand for characters with special meaning. The second operand, or list, is set up with all-zero function bytes for those characters to be skipped over and with nonzero function bytes for the characters to be detected.

TRANSLATE EXTENDED



The bytes of the first operand are compared to a test byte in general register 0 and, unless an equal comparison occurs, are used as eight-bit arguments to reference a 256-byte translation table designated by the second-operand address. Each function byte selected from the second operand replaces the corresponding argument in the first operand. The operation proceeds until a first-operand byte equal to the test byte is encountered, the end of the first operand is reached, or a CPU-determined number of bytes have been processed, whichever occurs first. The result is indicated in the condition code.

The R₁ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers R₁ and R₂, respectively. In the 24-bit or 31-bit addressing mode, the number of

bytes in the first-operand location is specified by the contents of bit positions 32-63 of general register $R_1 + 1$, and those contents are treated as a 32-bit unsigned binary integer. In the 64-bit addressing mode, the number of bytes in the first-operand location is specified by the entire contents of general register $R_1 + 1$, and those contents are treated as a 64-bit unsigned binary integer.

The handling of the addresses in general registers R_1 and R_2 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 consti-

tute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of the registers constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 constitute the address.

The test byte is in bit positions 56-63 of general register 0, and the contents of bit positions 0-55 of this register are ignored.

The contents of the registers just described are shown in Figure 7-299.

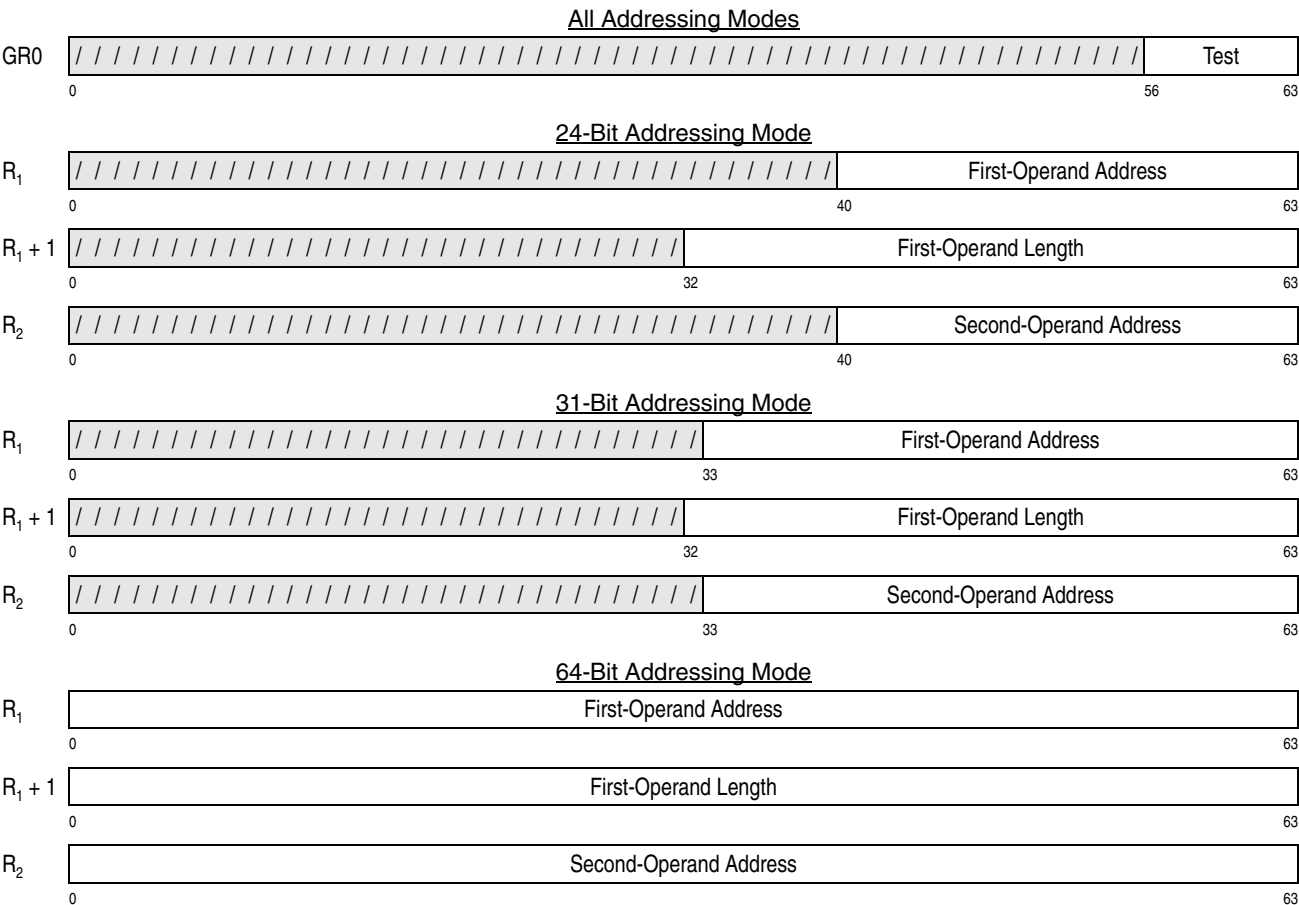


Figure 7-299. Register Contents for TRANSLATE EXTENDED

The bytes of the first operand are selected one by one for translation, proceeding left to right. Each argument byte is first compared to the test byte in general register 0. If the result is an equal comparison, the operation is completed. If the argument byte is not equal to the test byte, the argument byte is added to the initial second-operand address. The

addition is performed following the rules for address arithmetic, with the argument byte treated as an eight-bit unsigned binary integer and extended with zeros on the left. The sum is used as the address of the function byte, which then replaces the original argument byte. The second operand is not altered unless an overlap occurs.

The operation proceeds until a first-operand byte equal to the test byte is encountered, the first-operand location is exhausted, or a CPU-determined number of first-operand bytes have been processed.

When the first-operand location is exhausted without finding a byte equal to the test byte, condition code 0 is set. When a first-operand byte equal to the test byte is encountered, condition code 1 is set. When a CPU-determined number of bytes have been processed, condition code 3 is set. Condition code 3 may be set even when the first-operand location is exhausted or when the next byte to be processed is equal to the test byte. In these cases, condition code 0 or 1, respectively, will be set when the instruction is executed again.

If the operation is completed with condition code 0, the contents of general register R_1 are incremented by the contents of general register $R_1 + 1$, and then the contents of general register $R_1 + 1$ are set to zero. If the operation is completed with condition code 1, the contents of general register $R_1 + 1$ are decremented by the number of bytes processed before the first-operand byte equal to the test byte was encountered, and the contents of general register R_1 are incremented by the same number, so that general register R_1 contains the address of the equal byte. If the operation is completed with condition code 3, the contents of general register $R_1 + 1$ are decremented by the number of bytes processed, and the contents of general register R_1 are incremented by the same number, so that the instruction, when reexecuted, resumes at the next byte to be processed. When general register R_1 is updated in the 24-bit or 31-bit addressing mode, bits 32-39 of it, in the 24-bit mode, or bit 32, in the 31-bit mode, may be set to zeros or may remain unchanged from their original values.

In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 and $R_1 + 1$ always remain unchanged.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

When the R_2 register is the same register as the R_1 or $R_1 + 1$ register, the results are unpredictable.

When R_1 or R_2 is zero, the results are unpredictable.

When the second operand overlaps the first operand, the results are unpredictable.

Access exceptions for the portion of the first operand to the right of the last byte processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last byte processed.

Access exceptions for all 256 bytes of the second operand may be recognized, even if not all bytes are used.

Access exceptions are not recognized if the R_1 field is odd. When the length of the first operand is zero, no access exceptions for the first operand are recognized.

Resulting Condition Code:

- 0 Entire first operand processed without finding a byte equal to the test byte
- 1 First-operand byte is equal to the test byte
- 2 --
- 3 CPU-determined number of bytes processed

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Specification

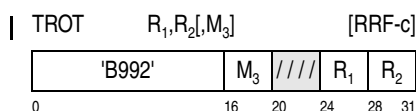
Programming Notes:

1. When condition code 3 is set, the program can simply branch back to the instruction to continue the translation. The program need not determine the number of bytes that were translated.
2. The instruction can improve performance by being used in place of a TRANSLATE AND TEST instruction that locates an escape character, followed by a TRANSLATE instruction that translates the bytes preceding the escape character.
3. The storage-operand references of TRANSLATE EXTENDED may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)

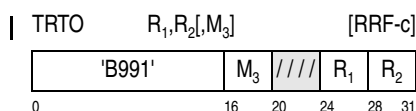
TRANSLATE ONE TO ONE



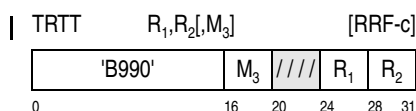
TRANSLATE ONE TO TWO



TRANSLATE TWO TO ONE



TRANSLATE TWO TO TWO



The characters of the second operand are used as arguments to select function characters from a translation table designated by the address in general register 1.

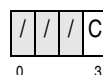
When the ETF2-enhancement facility is not installed, or when the test-character-comparison control is zero, each function character selected from the translation table is compared to a test character in general register 0, and, unless an equal comparison occurs, is placed at the first-operand location. The operation proceeds until a selected function character equal to the test character is encountered, the end of the second operand is reached, or a CPU-determined number of characters have been processed, whichever occurs first.

When the ETF2-enhancement facility is installed and the test-character-comparison control is one, test-character comparison is not performed. Each function character selected from the translation table is placed at the first operation location. The operation proceeds until the end of the second operand is

reached, or a CPU-determined number of characters have been processed, whichever occurs first.

The result is indicated in the condition code.

When the ETF2-enhancement facility is installed, the M_3 field has the following format:



The bits of the M_3 field are defined as follows:

- **Unassigned:** Bits 0-2 are unassigned and should contain zeros; otherwise, the program may not operate compatibly in the future.
- **Test-Character-Comparison Control (C):** The C bit, bit 3 of the M_3 field, controls test-character comparison. When the C bit is zero, test-character comparison is performed. When the C bit is one, test-character comparison is not performed.

When the ETF2-enhancement facility is not installed, the M_3 field is ignored.

The lengths of the operand and test characters are as follows:

- For TRANSLATE ONE TO ONE, the second-operand, first-operand, and test characters are single bytes.
- For TRANSLATE ONE TO TWO, the second-operand characters are single bytes, and the first-operand and test characters are double bytes.
- For TRANSLATE TWO TO ONE, the second-operand characters are double bytes, and the first-operand and test characters are single bytes.
- For TRANSLATE TWO TO TWO, the second-operand, first-operand, and test characters are double bytes.

For TRANSLATE ONE TO ONE and TRANSLATE TWO TO ONE, the test character is in bit positions 56-63 of general register 0. For TRANSLATE ONE TO TWO and TRANSLATE TWO TO TWO, the test character is in bit positions 48-63 of general register 0.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers R_1 and R_2 , respectively. In the 24-bit or 31-bit addressing mode, the number of bytes in the second-operand location is specified by the contents of bit positions 32-63 of general register $R_1 + 1$, and those contents are treated as a 32-bit unsigned binary integer. In the 64-bit addressing mode, the number of bytes in the second-operand location is specified by the contents of bit positions 0-63 of general register $R_1 + 1$, and those contents are treated as a 64-bit unsigned binary integer. The length of the first-operand location is considered to be the same as that of the second operand for TRANSLATE ONE TO ONE and TRANSLATE TWO TO TWO, twice that for TRANSLATE ONE TO TWO, and one half that for TRANSLATE TWO TO ONE.

For TRANSLATE TWO TO ONE and TRANSLATE TWO TO TWO, the length in general register $R_1 + 1$ must be an even number of bytes; otherwise, a specification exception is recognized.

The translation table is treated as being on a double-word boundary for TRANSLATE ONE TO ONE and TRANSLATE ONE TO TWO. For TRANSLATE TWO TO ONE and TRANSLATE TWO TO TWO, the translation table is treated as follows:

- When the ETF2-enhancement facility is not installed, the translation table is treated as being on a 4K-byte boundary.
- When the ETF2-enhancement facility is installed, the translation table is treated as being on a doubleword boundary.

The rightmost bits of the register that are not used to form the address, which are bits 61-63 in the double-word case and bits 52-63 in the 4K-byte case, are ignored but should contain zeros; otherwise, the program may not operate compatibly in the future.

The handling of the addresses in general registers R_1 , R_2 , and 1 is dependent on the addressing mode.

In the 24-bit addressing mode, the contents of bit positions 40-63 of general registers R_1 and R_2 and 40-60 or 40-51 of 1 constitute the address, and the

contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of registers R_1 and R_2 and 33-60 or 33-51 of 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of registers R_1 and R_2 and 0-60 or 0-51 of 1 constitute the address.

The contents of the registers just described are shown in Figure 7-300 on page 7-329.

In the access-register mode, the contents of access registers R_1 , R_2 , and 1 are used for accessing the first operand, second operand, and translation table, respectively.

The length of the translation table designated by the address contained in general register 1 is as follows:

- For TRANSLATE ONE TO ONE, the translation-table length is 256 bytes; each of the 256 function characters is a single byte.
- For TRANSLATE ONE TO TWO, the translation-table length is 512 bytes; each of the 256 function characters is a double byte.
- For TRANSLATE TWO TO ONE, the translation-table length is 65,536 (64K) bytes; each of the 64K function characters is a single byte.
- For TRANSLATE TWO TO TWO, the translation-table length is 131,072 (128K) bytes; each of the 64K function characters is a double byte.

The characters of the second operand are selected one by one for translation, proceeding left to right. Each argument character is added to the initial translation-table address. The addition is performed following the rules for address arithmetic, with the argument character treated as follows:

- For TRANSLATE ONE TO ONE, the argument character is treated as an eight-bit unsigned binary integer extended on the left with 56 zeros.
- For TRANSLATE ONE TO TWO, the argument character is treated as an eight-bit unsigned binary integer extended on the right with a zero and on the left with 55 zeros.
- For TRANSLATE TWO TO ONE, the argument character is treated as a 16-bit unsigned binary integer extended on the left with 48 zeros.

The sum is used as the address of the function character.

When the ETF2-enhancement facility is not installed, or when the test-character-comparison control is zero, processing is as follows. Each function character selected as described above is first compared to the test character in general register 0. If the result is an equal comparison, the operation is completed. If the function character is not equal to the test character, the function character is placed in the next available character position in the first operand, that is, the first function character is placed at the beginning of the first-operand location, and each successive function character is placed immediately to the right of the preceding character. The second operand and the translation table are not altered unless an overlap occurs.

The operation proceeds until a selected function character equal to the test character is encountered, the second-operand location is exhausted, or a CPU-determined number of second-operand characters have been processed.

When the ETF2-enhancement facility is installed and the test-character-comparison control is one, processing is as described above, except that no test-character comparison is performed.

When a selected function character equal to the test character is encountered, condition code 1 is set. When the second-operand location is exhausted without finding a selected function character equal to the test character, condition code 0 is set. When a CPU-determined number of characters have been processed, condition code 3 is set. Condition code 3 may be set even when the next character to be processed results in a function character equal to the test character or when the second-operand location is exhausted. In these cases, condition code 1 or 0, respectively, will be set when the instruction is executed again. When the ETF2-enhancement facility is installed and the test-character-comparison control is one, condition code 1 does not apply.

If the operation is completed with condition code 0, the contents of general register R_2 are incremented by the contents of general register $R_1 + 1$, and the contents of general register R_1 are incremented as follows:

- For TRANSLATE ONE TO ONE and TRANSLATE TWO TO TWO, the same as for general register R_2 .
- For TRANSLATE ONE TO TWO, by twice the amount for general register R_2 .
- For TRANSLATE TWO TO ONE, by one half the amount for general register R_2 .

The contents of general register $R_1 + 1$ are then set to zero.

If the operation is completed with condition code 1, the contents of general register $R_1 + 1$ are decremented by the number of second-operand bytes processed before the character that selected a function character equal to the test character was encountered, and the contents of general register R_2 are incremented by the same number, so that general register R_2 contains the address of the character that selected a function character equal to the test character. The contents of general register R_1 are incremented by the same, twice, or one half the number, as described above for condition code 0.

If the operation is completed with condition code 3, the contents of general register $R_1 + 1$ are decremented by the number of second-operand bytes processed, and the contents of general register R_2 are incremented by the same number, so that the instruction, when reexecuted, contains the address of the next character to be processed. The contents of general register R_1 are incremented by the same, twice, or one half the number, as described above for condition code 0.

When general registers R_1 and R_2 are updated in the 24-bit or 31-bit addressing mode, the bits in bit positions 32-39 of them that are not part of the address may be set to zeros or may remain unchanged from their original values. In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general registers R_1 , $R_1 + 1$, and R_2 always remain unchanged.

The contents of general registers 0 and 1 remain unchanged.

The amount of processing that results in the setting of condition code 3 is determined by the CPU on the basis of improving system performance, and it may be a different amount each time the instruction is executed.

During instruction execution, CPU retry may result in condition code 3 being set with possibly incorrect data having been stored in the first operand location at or to the right of the location designated by the final address in general register R_1 . The amount of data stored depends on the operation and the point in time at which CPU retry occurred. In all cases, the storing will occur again, with correct data stored, when the instruction is executed again to continue processing the same operands.

When any of the following conditions are true, the results are unpredictable:

- Either the R_1 or $R_1 + 1$ registers are the same register as the R_2 register.
- The R_1 register is register 0,
- The R_2 register is register 0, and either the ETF2-enhancement facility is not installed, or the facility is installed and the C bit is zero.
- The R_2 register is register 1.
- Any of the first and second operands and the translation table overlaps another of them.

Access exceptions for the portion of the first or second operand to the right of the last character processed may or may not be recognized. For an operand longer than 4K bytes, access exceptions are not recognized for locations more than 4K bytes beyond the last character processed.

Access exceptions for all characters of the translation table may be recognized even if not all characters are used.

Access exceptions are not recognized if the R_1 field is odd. When the length of the second operand is zero, no access exceptions for the first or second operand are recognized, and access exceptions for the translation table may or may not be recognized.

Resulting Condition Code:

- 0 Entire second operand processed; if test-character comparison was performed, no resulting function character was equal to the test character
- 1 Second-operand character found resulting in a function character equal to the test character

(applicable only when test-character comparison is performed)

- 2 --
- 3 CPU-determined number of characters processed

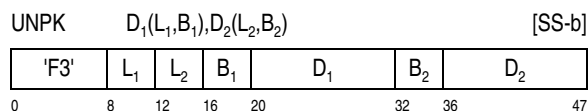
Program Exceptions:

- Access (fetch, operand 2 and translation table; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

Programming Notes:

1. These instructions differ from the TRANSLATE EXTENDED instruction by having the following attributes:
 - Depending on the instruction used, the sets of argument characters and function characters each can contain single-byte or double-byte characters.
 - The test character is compared to a resulting function character instead of to an argument character.
 - The argument (source) and function (destination) operands are different operands.
2. When condition code 3 is set, the program can simply branch back to the instruction to continue the translation. The program need not determine the number of characters that were translated.
3. The storage operand references of these instructions may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)

UNPACK



The format of the second operand is changed from signed-packed-decimal to zoned, and the result is placed at the first-operand location. The signed-packed-decimal and zoned formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the signed-packed-decimal format. Its digits and sign are placed unchanged in the first-operand location, using the zoned format. Zone bits with coding of 1111 are supplied for all bytes except the rightmost byte, the zone of which receives the sign of the second operand. The sign and digits are not checked for valid codes.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first-operand field is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if the operands were processed one byte at a time and as if the first result byte were stored immediately after fetching the first operand byte. The entire rightmost second-operand byte is used in forming the first result byte. For the remainder of the field, information for two result bytes is obtained from a single second-operand byte, and execution proceeds as if the leftmost four bits of the byte were to remain available for the next result byte and need not be refetched. Thus, the result is as if two result bytes were to be stored immediately after fetching a single operand byte.

Condition Code: The code remains unchanged.

Program Exceptions:

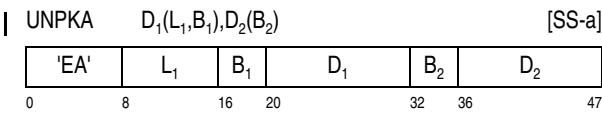
- Access (fetch, operand 2; store, operand 1)

Programming Notes:

1. An example of the use of the UNPACK instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. A field that is to be unpacked can be destroyed by improper overlapping. To save storage space for unpacking by overlapping the operands, the rightmost byte of the first operand must be to the right of the rightmost byte of the second operand by the number of bytes in the second operand minus 2. If only one or two bytes are to be unpacked, the rightmost bytes of the two operands may coincide.

3. The storage-operand references of UNPACK may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)

UNPACK ASCII



The format of the second operand is changed from signed-packed-decimal to ASCII, and the result is placed at the first-operand location. The signed-packed-decimal format is described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the signed-packed-decimal format. Its digits are converted to ASCII characters by extending them on the left with 0011 binary, and the ASCII characters are then placed at the first operand location. The digits are not checked for valid codes.

The sign of the second operand is not transferred to the first operand but is checked for validity and determines the condition code. If the sign is 1010, 1100, 1110 or 1111 binary (plus), condition code 0 is set. If the sign is 1011 or 1101 binary (minus), condition code 1 is set. If the sign is not one of the codes for plus or minus, condition code 3 is set.

The converted last digit is placed in the rightmost byte position of the result field, and the other converted digits are placed adjacent to the last and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left.

The length of the second operand is 16 bytes. The second operand consists of 31 digits and a sign.

The length of the first operand is designated by the contents of the L_1 field. The first-operand length must not exceed 32 bytes (L_1 must be less than or equal to 31); otherwise, a specification exception is recognized.

If the first operand is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions

for the unused portion of the second operand may or may not be indicated.

When the length of the first operand is 32 bytes, the leftmost byte is set to ASCII zero, 30 hex.

The results are unpredictable if the first and second operands overlap in any way.

As observed by other CPUs and by channel programs, the first operand is not necessarily stored into in any particular order.

Resulting Condition Code:

- 0 Sign is plus
- 1 Sign is minus
- 2 --
- 3 Sign is invalid

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

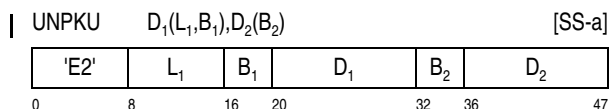
Programming Note:

1. The following example illustrates the use of the instruction to unpack to ASCII digits:

```
ASDIGITS DS CL31
PKDIGITS DS 0PL16
          DC X'1234567890'
          DC X'1234567890'
          DC X'1234567890'
          DC X'1C'
          :
          UNPKA ASDIGITS(31),PKDIGITS
```

2. The storage-operand references of UNPACK ASCII may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
3. The UNPACK ASCII and UNPACK UNICODE instructions set condition code 0 to indicate a positive sign, and these instructions provide no indication of a zero result. In all other instructions that indicate a signed result, condition code 0 indicates a result of zero, and condition code 2 indicates a positive result.

UNPACK UNICODE



The format of the second operand is changed from signed-packed-decimal to Unicode Basic Latin, and the result is placed at the first-operand location. The signed-packed-decimal format is described in Chapter 8, “Decimal Instructions.”

The second operand is treated as having the signed-packed-decimal format. Its digits are converted to two-byte Unicode characters by extending them on the left with 000000000011 binary (003 hex), and the Unicode characters are then placed at the first operand location. The digits are not checked for valid codes. The sign of the second operand is not transferred to the first operand but is checked for validity and determines the condition code. If the sign is 1010, 1100, 1110 or 1111 binary (plus), condition code 0 is set. If the sign is 1011 or 1101 binary (minus), condition code 1 is set. If the sign is not one of the codes for plus or minus, condition code 3 is set.

The converted last digit is placed in the rightmost character position of the result field, and the other converted digits are placed adjacent to the last and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left.

The length of the second operand is 16 bytes; the second operand consists of 31 digits and a sign.

The length of the first operand is designated by the contents of the L_1 field. The first-operand length must not exceed 32 characters or 64 bytes (L_1 must be less than or equal to 63 and must be odd); otherwise a specification exception is recognized.

If the first operand is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the length of the first operand is 32 characters, the leftmost character is set to Unicode Basic Latin zero, 0030 hex.

The results are unpredictable if the first and second operands overlap in any way.

As observed by other CPUs and by channel programs, the first operand is not necessarily stored into in any particular order.

Resulting Condition Code:

- 0 Sign is plus
- 1 Sign is minus
- 2 --
- 3 Sign is invalid

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Operation (if the extended-translation facility 2 is not installed)
- Specification

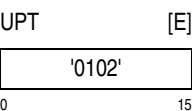
Programming Notes:

- 1. The following example illustrates the use of the instruction to unpack to European numbers:

```
UNDIGITS DS CL62
PKDIGITS DS 0PL16
          DC X'1234567890'
          DC X'1234567890'
          DC X'1234567890'
          DC X'1C'
          UNPKU UNDIGITS(62),PKDIGITS
```

- 2. The storage-operand references of UNPACK UNICODE may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)
- 3. The UNPACK ASCII and UNPACK UNICODE instructions set condition code 0 to indicate a positive sign, and these instructions provide no indication of a zero result. In all other instructions that indicate a signed result, condition code 0 indicates a result of zero, and condition code 2 indicates a positive result.

UPDATE TREE



The nodes of a tree in storage are examined successively on a path toward the base of the tree, and contents of general register 0, conceptually followed on the right by contents of general register 1, are conditionally interchanged with the contents of the nodes so as to give a unique maximum logical value in general register 0. The first half of a node and general register 0 contain a codeword, which is for use in sort/merge algorithms.

If the codeword in general register 0 equals the codeword in a node, the contents of the node are placed in general registers 2 and 3.

General register 4 contains the base address of the tree, and general register 5 contains the index of a node whose parent node will be examined first.

In the access-register mode, access register 4 specifies the address space containing the tree.

This instruction may be interrupted between units of operation. The condition code is unpredictable if the instruction is interrupted.

The size of a node, the size of a codeword, and the participation of bits 0-31 of general registers 1-5 in the operation depend on the addressing mode. In the 24-bit or 31-bit addressing mode, a node is eight bytes, a codeword is four bytes, and bits 0-31 are ignored and remain unchanged. In the 64-bit addressing mode, a node is 16 bytes, a codeword is eight bytes, and bits 0-31 are used in and may be changed by the operation.

Operation in the 24-Bit or 31-Bit Addressing Mode

In the 24-bit or 31-bit addressing mode, the double-word nodes of a tree in storage are examined successively on a path toward the base of the tree, and the contents of bit positions 32-63 of general register 0, conceptually followed on the right by the contents of bit positions 32-63 of general register 1, are conditionally interchanged with the contents of the nodes so as to give a unique maximum logical value in bit positions 32-63 of general register 0.

Bit positions 32-63 of general register 4 contain the base address of the tree, and bit positions 32-63 of general register 5 contain the index of a node whose parent node will be examined first. The base address is eight less than the address of the root node of the tree. The initial contents of bit positions 32-63 of gen-

eral registers 4 and 5 must be a multiple of 8; otherwise, a specification exception is recognized.

A unit of operation begins by shifting the contents of bit positions 32-63 of general register 5 right logically one position and then setting bit 61 to zero. However, bits 32-63 of general register 5 remain unchanged if the execution of a unit of operation is nullified or suppressed. If after shifting and setting bit 61 to zero, bits 32-63 of general register 5 are all zeros, the instruction is completed, and condition code 1 is set; otherwise, the unit of operation continues.

Bit 32 of general register 0 is tested. If bit 32 of general register 0 is one, the instruction is completed, and condition code 3 is set.

If bit 32 of general register 0 is zero, the sum of bits 32-63 of general registers 4 and 5 is used as the intermediate value for normal operand address generation. The generated address is the address of a node in storage.

Bits 32-63 of general register 0 are logically compared with the contents of the first word of the currently addressed node. If the register operand is low, the contents of bit positions 32-63 of general registers 0 and 1 are interchanged with those of the node, and a unit of operation is completed. If the register operand is high, no additional action is taken, and the unit of operation is completed. If the compare values are equal, bit positions 32-63 of general register 2, conceptually followed on the right by bit positions 32-63 of general register 3, are loaded from the currently addressed node, the instruction is completed, and condition code 0 is set.

In those cases when the value in the first word of the node is less than or equal to the value in bit positions 32-63 of the register, the contents of the node remain unchanged. However, in some models, these contents may be fetched and subsequently stored back.

Operation in the 64-Bit Addressing Mode

In the 64-bit addressing mode, the quadword nodes of a tree in storage are examined successively on a path toward the base of the tree, and the contents of general register 0, conceptually followed on the right by the contents of general register 1, are conditionally interchanged with the contents of the nodes so as to give a unique maximum logical value in general register 0.

General register 4 contains the base address of the tree, and general register 5 contain the index of a node whose parent node will be examined first. The base address is 16 less than the address of the root node of the tree. The initial contents of general registers 4 and 5 must be a multiple of 16; otherwise, a specification exception is recognized.

A unit of operation begins by shifting the contents of general register 5 right logically one position and then setting bit 60 to zero. However, general register 5 remains unchanged if the execution of a unit of operation is nullified or suppressed. If after shifting and setting bit 60 to zero, the contents of general register 5 are zero, the instruction is completed, and condition code 1 is set; otherwise, the unit of operation continues.

Bit 0 of general register 0 is tested. If bit 0 of general register 0 is one, the instruction is completed, and condition code 3 is set.

If bit 0 of general register 0 is zero, the sum of the contents of general registers 4 and 5 is used as the intermediate value for normal operand address generation. The generated address is the address of a node in storage.

The contents of general register 0 are logically compared with the contents of the first doubleword of the currently addressed node. If the register operand is low, the contents of general registers 0 and 1 are interchanged with those of the node, and a unit of operation is completed. If the register operand is high, no additional action is taken, and the unit of operation is completed. If the compare values are equal, general registers 2 and 3 are loaded from the currently addressed node, the instruction is completed, and condition code 0 is set.

In those cases when the value in the first doubleword of the node is less than or equal to the value in the register, the contents of the node remain unchanged. However, in some models, these contents may be fetched and subsequently stored back.

Specifications Independent of Addressing Mode

Access exceptions are recognized only for one node at a time. Access exceptions, change-bit action, and PER storage alteration do not occur for subsequent nodes until the previous node has been successfully compared and updated, and they also do not occur if a specification-exception condition exists.

Resulting Condition Code:

- 0 Equal compare values at currently addressed node
- 1 No equal compare values found on path, or no comparison made
- 2 --
- 3 In 24-bit or 31-bit mode, bits 32-63 of general register 5 nonzero and bits 32-63 of general register 0 negative; in 64-bit mode, general register 5 nonzero and general register 0 negative

Program Exceptions:

- Access (fetch and store, nodes of tree)
- Specification

Programming Notes:

1. An example of the use of UPDATE TREE is given in “Sorting Instructions” in Appendix A, “Number Representation and Instruction-Use Examples.”
2. For use in sorting in the 24-bit or 31-bit addressing mode, when equal compare values have been found, the contents of bit positions 32-63 of general registers 1 and 3 can be appropriate (depending on the contents of the tree) for the subsequent execution of COMPARE AND FORM CODEWORD. The contents of bit positions 32-63 of general register 2, shifted right 16 bit positions, can be similarly appropriate, and they can provide for minimal recomparison of partially equal keys. The same applies in the 64-bit addressing mode except to the contents of bit positions 0-63 of the registers and with the contents of bit positions 0-63 of general register 2 shifted right 48 bit positions. Refer to “Sorting Instructions” on page A-53 for a discussion of trees and their use in sorting.
3. The program should avoid placing a nonzero value in bit positions 32-38 of general register 5 when in the 24-bit addressing mode. If any bit in bit positions 32-38 is a one, the nodes of the tree will not be examined successively.
4. When bits 32-63 of general register 0 are negative in the 24-bit or 31-bit addressing mode, or when bits 0-63 are negative in the 64-bit mode, and provided that the tree has been updated

properly previously, the node represented by general registers 0 and 1 either is the node or is equal to the node (equal keys) that would be selected if the unit of operation continued. In this case, ending the unit of operation and setting condition code 3 is a faster means of selecting an appropriate node because it does not require further examination and updating of the tree.

5. Setting condition code 3 provides improved performance when the replacement record is equal to the old winner and, more importantly (since the first case can be detected by means of the condition code of CFC), when the update path contains a negative codeword, indicating equality with the old winner.
6. In those cases when the codeword in the node is less than or equal to the codeword in general register 0, depending on the model, the contents of the node may be fetched and subsequently stored back. As a result, any of the following may occur for the storage location containing the node: a PER storage-alteration event may be recognized; a protection exception for storing may be recognized; and, provided no access exceptions exist, the change bit may be set to one. Because the contents of storage remain unchanged, the change bit may or may not be one when a PER storage-alteration event is recognized.
7. Special precautions should be taken when UPDATE TREE is made the target of an execute-type instruction. See the programming note concerning interruptible instructions under EXECUTE.
8. Further programming notes concerning interruptible instructions are included in “Interruptible Instructions” on page 5-20.
9. The storage-operand references for UPDATE TREE may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
10. Figure 7-301 on page 7-337 is a summary of the operation of UPDATE TREE in the 24-bit or 31-bit addressing mode, and Figure 7-302 on page 7-338 is a summary of the operation in the 64-bit addressing mode.

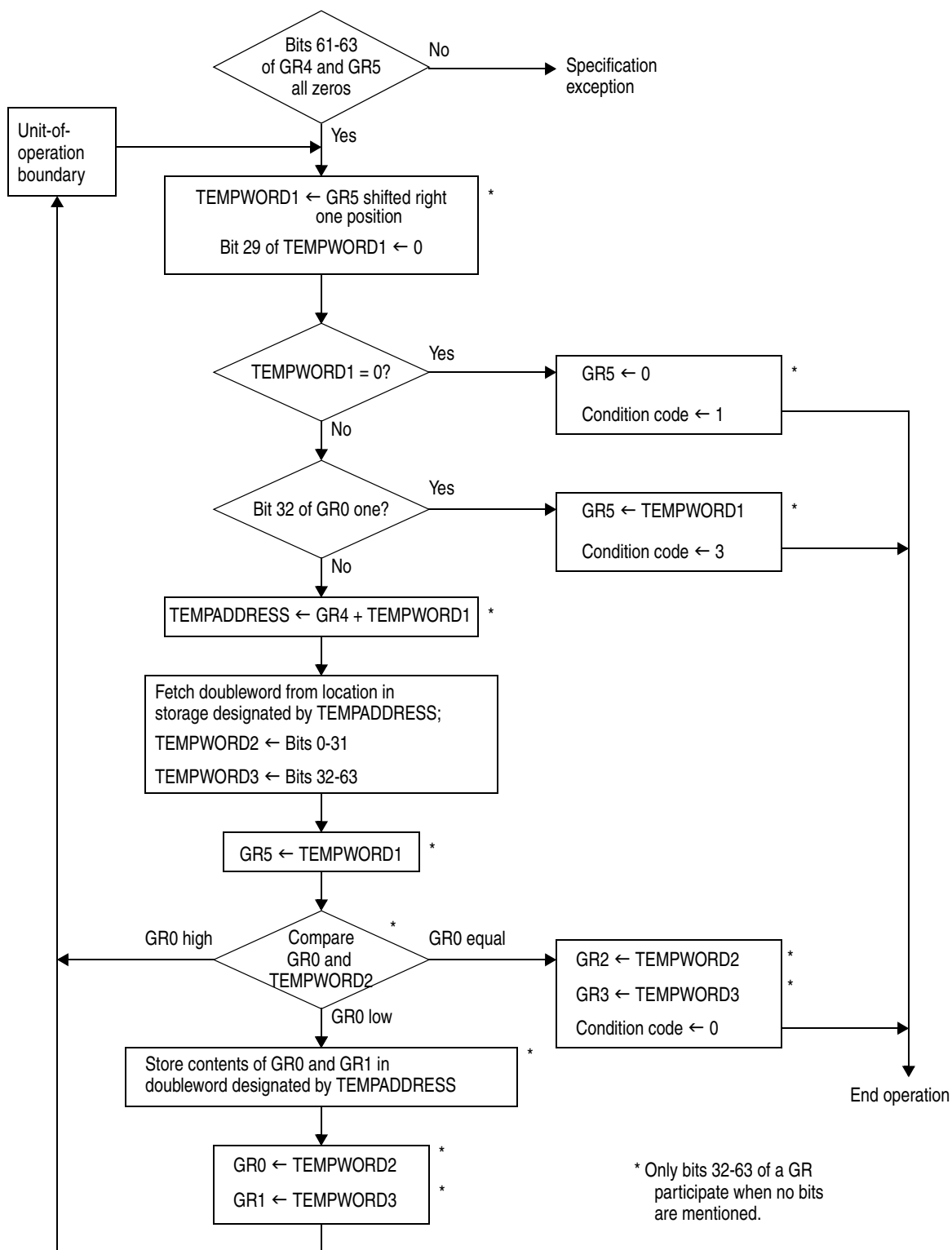


Figure 7-301. Execution of UPDATE TREE in the 24-Bit or 31-Bit Addressing Mode

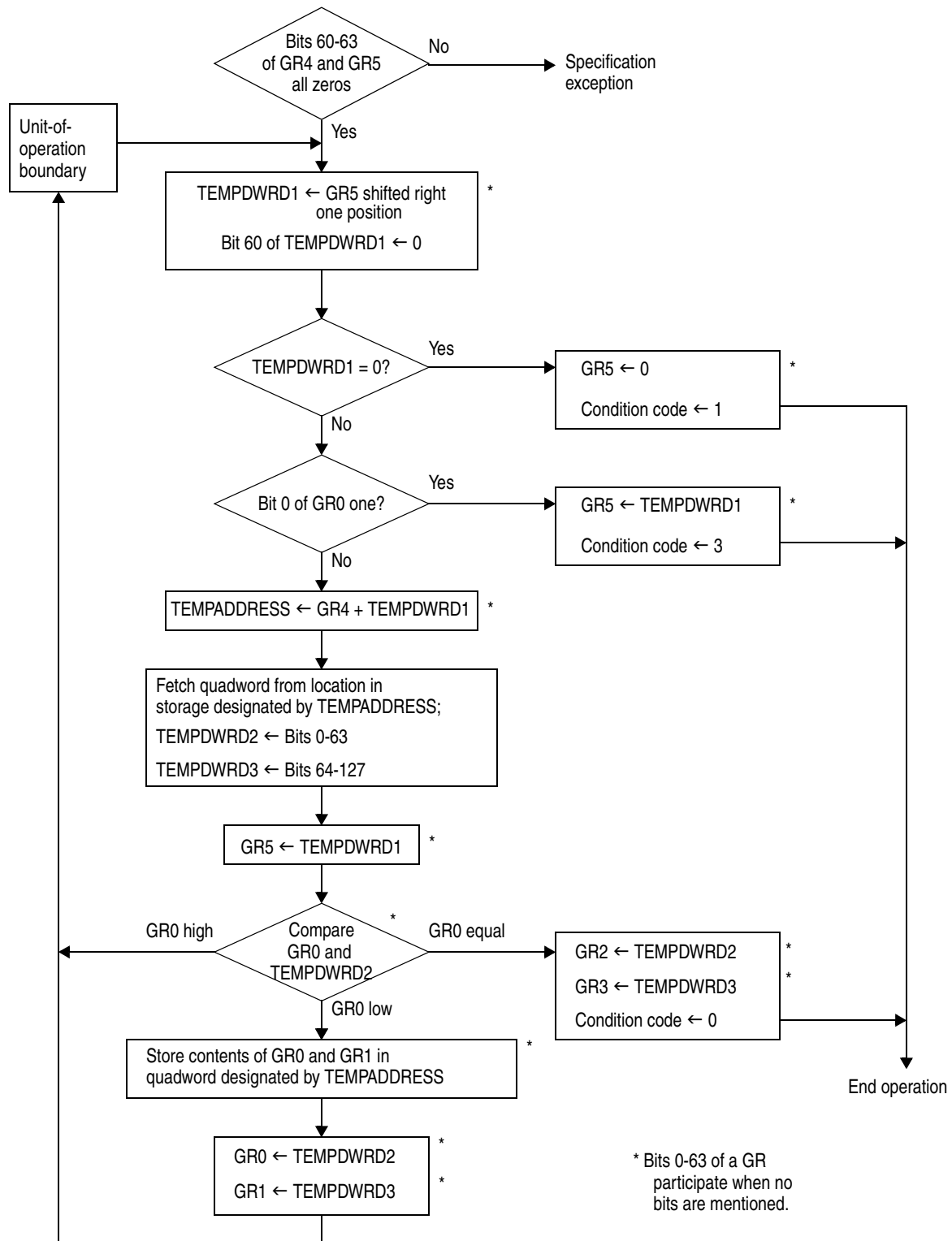


Figure 7-302. Execution of UPDATE TREE in the 64-Bit Addressing Mode

Protection of Cryptographic Key

When the message-security-assist-extension-3 facility is installed, two wrapping-key registers and two wrapping-key-verification-pattern registers are provided for each configuration. The two wrapping-key registers consist of a 192-bit DES wrapping-key (WK_d) register and a 256-bit AES wrapping-key (WK_a) register. The two wrapping-key-verification-pattern registers consist of a 192-bit DES wrapping-key-verification-pattern (WK_dVP) register and a 256-bit AES wrapping-key-verification-pattern (WK_aVP) register.

WK_d is used to protect user DES or TDES keys, and WK_dVP is used to identify the value of WK_d . WK_a is used to protect user AES keys, and WK_aVP is used to identify the value of WK_a . Whenever the contents of the WK_d or WK_a register are changed, the contents of the corresponding verification-pattern register are also changed.

Each time a clear reset is performed, a new set of wrapping keys and their associated verification patterns are generated. The contents of the two wrapping-key registers are kept internal to the model so that no program, including the operating system, can directly observe their clear value.

Programming Notes:

1. Upon condition code 1, the program should re-import the cryptographic key for encrypting it under the appropriate wrapping key before the operation is resumed.
2. Even though the architecture defines that a new set of wrapping keys are generated upon a clear reset, condition code 1 could occur at any time for message-security-assist functions using an encrypted key because the application may be relocated to another system with a different set of wrapping keys.

Symbols Used in Subsequent Descriptions

The following symbols are used in the subsequent description. For data-encryption-algorithm (DEA) functions, the DEA-key-parity bit in each byte of the DEA key is ignored, and the operation proceeds normally, regardless of the DEA-key parity of the key.

Further description of the data-encryption algorithm may be found in *Data Encryption Algorithm*, ANSI-X3.92.1981, American National Standard for Information Systems. Further description of the AES standard may be found in *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, dated November 26, 2001.

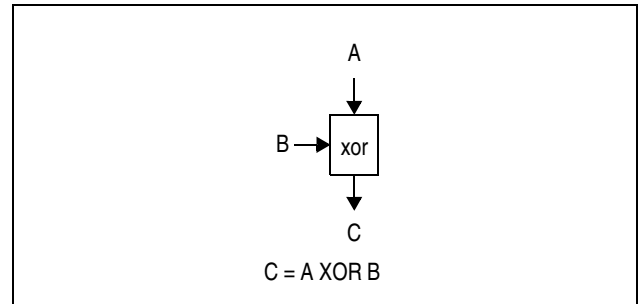


Figure 7-303. Symbol For Bit-Wise Exclusive OR

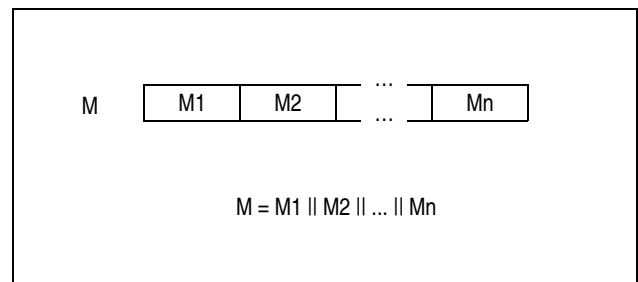


Figure 7-304. Symbol For Concatenation

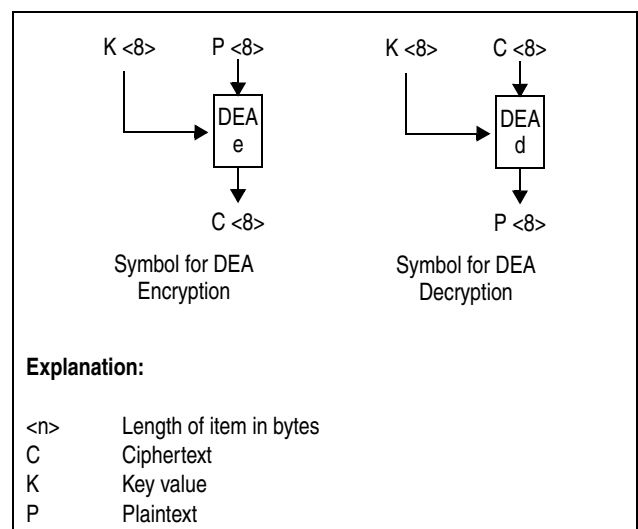


Figure 7-305. Symbols for DEA Encryption and Decryption

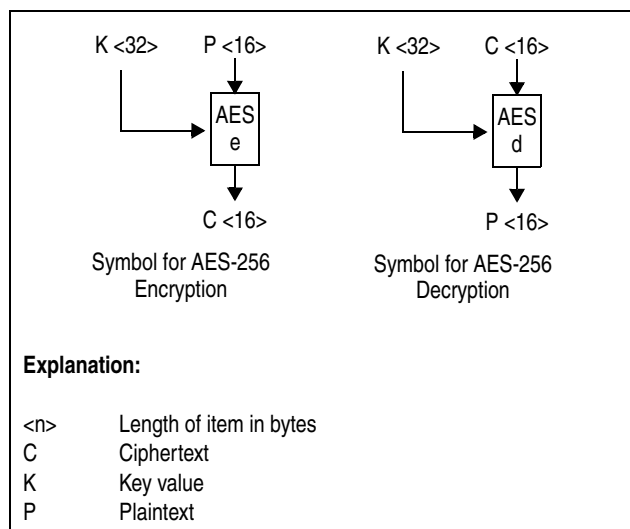


Figure 7-306. Symbols for AES-256 Encryption and Decryption

Protection of DES Keys

When the message-security-assist-extension-3 facility is installed, user DEA or TDEA keys may be protected under the 192-bit DEA wrapping key.

Figure 7-307 on page 7-340 describes encryption of a 64-bit DEA key, a 128-bit TDEA key, and a 192-bit TDEA key using the 192-bit DEA wrapping key.

Figure 7-308 on page 7-341 describes decryption of an encrypted 64-bit DEA key, an encrypted 128-bit TDEA key, and an encrypted 192-bit TDEA key using the 192-bit DEA wrapping key.

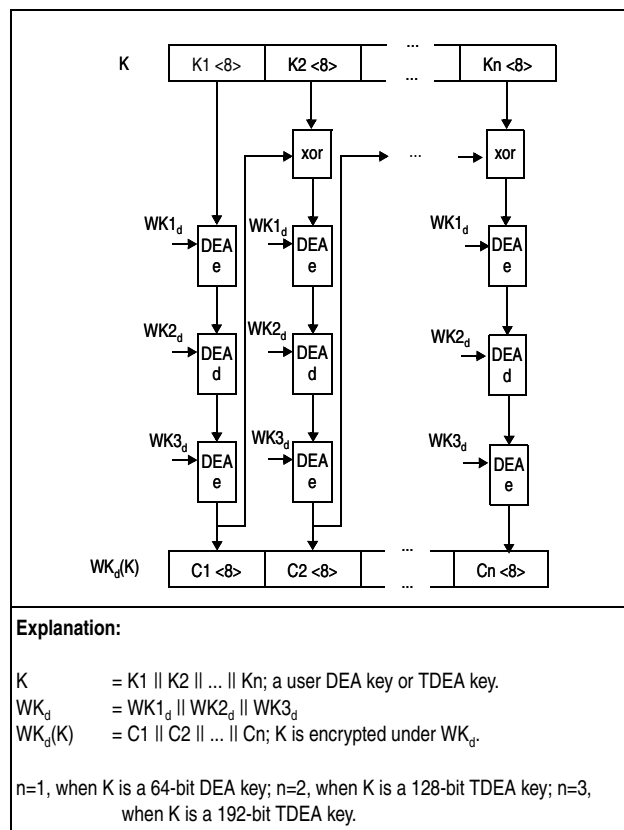


Figure 7-307. Encryption of DEA or TDEA Key Using WK_d

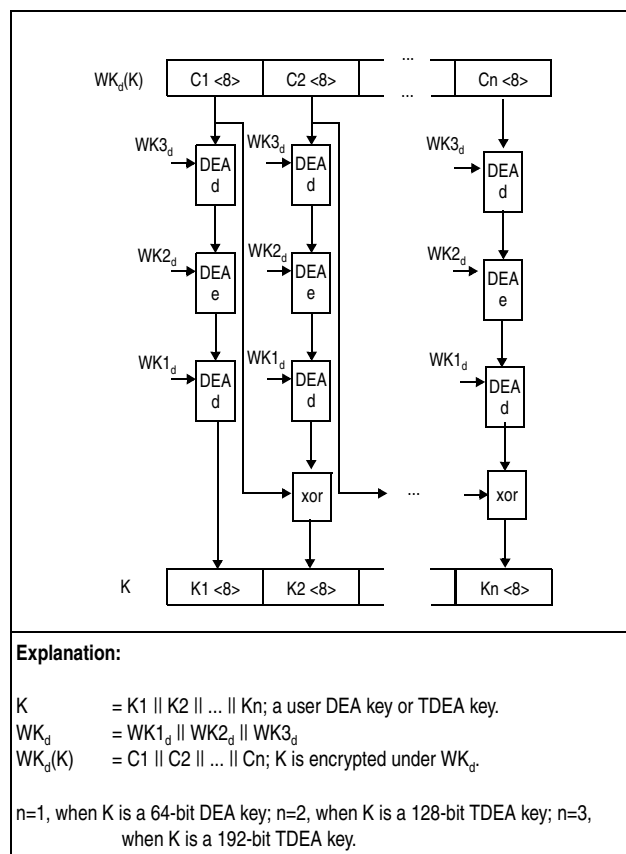


Figure 7-308. Decryption of DES or TDES Key Using WK_d

Protection of AES Keys

When the message-security-assist-extension-3 facility is installed, user AES keys may be protected under the 256-bit AES wrapping key.

Figure 7-309 on page 7-341 describes encryption of a 128-bit AES key using the 256-bit AES wrapping key. Figure 7-310 on page 7-341 describes decryption of an encrypted 128-bit AES key using the 256-bit AES wrapping key.

Figure 7-311 on page 7-342 describes encryption of a 192-bit AES key using the 256-bit AES wrapping key.

key. Figure 7-312 on page 7-342 describes decryption of an encrypted 192-bit AES key using the 256-bit AES wrapping key.

Figure 7-313 on page 7-342 describes encryption of a 256-bit AES key using the 256-bit AES wrapping key. Figure 7-314 on page 7-342 describes decryption of an encrypted 256-bit AES key using the 256-bit AES wrapping key.

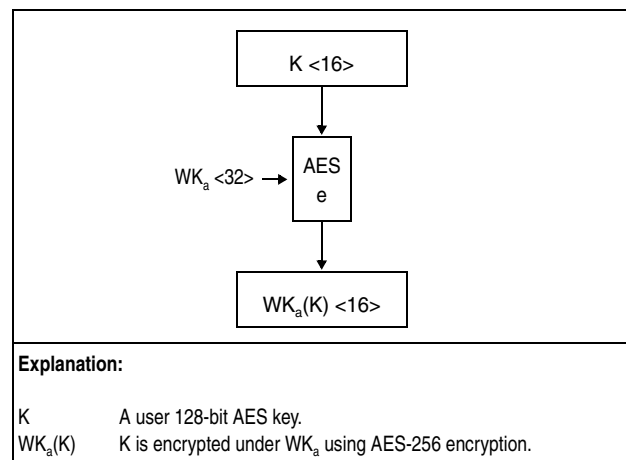


Figure 7-309. Encryption of 128-Bit AES Key Using WK_a

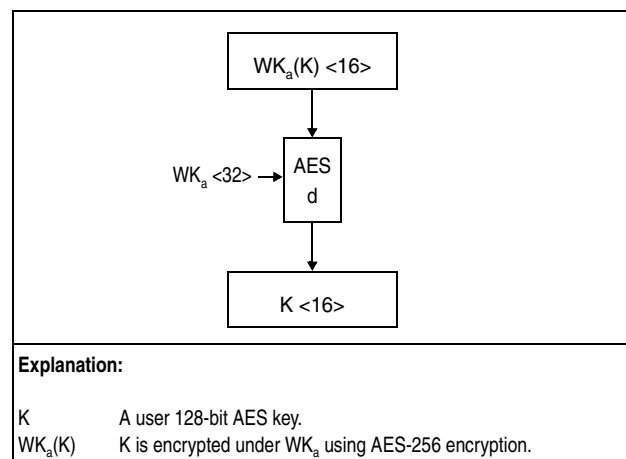


Figure 7-310. Decryption of 128-Bit AES Key Using WK_a

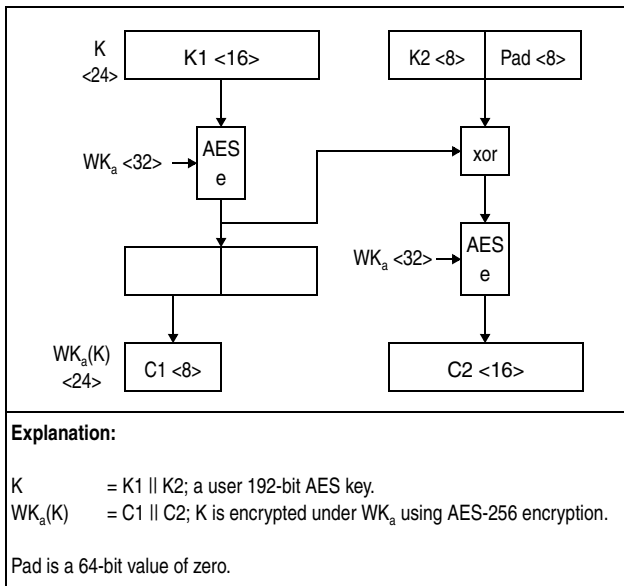


Figure 7-311. Encryption of 192-Bit AES Key Using WK_a

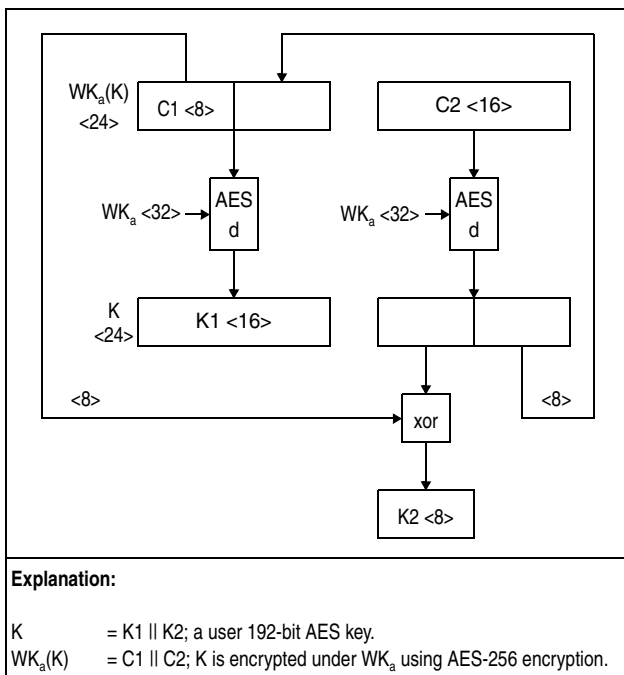


Figure 7-312. Decryption of 192-Bit AES Key Using WK_a

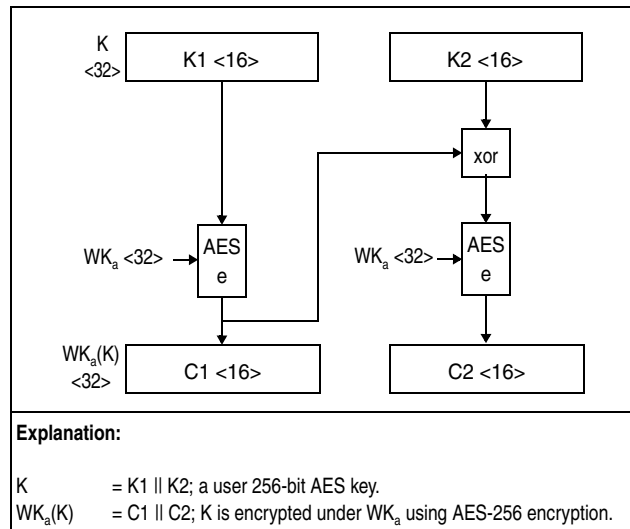


Figure 7-313. Encryption of 256-Bit AES Key Using WK_a

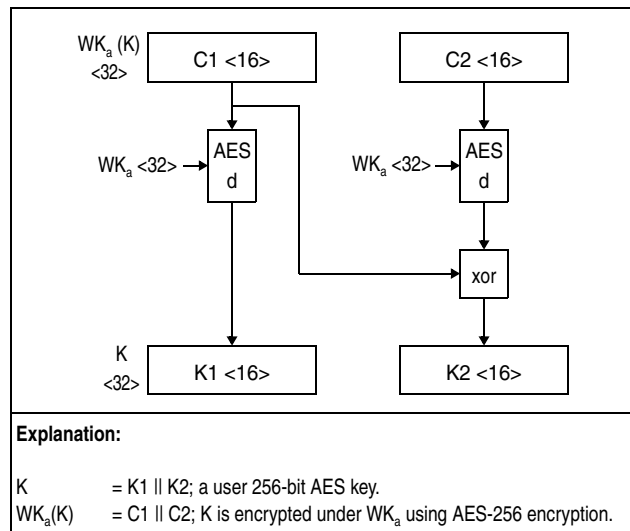


Figure 7-314. Decryption of 256-Bit AES Key Using WK_a

Chapter 8. Decimal Instructions

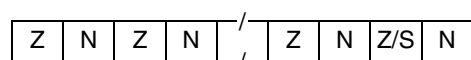
Decimal-Number Formats	8-1	Instructions	8-5
Zoned Format	8-1	ADD DECIMAL	8-6
Packed-Decimal Formats	8-1	COMPARE DECIMAL	8-6
Signed-Packed-Decimal Format	8-1	DIVIDE DECIMAL	8-7
Unsigned-Packed-Decimal Format	8-2	EDIT	8-7
Decimal Codes	8-2	EDIT AND MARK	8-10
Decimal Operations	8-3	MULTIPLY DECIMAL	8-11
Decimal-Arithmetic Instructions	8-3	SHIFT AND ROUND DECIMAL	8-12
Editing Instructions	8-4	SUBTRACT DECIMAL	8-13
Execution of Decimal Instructions	8-4	TEST DECIMAL	8-13
Other Instructions for Decimal Operands	8-4	ZERO AND ADD	8-13
Decimal-Operand Data Exception	8-4		

The decimal instructions of this chapter perform arithmetic and editing operations on decimal data. Additional operations on decimal data are provided by several of the instructions in Chapter 7, “General Instructions.” Also, several instructions in Chapter 20, “Decimal-Floating-Point Instructions” operate on decimal numbers in the packed-decimal format. Except for decimal-floating-point instructions, decimal operands always reside in storage, and most decimal instructions use the SS instruction format. Decimal operands occupy storage fields that can start on any byte boundary. For decimal-floating-point instructions, the packed-decimal operands are fixed length and occupy a general register, or general register pair.

Decimal-Number Formats

Decimal numbers may be represented in several formats, including zoned, signed-packed-decimal and unsigned-packed-decimal. These formats are of variable length; most instructions used to operate on these formats specify the length of their operands and results. Each byte of a format consists of a pair of four-bit codes; the four-bit codes include decimal-digit codes, sign codes, and a zone code.

Zoned Format



In the zoned format, the rightmost four bits of a byte are called the numeric bits (N) and normally consist

of a code representing a decimal digit. The leftmost four bits of a byte are called the zone bits (Z), except for the rightmost byte of a decimal operand, where these bits may be treated either as a zone or as a sign (S).

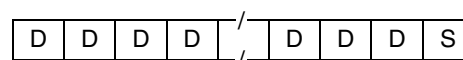
Decimal digits in the zoned format may be part of a larger character set, which includes also alphabetic and special characters. The zoned format is, therefore, suitable for input, editing, and output of numeric data in human-readable form. There are no decimal-arithmetic instructions which operate directly on decimal numbers in the zoned format; such numbers must first be converted to the signed-packed-decimal format.

The editing instructions produce a result of up to 256 bytes; each byte may be a decimal digit in the zoned format, a message byte, or a fill byte.

Packed-Decimal Formats

There are two packed-decimal formats, signed-packed, and unsigned-packed.

Signed-Packed-Decimal Format

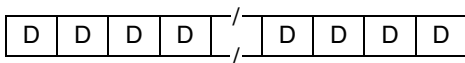


In the signed-packed-decimal format, each byte contains two decimal digits (D), except for the rightmost byte, which contains a sign (S) to the right of a decimal digit. Decimal arithmetic is performed with operands in the signed-packed-decimal format and

generates results in the signed-packed-decimal format.

The signed-packed-decimal operands and results of decimal-arithmetic instructions may be up to 16 bytes (31 digits and sign), except that the maximum length of a multiplier or divisor is eight bytes (15 digits and sign). In division, the sum of the lengths of the quotient and remainder may be from two to 16 bytes. The editing instructions can fetch as many as 256 decimal digits from one or more decimal numbers of variable length, each in the signed-packed-decimal format.

Unsigned-Packed-Decimal Format



In the unsigned-packed-decimal format, each byte contains two decimal digits (D) and there is no sign. The unsigned-packed-decimal format can be used as the source operand of the editing instructions, as the second operand of MOVE WITH OFFSET, and is also used by the decimal floating-point (DFP) instructions CONVERT TO UNSIGNED PACKED and CONVERT FROM UNSIGNED PACKED.

Decimal Codes

The decimal digits 0-9 have the binary encoding 0000-1001.

The preferred sign codes are 1100 for plus and 1101 for minus. These are the sign codes generated for the results of the decimal-arithmetic instructions and the CONVERT TO DECIMAL instruction. The instructions PACK ASCII and PACK UNICODE supply an implied positive sign (1100 binary). The DFP instructions CONVERT TO SIGNED PACKED include a plus sign-code selection bit. The plus sign can be encoded as either 1100 or 1111.

Alternate sign codes are also recognized as valid in the sign position: 1010, 1110, and 1111 are alternate codes for plus, and 1011 is an alternate code for minus. Alternate sign codes are accepted for any decimal source operand, but are not generated in the completed result of a decimal-arithmetic instruction or CONVERT TO DECIMAL. This is true even when an operand remains otherwise unchanged, such as when adding zero to a number. An alternate sign code is, however, left unchanged by MOVE NUMERICS, MOVE WITH OFFSET, MOVE ZONES, PACK, and UNPACK.

When an invalid sign or digit code is detected, a data exception is recognized. For the decimal-arithmetic instructions and CONVERT TO BINARY, and the decimal floating-point instructions CONVERT FROM SIGNED PACKED and CONVERT FROM UNSIGNED PACKED, the operation is suppressed.

For the editing instructions EDIT and EDIT AND MARK, an invalid sign code is not recognized. It is model dependent whether the operation is suppressed or terminated for a data exception due to an invalid digit code. No validity checking is performed by MOVE NUMERICS, MOVE WITH OFFSET, MOVE ZONES, PACK, PACK ASCII, PACK UNICODE, and UNPACK. The instructions UNPACK ASCII and UNPACK UNICODE do not check for invalid digit codes; invalid sign codes are recognized, but are not reported as a data exception, instead condition code 3 is set.

The zone code 1111 is generated in the left four bit positions of each byte representing a zone and a decimal digit in zoned-format results. Zoned-format results are produced by EDIT, EDIT AND MARK, and UNPACK. For EDIT and EDIT AND MARK, each result byte representing a zoned-format decimal digit contains the zone code 1111 in the left four bit positions and the decimal-digit code in the right four bit positions. For UNPACK, zone bits with a coding of 1111 are supplied for all bytes except the rightmost byte, the zone of which receives the sign.

The meaning of the decimal codes is summarized in Figure 8-1.

Code (Binary)	Recognized As	
	Digit	Sign
0000	0	Invalid
0001	1	Invalid
0010	2	Invalid
0011	3	Invalid
0100	4	Invalid
0101	5	Invalid
0110	6	Invalid
0111	7	Invalid
1000	8	Invalid
1001	9	Invalid
1010	Invalid	Plus
1011	Invalid	Minus
1100	Invalid	Plus (preferred)
1101	Invalid	Minus (preferred)
1110	Invalid	Plus
1111	Invalid	Plus (zone)

Figure 8-1. Summary of Digit and Sign Codes

Programming Note: Since 1111 is both the zone code and an alternate code for plus, unsigned (positive) decimal numbers may be represented in the zoned format with 1111 zone codes in all byte positions. The result of the PACK instruction converting such a number to the signed-packed-decimal format may be used directly as an operand for decimal instructions.

Decimal Operations

The decimal instructions in this chapter consist of two classes, the decimal-arithmetic instructions and the editing instructions.

Decimal-Arithmetic Instructions

The decimal-arithmetic instructions perform addition, subtraction, multiplication, division, comparison, and shifting.

Operands of the decimal-arithmetic instructions are in the signed-packed-decimal format and are treated as signed decimal integers. A decimal integer is represented in true form as an absolute value with a separate plus or minus sign. It contains an odd number of decimal digits, from one to 31, and the sign;

this corresponds to an operand length of one to 16 bytes.

A decimal zero normally has a plus sign, but multiplication, division, and overflow may produce a zero value with a minus sign. Such a negative zero is a valid operand and is treated as equal to a positive zero by COMPARE DECIMAL.

The lengths of the two operands specified in the instruction need not be the same. If necessary, the shorter operand is considered to be extended with zeros on the left. Results, however, cannot exceed the first-operand length as specified in the instruction.

When a carry or leftmost nonzero digits of the result are lost because the first-operand field is too short, the result is obtained by ignoring the overflow digits, condition code 3 is set, and, if the decimal-overflow mask bit is one, a program interruption for decimal overflow occurs. The operand lengths alone are not an indication of overflow; nonzero digits must have been lost during the operation.

The operands of decimal-arithmetic instructions should not overlap at all or should have coincident rightmost bytes. In ZERO AND ADD, the operands may also overlap in such a manner that the rightmost byte of the first operand (which becomes the result) is to the right of the rightmost byte of the second operand. For these cases of proper overlap, the result is obtained as if operands were processed right to left. Because the codes for digits and signs are verified during the performance of the arithmetic, improperly overlapping operands are recognized as data exceptions. However, in ZERO AND ADD when the rightmost byte of the first operand is to the left of the rightmost byte of the second operand, the entire second operand may be fetched, depending on the model, before any storing occurs, which will cause a data exception not to be recognized. See “Interlocks within a Single Instruction” on page 5-87 for how overlap is detected in the access-register mode.

Programming Note: A signed-packed-decimal number in storage may be designated as both the first and second operand of ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, SUBTRACT DECIMAL, or ZERO AND ADD. Thus, a decimal number may be added to itself, compared with itself, and so forth; SUBTRACT DECIMAL may be used to set a decimal field in storage to zero; and, for MULTIPLY DECIMAL, a decimal number may be

squared in place. In these cases, the lengths of the two operands are not necessarily equal and may, depending on the instruction, be prohibited from being equal.

Editing Instructions

The editing instructions are EDIT and EDIT AND MARK. For these instructions, only the first operand (the pattern) has an explicitly specified length. The second operand (the source) is considered to have as many digits as necessary for the completion of the operation.

Overlapping operands for the editing instructions yield unpredictable results.

Execution of Decimal Instructions

During the execution of a decimal instruction, all bytes of the operands are not necessarily accessed concurrently, and the fetch and store accesses to a single location do not necessarily occur one immediately after the other. Furthermore, for decimal instructions, data in source fields may be accessed more than once, and intermediate values may be placed in the result field that may differ from the original operand and final result values. (See “Storage-Operand Consistency” on page 5-95.) Thus, in a multiprocessing configuration, an instruction such as ADD DECIMAL cannot be safely used to update a shared storage location when the possibility exists that another CPU may also be updating that location.

Other Instructions for Decimal Operands

In addition to the decimal-arithmetic instructions in this chapter, (which operate on decimal numbers in the signed-packed-decimal format) several other instructions are provided for operating on decimal data in the zoned, signed-packed-decimal, or unsigned-packed-decimal format.

The editing instructions (described in this chapter) convert decimal numbers in either the signed-packed-decimal or unsigned-packed-decimal format to the zoned format. Zoned results of up to 256 bytes in length can be produced.

CONVERT TO DECIMAL and CONVERT TO BINARY (described in Chapter 7) provide conversion between signed-binary-integer formats and decimal numbers in the signed-packed-decimal format. Signed-packed-decimal operands of up to 16 bytes in length (31 digits and a sign) can be processed.

CONVERT TO SIGNED PACKED and CONVERT FROM SIGNED PACKED (described in Chapter 20) provide conversion between decimal floating point (DFP) formats and the signed-packed-decimal format. Signed-packed-decimal operands of up to 16 bytes in length (31 digits and a sign) can be processed.

CONVERT TO UNSIGNED PACKED and CONVERT FROM UNSIGNED PACKED (described in Chapter 20) provide conversion between decimal floating point (DFP) formats and the unsigned-packed-decimal format. Unsigned-packed-decimal operands of up to 16 bytes in length (32 digits) can be processed.

MOVE WITH OFFSET (described in Chapter 7) operates on data in the packed format. (The first operand is treated as signed-packed decimal and the second operand as unsigned-packed decimal.) Operands are not checked for valid codes. Signed-packed-decimal operands of up to 16 bytes in length (31 digits and a sign) can be processed.

PACK, UNPACK, PACK ASCII, UNPACK ASCII, PACK UNICODE, and UNPACK UNICODE provide conversion between zoned or character data and signed-packed-decimal data. Operands are not checked for valid codes. Signed-packed-decimal operands of up to 16 bytes in length (31 digits and a sign) can be processed.

MOVE NUMERICS and MOVE ZONES (described in Chapter 7) operate on data in the zoned format. Operands are not checked for valid codes. Zoned operands of up to 256 bytes in length can be processed.

Decimal-Operand Data Exception

A decimal-operand data exception is recognized when any of the following is true:

1. The sign or digit codes are invalid in the operands of the decimal instructions or in CONVERT TO BINARY (described in Chapter 7, “General Instructions”); or in CONVERT FROM SIGNED

PACKED and CONVERT FROM UNSIGNED PACKED (described in Chapter 20, “Decimal-Floating-Point Instructions”).

2. The operand fields in ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, and SUBTRACT DECIMAL overlap in a way other than with coincident rightmost bytes; or operand fields in ZERO AND ADD overlap, and the rightmost byte of the second operand is to the right of the rightmost byte of the first operand. On some models, the improper overlap of operands for ZERO AND ADD is not recognized as a decimal-operand data exception; instead, the operation is performed as if the entire second operand were fetched before any byte of the result is stored.
3. The multiplicand in MULTIPLY DECIMAL has an insufficient number of leftmost zeros.

A decimal-operand data exception causes the operation to be suppressed, except that, for EDIT and EDIT AND MARK, and COMPRESSION CALL, the operation may be suppressed or terminated. In the case of EDIT and EDIT AND MARK, and COMPRESSION CALL, an invalid sign code cannot occur.

Instructions

The decimal instructions and their mnemonics, formats, and operation codes are listed in Figure 8-2 on page 8-5. The figure also indicates when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

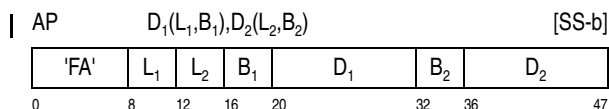
Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For ADD DECIMAL, for example, AP is the mnemonic and $D_1(L_1, B_1), D_2(L_2, B_2)$ the operand designation.

Programming Note: The decimal instruction TEST DECIMAL is available when the extended-translation facility 2 is installed.

Name	Mne- monic	Characteristics						Op Code	Page		
ADD DECIMAL	AP	SS-b	C	A	Dd	DF	ST	B ₁ B ₂	FA	8-6	
COMPARE DECIMAL	CP	SS-b	C	A	Dd			B ₁ B ₂	F9	8-6	
DIVIDE DECIMAL	DP	SS-b		A	SP	Dd	DK	ST	B ₁ B ₂	FD	8-7
EDIT	ED	SS-a	C	A	Dd			ST	B ₁ B ₂	DE	8-7
EDIT AND MARK	EDMK	SS-a	C	A	Dd	G1		ST	B ₁ B ₂	DF	8-10
MULTIPLY DECIMAL	MP	SS-b		A	SP	Dd		ST	B ₁ B ₂	FC	8-11
SHIFT AND ROUND DECIMAL	SRP	SS-c	C	A	Dd	DF		ST	B ₁ B ₂	F0	8-12
SUBTRACT DECIMAL	SP	SS-b	C	A	Dd	DF		ST	B ₁ B ₂	FB	8-13
TEST DECIMAL	TP	RSL	C	E2	A				B ₁ B ₂	EBC0	8-13
ZERO AND ADD	ZAP	SS-b	C	A	Dd	DF		ST	B ₁ B ₂	F8	8-13
Explanation:											
A Access exceptions for logical addresses.											
B ₁ B ₁ field designates an access register in the access-register mode.											
B ₂ B ₂ field designates an access register in the access-register mode.											
C Condition code is set.											
Dd Decimal-operand data exception.											
DF Decimal-overflow exception.											
DK Decimal-divide exception.											
E2 Extended-translation facility 2.											
G1 Instruction execution includes the implied use of general register 1.											
RSL RSL instruction format.											
SP Specification exception.											
SS SS instruction format.											
ST PER storage-alteration event.											

Figure 8-2. Summary of Decimal Instructions

ADD DECIMAL



The second operand is added to the first operand, and the resulting sum is placed at the first-operand location. The operands and result are in the signed-packed-decimal format.

Addition is algebraic, taking into account the signs and all digits of both operands. All sign and digit codes are checked for validity.

If the first operand is too short to contain all leftmost nonzero digits of the sum, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs.

The sign of the sum is determined by the rules of algebra. In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, a zero result is given either a positive or negative sign, as determined by what the sign of the correct sum would have been.

Resulting Condition Code:

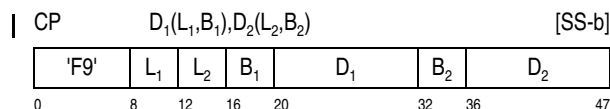
- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Decimal overflow

Programming Note: An example of the use of the ADD DECIMAL instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

COMPARE DECIMAL



The first operand is compared with the second operand, and the result is indicated in the condition code. The operands are in the signed-packed-decimal format.

Comparison is algebraic and follows the procedure for decimal subtraction, except that both operands remain unchanged. When the difference is zero, the operands are equal. When a nonzero difference is positive or negative, the first operand is high or low, respectively.

Overflow cannot occur because the difference is discarded.

All sign and digit codes are checked for validity.

Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 --

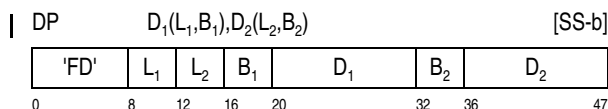
Program Exceptions:

- Access (fetch, operands 1 and 2)
- Data

Programming Notes:

1. An example of the use of the COMPARE DECIMAL instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The preferred and alternate sign codes for a particular sign are treated as equivalent for comparison purposes.
3. A negative zero and a positive zero compare equal.

DIVIDE DECIMAL



The first operand (the dividend) is divided by the second operand (the divisor). The resulting quotient and remainder are placed at the first-operand location. The operands and results are in the signed-packed-decimal format.

The quotient is placed leftmost in the first-operand location. The number of bytes in the quotient field is equal to the difference between the dividend and divisor lengths ($L_1 - L_2$). The remainder is placed rightmost in the first-operand location and has a length equal to the divisor length. Together, the quotient and remainder fields occupy the entire first operand; therefore, the address of the quotient is the address of the first operand.

The divisor length cannot exceed 15 digits and sign (L_2 not greater than seven) and must be less than the dividend length (L_2 less than L_1); otherwise, a specification exception is recognized.

The dividend, divisor, quotient, and remainder are each signed decimal integers in the signed-packed-decimal format and are right-aligned in their fields. All sign and digit codes of the dividend and divisor are checked for validity.

The sign of the quotient is determined by the rules of algebra from the dividend and divisor signs. The sign of the remainder has the same value as the dividend sign. These rules hold even when the quotient or remainder is zero.

Overflow cannot occur. If the divisor is zero or the quotient is too large to be represented by the number of digits specified, a decimal-divide exception is recognized. This includes the case of division of zero by zero. The decimal-divide exception is indicated only if the sign codes of both the dividend and divisor are valid, and only if the digit or digits used in establishing the exception are valid.

Condition Code: The code remains unchanged.

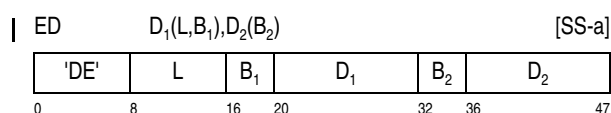
Program Exceptions:

- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Decimal divide
- Specification

Programming Notes:

1. An example of the use of the DIVIDE DECIMAL instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."
2. The dividend cannot exceed 31 digits and sign. Since the remainder cannot be shorter than one digit and sign, the quotient cannot exceed 29 digits and sign.
3. The condition for a decimal-divide exception can be determined by a trial comparison. The leftmost digit of the divisor is aligned one digit to the right of the leftmost dividend digit, with rightmost zeros appended up to the length of the dividend. When the divisor, so aligned, is less than or equal to the dividend, ignoring signs, a divide exception is indicated.
4. If a data exception does not exist, a decimal-divide exception occurs when the leftmost dividend digit is not zero.

EDIT



The second operand (the source), which normally contains one or more decimal numbers in the signed-packed-decimal or unsigned-packed-decimal format, is changed to the zoned format and modified under the control of the first operand (the pattern). The edited result replaces the first operand.

The length field specifies the length of the first operand, which may contain bytes of any value.

The length of the source is determined by the operation according to the contents of the pattern. The source normally consists of one or more decimal numbers, each in the signed-packed-decimal or unsigned-packed-decimal format. The leftmost four bits of each source byte must specify a decimal-digit code (0000-1001); a sign code (1010-1111) is recog-

nized as a data exception. The rightmost four bits may specify either a sign code or a decimal-digit code. Access and data exceptions are recognized only for those bytes in the second operand which are actually required.

The result is obtained as if both operands were processed left to right one byte at a time. Overlapping pattern and source fields give unpredictable results.

During the editing process, each byte of the pattern is affected in one of three ways:

- 1. It is left unchanged.
- 2. It is replaced by a source digit expanded to the zoned format.
- 3. It is replaced by the first byte in the pattern, called the fill byte.

Which of the three actions takes place is determined by one or more of the following: the type of the pattern byte, the state of the significance indicator, and whether the source digit examined is zero.

Pattern Bytes: There are four types of pattern bytes: digit selector, significance starter, field separator, and message byte. Their coding is as follows:

Name	Code	
	(Binary)	(Hex)
Digit Selector	0010 0000	20
Significance starter	0010 0001	21
Field separator	0010 0010	22
Message byte	Any other	Any other

The detection of either a digit selector or a significance starter in the pattern causes an examination to be made of the significance indicator and of a source digit. As a result, either the expanded source digit or the fill byte, as appropriate, is selected to replace the pattern byte. Additionally, encountering a digit selector or a significance starter may cause the significance indicator to be changed.

The field separator identifies individual fields in a multiple-field editing operation. It is always replaced in the result by the fill byte, and the significance indicator is always off after the field separator is encountered.

Message bytes in the pattern are either replaced by the fill byte or remain unchanged in the result, depending on the state of the significance indicator. They may thus be used for padding, punctuation, or text in the significant portion of a field or for the insertion of sign-dependent symbols.

Fill Byte: The first byte of the pattern is used as the fill byte. The fill byte can have any code and may concurrently specify a control function. If this byte is a digit selector or significance starter, the indicated editing action is taken after the code has been assigned to the fill byte.

Source Digits: Each time a digit selector or significance starter is encountered in the pattern, a new source digit is examined for placement in the pattern field. Either the source digit is disregarded, or it is expanded to the zoned format, by appending the zone code 1111 on the left, and stored in place of the pattern byte.

Execution is as if the source digits were selected one byte at a time. The leftmost four bits of each byte are examined first, and the rightmost four bits, when they represent a decimal-digit code, remain available for the next pattern byte that calls for a digit examination. When the leftmost four bits contain an invalid digit code, a data exception is recognized, and the operation is either suppressed or terminated.

At the time the left digit of a source byte is examined, the rightmost four bits are checked for the existence of a sign code. When a sign code is encountered in the rightmost four bit positions, these bits are not treated as a decimal-digit code, and a new source byte is fetched from storage when the next pattern byte calls for a source-digit examination.

When the pattern contains no digit selector or significance starter, no source bytes are fetched and examined.

Significance Indicator: The significance indicator is turned on or off to indicate the significance or non-significance, respectively, of subsequent source digits or message bytes. Significant source digits replace their corresponding digit selectors or significance starters in the result. Significant message bytes remain unchanged in the result.

The significance indicator, by its on or off state, indicates also the negative or positive value, respec-

tively, of a completed source field and is used as one factor in the setting of the condition code.

The significance indicator is set to off at the start of the editing operation, after a field separator is encountered, or after a source byte is examined that has a plus code in the rightmost four bit positions.

The significance indicator is set to on when a significance starter is encountered whose source digit is a valid decimal digit, or when a digit selector is encountered whose source digit is a nonzero decimal digit, provided that in both instances the source byte does not have a plus code in the rightmost four bit positions.

In all other situations, the significance indicator is not changed. A minus sign code has no effect on the significance indicator.

Result Bytes: The result of an editing operation replaces and is equal in length to the pattern. It is composed of pattern bytes, fill bytes, and zoned source digits.

If the pattern byte is a message byte and the significance indicator is on, the message byte remains unchanged in the result. If the pattern byte is a field separator or if the significance indicator is off when a message byte is encountered in the pattern, the fill byte replaces the pattern byte in the result.

If the digit selector or significance starter is encountered in the pattern with the significance indicator off and the source digit zero, the source digit is considered nonsignificant, and the fill byte replaces the pattern byte. If the digit selector or significance starter is encountered either with the significance indicator on or with a nonzero decimal source digit, then the source digit (a) is considered significant, (b) is changed to the zoned format, and (c) replaces the pattern byte in the result. Examination of the significance indicator occurs prior to the processing of the source digit (which might change the significance indicator).

Condition Code: The sign and magnitude of the last field edited are used to set the condition code. The term “last field” refers to those source digits, if any, in the second operand selected by digit selectors or significance starters after the last field separator; if the pattern contains no field separator, there is only one field, which is considered to be the last field.

If no such source digits are selected, the last field is considered to be of zero length.

Condition code 0 is set when the last field edited is zero or of zero length.

Condition code 1 is set when the last field edited is nonzero and the significance indicator is on. (This indicates a result less than zero if the last source byte examined contained a sign code in the rightmost four bits.)

Condition code 2 is set when the last field edited is nonzero and the significance indicator is off. (This indicates a result greater than zero if the last source byte examined contained a sign code in the rightmost four bits.)

For the purposes of setting condition codes 2 or 3, the significance indicator is examined after the processing of the last source digit.

Figure 8-3 on page 8-10 summarizes the functions of the EDIT and EDIT AND MARK operations. The leftmost four columns list all the significant combinations of the four conditions that can be encountered in the execution of an editing operation. The rightmost two columns list the action taken for each case — the type of byte placed in the result field and the new setting of the significance indicator.

Resulting Condition Code:

- 0 Last field zero or zero length
- 1 Last field less than zero
- 2 Last field greater than zero
- 3 --

Program Exceptions:

- Access (fetch, operand 2; fetch and store, operand 1)
- Data

Programming Notes:

1. Examples of the use of the EDIT instruction are given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. Editing includes sign and punctuation control, and the suppression of leading zeros by replacing them with blanks, or the protection of leading zeros from malicious alteration by replacing them

Conditions				Results	
Pattern Byte	Previous State of Significance Indicator	Source Digit	Right Four Source Bits Are Plus Code	Result Byte	State of Significance Indicator at End of Digit Examination
Digit selector	Off	0	*	Fill byte	Off
		1-9	No	Source digit#	On
			Yes	Source digit#	Off
	On	0-9	No	Source digit	On
			Yes	Source digit	Off
Significance starter	Off	0	No	Fill byte	On
			Yes	Fill byte	Off
		1-9	No	Source digit#	On
			Yes	Source digit#	Off
	On	0-9	No	Source digit	On
			Yes	Source digit	Off
Field separator	*	**	**	Fill byte	Off
Message byte	Off	**	**	Fill byte	Off
	On	**	**	Message byte	On
Explanation: * No effect on result byte or on new state of significance indicator. ** Not applicable because source is not examined. # For EDIT AND MARK only, the address of the rightmost such result byte is placed in general register 1.					

Figure 8-3. Summary of Editing Functions

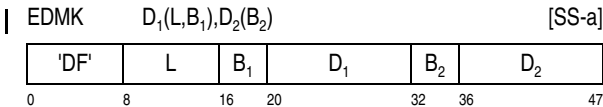
with asterisks or other characters. It also facilitates programmed blanking of all-zero fields. Several fields may be edited in one operation, and numeric information may be combined with text.

3. In most cases, the source is shorter than the pattern because each four-bit source digit produces an eight-bit byte in the result.
4. The total number of digit selectors and significance starters in the pattern always equals the number of source digits edited.
5. If the fill byte is a blank, if no significance starter exists in the pattern, and if the source digit examined for each digit selector is zero, the editing operation blanks the result field.
6. The resulting condition code indicates whether or not the last field is all zeros and, if nonzero, reflects the state of the significance indicator. The significance indicator reflects the sign of the source field only if the last source byte examined contains a sign code in the rightmost four bits. For multiple-field editing operations, the condition

code reflects the sign and value only of the field following the last field separator.

7. Significant performance degradation is possible when the second-operand address of an EDIT instruction designates a location that is closer to the left of an access boundary than the length of the first operand of that instruction (for the purposes of this discussion, an access boundary is 4 K-bytes, except when fetch-protection override applies in which case it is 2 K-bytes). This is because the machine may perform a trial execution of the instruction to determine if the second operand actually crosses the boundary. The second operand of EDIT, while normally shorter than the first operand, can in the extreme case have the same length as the first.

EDIT AND MARK



The second operand (the source), which normally contains one or more decimal numbers in the signed-packed-decimal or unsigned-packed-decimal format, is changed to the zoned format and modified under the control of the first operand (the pattern). The address of the first significant result byte of the rightmost (or only) field is inserted in general register 1. The edited result replaces the pattern.

EDIT AND MARK is identical to EDIT, except for the additional function of inserting the address of the result byte in general register 1 if the result byte is a zoned source digit and the significance indicator was off before the examination of the source bytes. If no result byte meets the criteria, general register 1 remains unchanged; if more than one result byte meets the criteria, the address of the rightmost such result byte is inserted.

In the 24-bit addressing mode, the address replaces bits 40-63 of general register 1, and bits 0-39 of the register are not changed. In the 31-bit addressing mode, the address replaces bits 33-63 of general register 1, bit 32 of the register is set to zero, and bits 0-31 of the register remain unchanged. In the 64-bit addressing mode, the address replaces bits 0-63 of general register 1.

The contents of access register 1 remain unchanged.

See Figure 8-3 on page 8-10 for a summary of the EDIT and EDIT AND MARK operations.

Resulting Condition Code:

- 0 Last field zero or zero length
- 1 Last field less than zero
- 2 Last field greater than zero
- 3 --

Program Exceptions:

- Access (fetch, operand 2; fetch and store, operand 1)
- Data

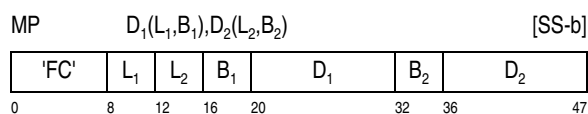
Programming Notes:

1. Examples of the use of the EDIT AND MARK instruction are given Appendix A, "Number Representation and Instruction-Use Examples."
2. EDIT AND MARK facilitates the programming of floating currency-symbol insertion. Using appropriate

source and pattern data, the address inserted in general register 1 is one greater than the address where a floating currency-sign would be inserted.

3. No address is inserted in general register 1 when the significance indicator is turned on as a result of encountering a significance starter with the corresponding source digit zero. To ensure that general register 1 contains a proper address when this occurs, the address of the pattern byte that immediately follows the appropriate significance starter could be placed in the register beforehand.
4. When multiple fields are edited with one execution of the EDIT AND MARK instruction, the address, if any, inserted in general register 1 applies to the rightmost field edited for which the criteria were met.
5. See also the programming note under EDIT regarding performance degradation due to a possible trial execution.

MULTIPLY DECIMAL



The product of the first operand (the multiplicand) and the second operand (the multiplier) is placed at the first-operand location. The operands and result are in the signed-packed-decimal format.

The multiplier length cannot exceed 15 digits and sign (L_2 not greater than seven) and must be less than the multiplicand length (L_2 less than L_1); otherwise, a specification exception is recognized.

The multiplicand must have at least as many bytes of leftmost zeros as the number of bytes in the multiplier; otherwise, a data exception is recognized. This restriction ensures that no product overflow occurs.

The multiplicand, multiplier, and product are each signed decimal integers in the signed-packed-decimal format and are right-aligned in their fields. All sign and digit codes of the multiplicand and multiplier are checked for validity. The sign of the product is determined by the rules of algebra from the multiplier

and multiplicand signs, even if one or both operands are zeros.

Condition Code: The code remains unchanged.

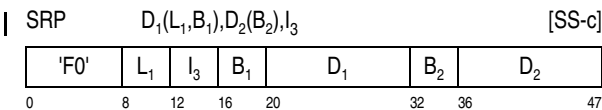
Program Exceptions:

- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Specification

Programming Notes:

1. An example of the use of the MULTIPLY DECIMAL instruction is given Appendix A, “Number Representation and Instruction-Use Examples.”
2. The product cannot exceed 31 digits and sign. The leftmost digit of the product is always zero.

SHIFT AND ROUND DECIMAL



The first operand is shifted in the direction and for the number of decimal-digit positions specified by the second-operand address, and, when shifting to the right is specified, the absolute value of the first operand is rounded by the rounding digit, I_3 . The first operand and the result are in the signed-packed-decimal format.

The first operand is considered to be in the signed-packed-decimal format. Only its digit portion is shifted; the sign position does not participate in the shifting. Zeros are supplied for the vacated digit positions. The result replaces the first operand. Nothing is stored outside of the specified first-operand location.

The second-operand address, specified by the B_2 and D_2 fields, is not used to address data; bits 58-63 of that address are the shift value, and the leftmost bits of the address are ignored.

The shift value is a six-bit signed binary integer, indicating the direction and the number of decimal-digit positions to be shifted. Positive shift values specify shifting to the left. Negative shift values, which are

represented in two's complement notation, specify shifting to the right. The following are examples of the interpretation of shift values:

Shift Value (Binary)	Amount and Direction
011111	31 digits to the left
000001	One digit to the left
000000	No shift
111111	One digit to the right
100000	32 digits to the right

For a right shift, the I_3 field, bits 12-15 of the instruction, is used as a decimal rounding digit. The first operand, which is treated as positive by ignoring the sign, is rounded by decimally adding the rounding digit to the leftmost of the digits to be shifted out and by propagating the carry, if any, to the left. The result of this addition is then shifted right. Except for validity checking and the participation in rounding, the digits shifted out of the rightmost decimal-digit position are ignored and are lost.

If one or more nonzero digits are shifted out during a left shift, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs. Overflow cannot occur for a right shift, with or without rounding, or when no shifting is specified.

In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, the sign of the result is the same as the original sign but with the preferred sign code.

A data exception is recognized when the first operand does not have valid sign and digit codes or when the rounding digit is not a valid digit code. The validity of the first-operand codes is checked even when no shift is specified, and the validity of the rounding digit is checked even when no addition for rounding takes place.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

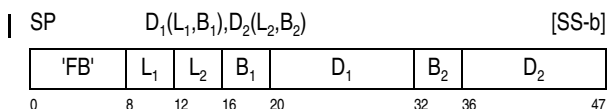
Program Exceptions:

- Access (fetch and store, operand 1)
- Data
- Decimal overflow

Programming Notes:

1. Examples of the use of the SHIFT AND ROUND DECIMAL instruction are given in Appendix A, “Number Representation and Instruction-Use Examples.”
2. SHIFT AND ROUND DECIMAL can be used for shifting up to 31 digit positions left and up to 32 digit positions right. This is sufficient to clear all digits of any decimal number even with rounding.
3. For right shifts, the rounding digit 5 provides conventional rounding of the result. The rounding digit 0 specifies truncation without rounding.
4. When the B_2 field is zero, the six-bit shift value is obtained directly from bits 42-47 of the instruction.

SUBTRACT DECIMAL



The second operand is subtracted from the first operand, and the resulting difference is placed at the first-operand location. The operands and result are in the signed-packed-decimal format.

SUBTRACT DECIMAL is executed the same as ADD DECIMAL, except that the second operand is considered to have a sign opposite to the sign in storage. The second operand in storage remains unchanged.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow
- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Decimal overflow

TEST DECIMAL



The first operand is tested for valid decimal digits and a valid sign code, and the result is indicated in the condition code. The operand is in the signed-packed-decimal format.

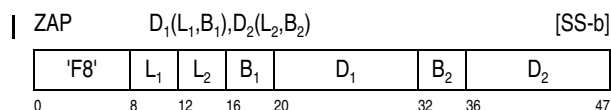
Resulting Condition Code:

- 0 All digit codes and the sign valid
- 1 Sign invalid
- 2 At least one digit code invalid
- 3 Sign invalid and at least one digit code invalid

Program Exceptions:

- Access (fetch, operand 1)
- Operation (if the extended-translation facility 2 is not installed)

ZERO AND ADD



The second operand is placed at the first-operand location. The operation is equivalent to an addition to zero. The operand and result are in the signed-packed-decimal format.

Only the second operand is checked for valid sign and digit codes. Extra zeros are supplied on the left for the shorter operand if needed.

If the first operand is too short to contain all leftmost nonzero digits of the second operand, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs.

In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, a zero result is given the sign of the second operand but with the preferred sign code.

The two operands may overlap, provided the rightmost byte of the first operand is coincident with or to the right of the rightmost byte of the second operand. In this case, the result is obtained as if the operands were processed right to left. When the operands overlap and the rightmost byte of the first operand is to the left of the rightmost byte of the second operand, then, depending on the model, either a data exception is recognized or the result is obtained as if the entire second operand were fetched before any byte of the result is stored.

Resulting Condition Code:

- 0 Result zero; no overflow
- 1 Result less than zero; no overflow

- 2 Result greater than zero; no overflow
- 3 Overflow

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Data
- Decimal overflow

Programming Note: An example of the use of the ZERO AND ADD instruction is given in Appendix A, "Number Representation and Instruction-Use Examples."

Chapter 9. Floating-Point Overview and Support Instructions

Sign Bit	9-2	IEEE Exceptions	9-17
Finite Floating-Point Numbers	9-2	Concurrent IEEE Exceptions.	9-18
Infinities	9-2	IEEE Invalid Operation	9-18
Not-A-Number (NaN).	9-2	IEEE Division-By-Zero	9-19
Signaling and Quiet NaNs	9-2	IEEE Overflow.	9-19
Payload	9-3	IEEE Underflow.	9-19
Propagation of NaNs	9-3	IEEE Inexact	9-20
Default QNaN	9-3	Quantum Exception.	9-21
Hexadecimal-Floating-Point (HFP)	9-3	Suppression of Certain IEEE Exceptions	9-23
Binary Floating-Point (BFP).	9-4	IEEE Same-Radix Format Conversion	9-23
Decimal Floating-Point (DFP)	9-4	IEEE Comparison	9-24
Canonical DFP Data	9-4	Condition Codes for IEEE Instructions	9-24
Comparison of Floating-Point Number		Instructions	9-24
Representations	9-4	CONVERT BFP TO HFP	9-27
Floating-Point Number Ranges	9-4	CONVERT HFP TO BFP	9-28
Equivalent Floating-Point Number		COPY SIGN	9-29
Representations.	9-5	EXTRACT FPC.	9-30
Effective Width.	9-7	LOAD	9-30
Floating-Point Data in Storage.	9-8	LOAD COMPLEMENT	9-30
Registers And Controls	9-8	LOAD FPC	9-31
Floating-Point Registers	9-8	LOAD FPC AND SIGNAL.	9-31
Additional Floating-Point (AFP) Registers	9-9	LOAD FPR FROM GR	9-33
Valid Floating-Point-Register Designations.	9-9	LOAD GR FROM FPR	9-33
Floating-Point-Control (FPC) Register.	9-9	LOAD NEGATIVE.	9-33
IEEE Masks and Flags	9-10	LOAD POSITIVE.	9-34
FPC DXC Byte.	9-10	LOAD ZERO	9-34
Operations on the FPC Register	9-10	PERFORM FLOATING-POINT OPERATION.	9-34
AFP-Register-Control Bit	9-11	General Register 0 (GR0).	9-35
IEEE Computational Operations	9-11	Return Code	9-37
Intermediate Values.	9-11	Sign Preservation	9-37
Precise Intermediate Value	9-12	Preferred Quantum	9-37
Precision-Rounded Value	9-12	NaN Conversion	9-38
Denormalized Value	9-12	Scaled Value and Signed Scaling Exponent	
Functionally-Rounded Value	9-12	(Ω) for PFPO.	9-38
Rounded Intermediate Value	9-12	HFP Values.	9-38
Scaled Value	9-12	HFP Zero Result	9-38
Scale Factor (Ψ)	9-12	HFP Overflow and Underflow for PFPO	9-38
Unsigned Scaling Exponent (α).	9-12	IEEE Exceptions for PFPO	9-39
Signed Scaling Exponent (Ω)	9-12	PFPO Action Figures	9-44
IEEE Rounding	9-13	SET BFP ROUNDING MODE.	9-45
Permissible Set	9-13	SET DFP ROUNDING MODE	9-45
Selection of Candidates.	9-13	SET FPC.	9-46
Ties	9-13	SET FPC AND SIGNAL	9-46
Voting Digit and Common-Rounding- Point		STORE	9-47
View	9-13	STORE FPC	9-47
Rounding Methods.	9-13	Summary of All Floating-Point Instructions.	9-47
Explicit Rounding Methods	9-16	Impacts on ESA/390	9-51
Summary of Rounding Action	9-17	Impacts of the Decimal-Floating-Point Facility	9-51

Floating-point instructions are used to perform calculations on operands having a wide range of magnitude and to obtain results scaled to preserve precision.

Floating-point operands have formats based on three radices: 2, 10, or 16. These radix values lead to the terminology “binary,” “decimal,” and “hexadecimal” floating point (BFP, DFP, and HFP), respectively. The formats are also based on three operand lengths: short (32 bits), long (64 bits), and extended (128 bits). Short operands require less storage than long or extended operands. On the other hand, long and extended operands permit greater precision in computation.

Sign Bit

All floating-point data have a sign bit. The sign bit is zero for plus and one for minus.

Finite Floating-Point Numbers

A finite floating-point number has three components: a sign bit, an exponent, and a significand. The magnitude (an unsigned value) of the number is the product of the significand and the radix raised to the power of the exponent. The number is positive or negative depending on whether the sign bit is zero or one, respectively.

The significand consists of a string of digits, where each digit is an integral value from zero to one less than the radix (2, 10, or 16). (Thus, a BFP digit is one bit, an HFP digit is four bits, and a DFP digit is a value from zero to nine.) The number of digit positions in the significand is called the precision of the floating-point number. The significand has an implied radix point, which, depending on the view, may be considered to be on the left, to the right of the leftmost digit, on the right, or elsewhere.

The exponent, a signed value, is represented as an unsigned binary value by adding a bias; the result, for BFP and DFP, is called the biased exponent; for HFP, it is called the characteristic. The value of the bias depends on the view. In the *fraction view*, the radix point is considered to be the left of the significand. In the *left-units view*, the radix point is considered to be to the right of the leftmost digit. In the *right-units view*, the radix point is considered to be on the right of the

significand. By choosing the appropriate bias, any finite floating-point number can be considered in any of these views, or even in another view. For the first three of these views, the bias is called the *fraction-view bias*, *left-units-view bias*, and *right-units-view bias*, respectively. Except where otherwise indicated, HFP is defined in terms of the fraction view, BFP terms of the left-units view, and DFP in terms of the right-units view.

For HFP, the significand is considered to be a fraction with the implied radix point on the left. In this view, the significand is referred to as the fraction. For BFP, the significand consists of an implicit unit digit to the left of an implied radix point and an explicit fraction field to the right. For DFP, the significand is considered to be an integer with the implied radix point on the right.

Infinities

BFP and DFP data include an infinite numeric datum, called infinity. Infinities can participate in most arithmetic operations and give a consistent result, usually infinity. An infinity has a sign bit. In comparisons, infinities of the same sign compare equal, $+\infty$ compares greater than any finite number, and $-\infty$ compares less than any finite number.

Not-A-Number (NaN)

BFP and DFP data types include a nonnumeric datum, called not-a-number (or NaN). A NaN is produced in place of a numeric result after an invalid operation when there is no IEEE trap action. NaNs may also be used by the program to flag special operands, such as the contents of an uninitialized storage area. A NaN has a sign bit, a NaN-type bit, and a payload.

Signaling and Quiet NaNs

There are two types of NaNs, signaling and quiet. A signaling NaN (SNaN) is distinguished from the corresponding quiet NaN (QNaN) by the NaN-type bit.

For BFP, the NaN-type bit is the leftmost bit of the fraction field, and is called the QNaN bit: a BFP NaN is an SNaN, or a QNaN, depending on whether the QNaN bit is zero, or one, respectively. For DFP, the

NaN-type bit is in bit position 6 in all three formats, and is called the SNaN bit: a DFP NaN is a QNaN, or an SNaN, depending on whether the SNaN bit is zero, or one, respectively.

Payload

NaNs include diagnostic information called the payload. For BFP, the payload has two fewer bits than the precision and is considered to be a left-aligned bit-reversed binary integer. For DFP, the payload has one fewer digit than the precision and is considered to be a right-aligned decimal integer.

For both BFP and DFP the payload of a NaN is considered to be an unsigned integer. Let p represent the precision, in digits, of a particular format. For DFP, bit 6 of the format is called the SNaN bit and the $p-1$ digits in the trailing significand field are the numeric value of the payload. For BFP, the leftmost bit in the fraction field is called the QNaN bit and the remaining $p-2$ bits of the fraction field are the numeric value of the payload.

The numeric value of the BFP NaN payload is bit-reversed. That is, the first bit to the right of the QNaN bit is considered to have a value of one, the next bit a value of two, and so on, with each bit having a value of twice the value of the bit to its left.

Propagation of NaNs

Normally, QNaNs are just propagated during computations so that they will remain visible at the end; while an SNaN operand causes an IEEE-invalid-operation exception. If the IEEE-invalid-operation mask (FPC 0.0) is zero, the result is the corresponding QNaN, which is produced by inverting the NaN-type bit, and setting the IEEE-invalid-operation flag (FPC 1.0) is set to one. If the IEEE-invalid-operation mask (FPC 0.0) is one, the operation is suppressed, and a data exception for IEEE-invalid operation occurs.

Default QNaN

A special QNaN is supplied as the default result for an IEEE-invalid-operation exception; it has a plus sign and a payload of zeros.

Programming Notes:

1. The program can generate and assign values to the payload of a NaN. The CPU propagates those values unchanged, except that an SNaN is changed to the corresponding QNaN if the IEEE-

invalid-operation mask bit is zero, and conversion to a narrower format truncates digits from the payload. For BFP, bits are truncated on the right; for DFP, digits are truncated on the left. For the PFPO-convert-floating-point-radix operation, payloads are preserved, except that payloads larger than the capacity of the target format are replaced by the default QNaN (payload zero).

2. Depending on the application, the program may or may not desire that an SNaN signal the IEEE invalid-operation exception. The sign-handling instructions LOAD COMPLEMENT, LOAD NEGATIVE, and LOAD POSITIVE do not signal the invalid-operation exception but, instead, treat SNaNs like any other data. LOAD AND TEST signals the invalid-operation exception when the operand is an SNaN. This instruction, in conjunction with the above instructions, gives the program the choice of either option.
3. The instructions LOAD LENGTHENED and LOAD ROUNDED change the precision of a floating-point datum. The BFP versions of these instructions signal the invalid-operation exception when the operand is an SNaN. The DFP instructions have a modifier bit to control whether the exception is signaled.

Hexadecimal-Floating-Point (HFP)

Hexadecimal-floating-point (HFP) operands have formats which provide for exponents that specify powers of the radix 16 and significands that are hexadecimal numbers. The exponent range is the same for the short, long, and extended formats. The results of most operations on HFP data are truncated to fit into the target format, but there are instructions available to round the result when converting to a narrower format. For HFP operands, the implicit unit digit of the significand is always zero. Since the value of the significand and fraction are the same, HFP operations are described in terms of the fraction, and the term significand is not used.

Either normalized or unnormalized numbers may be used as operands for any HFP or DFP operation, where, for HFP, a normalized number is one having a nonzero leftmost fraction digit, or, for DFP, a normalized number is one having a nonzero leftmost significand digit. Most HFP instructions generate normalized results for greatest precision. HFP add and subtract instructions that generate unnormalized results are also available.

Binary Floating-Point (BFP)

Binary-floating-point (BFP) operands have formats which provide for exponents that specify powers of the radix 2 and significands that are binary numbers. The exponent range differs for different formats, the range being greater for the longer formats. In the long and extended formats, the exponent range is significantly greater for BFP data than for HFP data. The results of operations performed on BFP data are rounded automatically to fit into the target format; the manner of rounding is determined by a program-settable BFP rounding mode.

There are no unnormalized operands for BFP operations. For normal BFP numbers, the implicit unit digit of the significand is one. For values too small in magnitude to be represented in normalized form, the implicit unit digit is zero. These numbers are called “subnormal” numbers.¹ Unlike the HFP and DFP formats, where the same value can have multiple representations in a given format because of the possibility of unnormalized numbers, the BFP format does not allow such redundancy.

Decimal Floating-Point (DFP)

Decimal-floating-point (DFP) operands have formats which provide for exponents that specify powers of the radix 10 and significands that are decimal numbers. The exponent range differs for different formats, the range being greater for the longer formats. The exponent range is greater for DFP data than for BFP data. The results of operations performed on DFP data are rounded automatically to fit into the target format; the manner of rounding is determined by a program-settable DFP rounding mode.

Like HFP, DFP numbers can be normalized or unnormalized. Either normalized or unnormalized numbers may be used as operands for any DFP operation. For DFP, a normalized number is one having a nonzero leftmost significand digit. Because of the possibility of unnormalized numbers, the same value can have multiple representations in a given DFP format. The representations having the same value are called members of a cohort. Unlike HFP, DFP instructions generate normalized results for greater precision only when the result is inexact. When the result is

exact, most DFP instructions produce a value in the form that preserves information called the quantum.

Canonical DFP Data

A *canonical DFP number* has only canonical declets, that is, the digits in the trailing-significand field are encoded using only the preferred DPD codes.

A *canonical DFP infinity* has bit 6 and all bits to the right of this in the format set to zeros.

A *canonical DFP NaN* has zeros in all bits to the right of the NaN-type bit in the combination field and only canonical declets in the trailing significand field.

Noncanonical DFP numbers, infinities, and NaNs are accepted as source operands, but all DFP results are canonical.

Comparison of Floating-Point Number Representations

Floating-Point Number Ranges

Figure 9-1 shows the range of numbers, in decimal form, that can be represented in different floating-point formats.

	Type	Short	Long	Extended
Nmax	BFP	$\pm 3.4 \times 10^{+38}$	$\pm 1.8 \times 10^{+308}$	$\pm 1.2 \times 10^{+4932}$
	HFP	$\pm 7.2 \times 10^{+75}$	$\pm 7.2 \times 10^{+75}$	$\pm 7.2 \times 10^{+75}$
	DFP	$\pm 1.0 \times 10^{+97}$	$\pm 1.0 \times 10^{+385}$	$\pm 1.0 \times 10^{+6145}$
Nmin	BFP	$\pm 1.2 \times 10^{-38}$	$\pm 2.2 \times 10^{-308}$	$\pm 3.4 \times 10^{-4932}$
	HFP	$\pm 5.4 \times 10^{-79}$	$\pm 5.4 \times 10^{-79}$	$\pm 5.4 \times 10^{-79}$
	DFP	$\pm 1.0 \times 10^{-95}$	$\pm 1.0 \times 10^{-383}$	$\pm 1.0 \times 10^{-6143}$
Dmin	BFP	$\pm 1.4 \times 10^{-45}$	$\pm 4.9 \times 10^{-324}$	$\pm 6.5 \times 10^{-4966}$
	HFP	$\pm 5.1 \times 10^{-85}$	$\pm 1.2 \times 10^{-94}$	$\pm 1.7 \times 10^{-111}$
	DFP	$\pm 1.0 \times 10^{-101}$	$\pm 1.0 \times 10^{-398}$	$\pm 1.0 \times 10^{-6176}$
Explanation:				
Dmin	Smallest (in magnitude) representable subnormal (BFP or DFP) or nonzero unnormalized (HFP) number.			
Nmax	Largest (in magnitude) representable number.			
Nmin	Smallest (in magnitude) representable normal (BFP or DFP) or normalized (HFP) number.			
Values are decimal approximations.				

Figure 9-1. Number Ranges for BFP and HFP Formats

1. Subnormal numbers were previously called “denormalized” numbers.

Equivalent Floating-Point Number Representations

The exponent of an HFP number is represented in the format as an unsigned seven-bit binary integer called the characteristic. The characteristic is obtained by adding 64 to the exponent value (excess-64 notation). The range of the characteristic is 0 to 127, which corresponds to an exponent range of -64 to +63.

The exponent of a BFP or DFP datum is represented in the format as an unsigned binary integer called the biased exponent. The biased exponent is obtained by adding a bias to the exponent value. The number of bit positions containing the biased exponent, the value of the bias, and the exponent range depend on the data format (short, long, or extended) and are shown for the three formats in Figure 19-4 on page 19-3 and the three DFP formats in Figure 20-3 on page 20-6. Biased exponents are similar to the characteristics of the HFP format, except that, for BFP, special meanings are attached to biased expo-

nents of all zeros and all ones. This is discussed in the section “Classes of BFP Data” on page 19-4.

In each of the three BFP or HFP formats, the binary or hexadecimal point of a number, respectively, is considered to be to the left of the leftmost fraction digit. To the left of the point there is an implied unit digit, which is considered to be zero for HFP numbers or, for BFP numbers, one for normal numbers and zero for zeros and subnormal numbers.

Figure 9-2 on page 9-6 and Figure 9-3 on page 9-7 give examples of the closest representation of the same numbers in the BFP, DFP, and HFP formats, with BFP and DFP values being rounded to nearest and HFP values being truncated.

The figures do not necessarily show the results of conversions exactly. Rounding errors may make a small difference. Also, Figure 9-2 on page 9-6 shows corresponding rounded short-format numbers, not the long HFP results of conversion from short BFP operands.

Value		S	BE or C	Trailing Significand
1.0	B	0	011 1111 1	000 0000 0000 0000 0000 0000
	H	0	100 0001	0001 0000 0000 0000 0000 0000
	D	0	010 0010 0100	0000 0000 0000 0001 0000 [10×10 ⁻¹]
0.5	B	0	011 1111 0	000 0000 0000 0000 0000 0000
	H	0	100 0000	1000 0000 0000 0000 0000 0000
	D	0	010 0010 0100	0000 0000 0000 0000 0000 0101 [5×10 ⁻¹]
1/64	B	0	011 1100 1	000 0000 0000 0000 0000 0000
	H	0	011 1111	0100 0000 0000 0000 0000 0000
	D	0	010 0001 1111	0000 0101 0111 0010 0101 [15625×10 ⁻⁶]
+0	B	0	000 0000 0	000 0000 0000 0000 0000 0000
	H	0	000 0000	0000 0000 0000 0000 0000 0000
	D	0	010 0010 0101	0000 0000 0000 0000 0000 0000 [+0×10 ⁰]
-0	B	1	000 0000 0	000 0000 0000 0000 0000 0000
	H	1	000 0000	0000 0000 0000 0000 0000 0000
	D	1	010 0010 0101	0000 0000 0000 0000 0000 0000 [-0×10 ⁰]
-15.0	B	1	100 0001 0	111 0000 0000 0000 0000 0000
	H	1	100 0001	1111 0000 0000 0000 0000 0000
	D	1	010 0010 0100	0000 0000 0000 1101 0000 [-150×10 ⁻¹]
20/7	B	0	100 0000 0	011 0110 1101 1011 0110 1110
	H	0	100 0001	0010 1101 1011 0110 1101 1011
	D	0	010 1001 1111	1101 0111 0100 1100 0011 [2857143×10 ⁻⁶]
2 ⁻¹²⁶	B	0	000 0000 1	000 0000 0000 0000 0000 0000
	H	0	010 0001	0100 0000 0000 0000 0000 0000
	D	0	000 0111 1001	0011 1101 0110 0101 1010 [1175494×10 ⁻⁴⁴]
2 ⁻¹⁴⁹	B	0	000 0000 0	000 0000 0000 0000 0000 0001
	H	0	001 1011	1000 0000 0000 0000 0000 0000
	D	0	000 0111 0010	1000 0000 0101 0101 1110 [1401298×10 ⁻⁵¹]
2 ¹²⁸ ×F F=1-2 ⁻²⁴	B	0	111 1111 0	111 1111 1111 1111 1111 1111
	H	0	110 0000	1111 1111 1111 1111 1111 1111
	D	0	100 1100 0101	1000 0000 1001 0010 1101 [3402823×10 ⁻³²]
2 ⁻²⁶⁰	B	Zero (number too small)		
	H	0	000 0000	0001 0000 0000 0000 0000 0000
	D	0	001 0101 0000	0111 1110 1111 0000 0101 [5397605×10 ⁻⁸⁵]
2 ²⁴⁸ ×F F=1-2 ⁻²⁴	B	Not representable		
	H	0	111 1110	1111 1111 1111 1111 1111 1111
	D	0	101 0010 1001	1010 1000 1100 1010 1000 [4523128×10 ⁺⁶⁸]
Explanation:				
B	BFP.			
BE or C	Biased exponent of BFP number, combination field of DFP number, or characteristic of HFP number.			
D	DFP.			
H	HFP.			
S	Sign.			

Figure 9-2. Examples of Floating-Point Numbers in Short Format

Value	S	BE or C	Trailing Significand
1.0	B 0	011 1111 1111	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H 0	100 0001	0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	D 0	010 0010 0011 01	00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0000 [10×10 ⁻¹]
0.5	B 0	011 1111 1110	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H 0	100 0000	1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	D 0	010 0010 0011 01	00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0101 [5×10 ⁻¹]
1/64	B 0	011 1111 1001	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H 0	011 1111	0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	D 0	010 0010 0010 00	00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0101 0111 0010 0101 [15625×10 ⁻⁶]
+0	B 0	000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H 0	000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	D 0	010 0010 0011 10	00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 [0×10 ⁰]
-0	B 1	000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H 1	000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	D 1	010 0010 0011 10	00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 [-0×10 ⁰]
-15.0	B 1	100 0000 0010	1110 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H 1	100 0001	1111 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	D 1	010 0010 0011 01	00 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1101 0000 [-150×10 ⁻¹]
20/7	B 0	100 0000 0000	0110 1101 1011 0110 1101 1011 0110 1101 1011 0110 1101 1011 0110 1101 1011 0111
	H 0	100 0001	0010 1101 1011 0110 1101 1011 0110 1101 1011 0110 1101 1011 0110 1101 1011 0110
	D 0	010 1001 1111 11	11 0101 1101 0011 0000 1011 0101 1101 0011 0000 1011 0101 1101 0110 0110 1101 [2857142857142857×10 ⁻¹⁵]
2 ⁻¹⁰²²	B 0	000 0000 0001	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H	Zero (number too small)	
	D 0	000 1001 0010 11	01 0010 0101 0001 1100 1110 0011 1110 1010 0001 1101 0000 0001 [2225073858507201×10 ⁻³²³]
2 ⁻¹⁰⁷⁴	B 0	000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001
	H	Zero (number too small)	
	D 0	001 0000 1110 11	00 1100 1100 1101 0101 1010 0101 1000 1000 0100 1010 0110 0101 [4940656458412465×10 ⁻³³⁹]
2 ¹⁰²⁴ × F F=1-2 ⁻⁵³	B 0	111 1111 1110	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111
	H	Not representable	
	D 0	100 0110 1100 11	11 1111 1011 1100 1110 1100 1011 0100 0101 1011 0001 1001 0110 [1797693134862316×10 ⁺²⁹³]
2 ⁻²⁶⁰	B 0	010 1111 1011	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H 0	000 0000	0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	D 0	011 0100 1100 00	01 1111 1011 1100 0001 0101 1100 0110 1010 1111 0000 0010 1000 [5397605346934×10 ⁻⁹⁴]
2 ²⁴⁸ × F F=1-2 ⁻⁵⁶	B 0	100 1111 0111	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
	H 0	111 1110	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111
	D 0	011 0011 0010 01	10 1010 0011 0010 1010 0010 0100 1011 0100 1111 0011 0110 0100 [4523128485832664×10 ⁺⁵⁹]
Explanation:			
B	BFP.		
BE or C	Biased exponent of BFP number, combination field of DFP number, or characteristic of HFP number.		
D	DFP.		
H	HFP.		
S	Sign.		

Figure 9-3. Examples of Floating-Point Numbers in Long Format

Effective Width

The resolution of a floating-point format may be described in terms of the spacing between adjacent representable numbers in the format. This is called an ulp (unit in the last place). Thus, for example, when rounding to nearest, the maximum error is 1/2 ulp. The magnitude of an ulp is directly related to the precision of the format. Another way to measure the resolution of the format in the vicinity of a particular

value is in terms of relative spacing, which is simply the value of an ulp divided by the value of the number. Relative spacing may be converted to a form called effective width, which is very similar to precision. The *effective width* (W_x) for any particular normalized floating-point value, v , is defined as:

$$W_x(v) = \log_x(v/u)$$

Where:

- u Value of a unit in the last position (ulp).
- v Value of the number.
- W_x Effective width represented in base x.
- x Any arbitrary base.

When effective width is measured using the native base for a particular format, it differs from the precision (p) of the format as shown in the following relation:

$$p-1 \leq W_b \leq p$$

Where:

- b Native base of a particular format.
- p Precision of the format.
- W_b Effective width for any normalized value represented in the native base b of the format.

The following figure shows the maximum and minimum values of the effective width for normalized numbers for all supported floating-point formats.

Length	Radix	W ₂		W ₁₀	
		Max	Min	Max	Min
Short	HFP	24.00	20.00	7.22	6.02
	BFP	24.00	23.00	7.22	6.92
	DFP	23.25	19.93	7.00	6.00
Long	HFP	56.00	52.00	16.86	15.65
	BFP	53.00	52.00	15.95	15.65
	DFP	53.15	49.83	16.00	15.00
Extended	HFP	112.00	108.00	33.72	32.51
	BFP	113.00	112.00	34.02	33.72
	DFP	112.95	109.62	34.00	33.00
Explanation:					
W ₂		effective width represented in base 2.			
W ₁₀		effective width represented in base 10.			

Figure 9-4. Maximum and Minimum Effective Width

Programming Note: The following example using a 3-digit decimal significand, may be helpful in understanding the concept of effective width and the maximum and minimum values. Consider the three adjacent representable numbers, Va, Vb, and Vc, as shown in the following figure:

Symbol or Formula	Value
Va	9.99
Vb	10.0
Vc	10.1
Vb-Va	0.01
Vc-Vb	0.1

The distance between points Va and Vb (Vb-Va) represents a resolution of one part in a thousand; whereas, the distance between points Vb and Vc (Vc-Vb) represents a resolution of one part in a hundred. In the first case, the effective width is three digits; in the second case, it is only two digits. For other values in the format, the effective width lies between these two extremes.

Floating-Point Data in Storage

All floating-point data formats appear in storage in the same left-to-right sequence as all other data formats. Bits of a data format that are numbered 0-7 constitute the byte in the leftmost (lowest-numbered) byte location in storage, bits 8-15 form the byte in the next sequential location, and so on. (See also the section “Storage Addressing” on page 3-2.)

Most of the floating-point instructions are defined in detail in this publication in Chapter 18, “Hexadecimal-Floating-Point Instructions,” Chapter 19, “Binary-Floating-Point Instructions,” and Chapter 20, “Decimal-Floating-Point Instructions.” This chapter, Chapter 9, defines in detail instructions called floating-point-support (FPS) instructions. The FPS instructions either have operands that may be in any floating-point format or have the function of converting between formats. This chapter also provides summary information about all of the floating-point instructions.

Registers And Controls

Floating-Point Registers

All floating-point instructions (FPS, BFP, DFP, and HFP) use the same 16 floating-point registers. The floating-point registers are identified by the numbers 0-15 and are designated by a four-bit R field in float-

ing-point instructions. Each floating-point register is 64 bits long and can contain either a short (32-bit) or a long (64-bit) floating-point operand.

A short floating-point datum requires only the left-most 32 bit positions of a floating-point register. The rightmost 32 bit positions of the register are ignored when the register is the source of an operand in the short format, and they remain unchanged when a short result is placed in the register.

A datum in the extended (128-bit) format occupies a register pair. Register pairs are formed by coupling the 16 registers as follows: 0 and 2, 4 and 6, 8 and 10, 12 and 14, 1 and 3, 5 and 7, 9 and 11, and 13 and 15.

Each of the eight pairs is referred to by the number of the lower-numbered register of the pair.

Additional Floating-Point (AFP) Registers

Floating-point registers 0, 2, 4, and 6 are ones that were originally available on ESA/390 models. The remaining 12 floating-point registers (1, 3, 5, and 7-15) were added to ESA/390 and are referred to as the additional floating-point (AFP) registers. The AFP registers can be used only if bit 45 of control register 0, the AFP-register-control bit, is one. Attempting to use an AFP register when the AFP-register-control bit is zero results in an AFP-register data exception (DXC 1).

Valid Floating-Point-Register Designations

Any installed register may be designated by an instruction to specify the register location of a short or long floating-point operand.

An instruction specifying a floating-point operand in the extended format must designate register 0, 1, 4, 5, 8, 9, 12, or 13; otherwise, a specification exception is recognized.

Floating-Point-Control (FPC) Register

The floating-point-control (FPC) register is a 32-bit register that contains mask bits, flag bits, a data-exception code, and two rounding-mode fields. An overview of the FPC register is shown in Figure 9-5.

Details are shown in Figure 9-6 and in Figure 9-8. (In Figure 9-6, the abbreviations “IM” and “SF” are based on the terms “interruption mask” and “status flag,” respectively.)

The bits of the FPC register are often referred to as, for example, FPC 1.0, meaning bit 0 of byte 1 of the register.

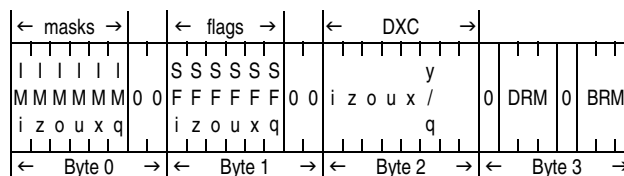


Figure 9-5. FPC Register Overview

Byte	Bit(s)	Name	Abbr.
0	0	IEEE-invalid-operation mask	IMi
0	1	IEEE-division-by-zero mask	IMz
0	2	IEEE-overflow mask	IMo
0	3	IEEE-underflow mask	IMu
0	4	IEEE-inexact mask	IMx
0	5	Quantum-exception mask	IMq
0	6-7	(Unassigned)	0
1	0	IEEE-invalid-operation flag	SFi
1	1	IEEE-division-by-zero flag	SFz
1	2	IEEE-overflow flag	SFo
1	3	IEEE-underflow flag	SFu
1	4	IEEE-inexact flag	SFx
1	5	Quantum-exception flag	SFq
1	6-7	(Unassigned)	0
2	0-7	Data-exception code	DXC
3	0	(Unassigned)	0
3	1-3	DFP rounding mode	DRM
3	4	(Unassigned)	0
3	5-7	BFP rounding mode	BRM

Figure 9-6. FPC-Register Bit Assignments

FPC Byte 3 Bits 1-3	Rounding Method
000	Round to nearest with ties to even
001	Round toward 0
010	Round toward $+\infty$
011	Round toward $-\infty$
100	Round to nearest with ties away from 0
101	Round to nearest with ties toward 0
110	Round away from 0
111	Round to prepare for shorter precision

Figure 9-7. DFP Rounding Mode

FPC Byte 3		
Bits 6-7 ¹	Bits 5-7 ²	Rounding Method
00	000	Round to nearest with ties to even
01	001	Round toward 0
10	010	Round toward $+\infty$
11	011	Round toward $-\infty$
—	100	Reserved / Invalid
—	101	Reserved / Invalid
—	110	Reserved / Invalid
—	111	Round to prepare for shorter precision

Explanation:

¹ Used when the floating-point extension facility is not installed.

² Used when the floating-point extension facility is installed. However, bits 5-7 may be used by the other architecture mode of the configuration. (See the section, “Impacts on ESA/390”, for details.)

— Does not apply

Figure 9-8. BFP Rounding Mode

IEEE Masks and Flags

When the floating-point extension facility is installed, the FPC register contains six IEEE mask bits and six IEEE flag bits that each correspond to one of the six IEEE exceptions that may be recognized when an IEEE computational instruction is executed. When the floating-point extension facility is not installed, the quantum exception is not recognized; the mask bit FPC 0.5 and the status flag bit FPC 1.5 corresponding to the quantum exception are unsupported and are zeros.

The mask bits, when one, in the FPC register cause an interruption to occur if an exception is recognized. If the mask bit for an exception is zero, the recognition of the exception causes the corresponding flag

bit to be set to one. Thus, a flag bit indicates whether the corresponding exception has been recognized at least once since the program last set the flag bit to zero. Except for PFPO, the mask bits are ignored, and the flag bits remain unchanged, when exceptions are recognized for floating-point-support (FPS) and HFP instructions.

The IEEE flag bits in the FPC register are set to zero only by explicit program action, initial CPU reset, clear reset, or power-on reset.

Note: The quantum exception is not part of the IEEE Standard 754-2008. However, since this data exception with its mask and status flag in the FPC is handled very much like the five IEEE exceptions, this architecture lists the quantum exception among the IEEE exceptions.

FPC DXC Byte

Byte 2 of the FPC register contains the data-exception code (DXC), which is an eight-bit code indicating the specific cause of a data exception. When the AFP-register-control bit, bit 45 of control register 0, is one and a program interruption causes the DXC to be placed at real location 147, the DXC is also placed in the DXC field of the FPC register. The DXC field in the FPC register remains unchanged when the AFP-register-control bit is zero or when any other program exception is reported. The DXC is described in “Data-Exception Code (DXC)” on page 6-14.

The DXC is a code, meaning it should be treated as an integer rather than as individual bits. However, when bits 6 and 7 are zero, bits 0-5 are bit significant; bits 0-4 (i,z,o,u,x) are trap flags and correspond to the same bits in bytes 0 and 1 of the FPC register (IEEE masks and IEEE flags), and bit 5 (y) is used in conjunction with bit 4, inexact (x), to indicate that the result has been incremented in magnitude. When the floating-point extension facility is installed, DXC bit 5 (q) is also the quantum-exception trap flag. The trap flag for an exception, instead of the IEEE flag, is set to one when an interruption for the exception is enabled by the corresponding IEEE mask bit.

Operations on the FPC Register

The following unprivileged instructions allow problem-state programs to operate on the FPC register:

```

EXTRACT FPC (DXC 2)
LOAD FPC (DXC 2)
LOAD FPC AND SIGNAL (DXC 3)

```


SET BFP ROUNDING MODE (DXC 2)
SET DFP ROUNDING MODE (DXC 3)
SET FPC (DXC 2)
SET FPC AND SIGNAL (DXC 3)
STORE FPC (DXC 2)

These instructions are subject to the AFP-register-control bit, bit 45 of control register 0. An attempt to execute any of the above instructions when the AFP-register-control bit is zero results in a data exception. The term (DXC 2) or (DXC 3) after the name indicates the type of exception: that is, BFP-instruction data exception, (DXC 2), or DFP-instruction data exception, (DXC 3), respectively.

Programming Note: The use of DXC 2 and DXC 3 for these floating-point-support (FPS) instructions is rather arbitrary, but assuming a valid operating system, these exceptions are never reported to the user, as the operating system will set bit 45 of control register 0 to one and re-issue the instruction.

AFP-Register-Control Bit

Bit 45 of control register 0 is the AFP-register-control bit. The AFP registers, the BFP instructions, the DFP instructions, and PFPO can be used successfully only when the AFP-register-control bit is one. Attempting to use one of the 12 additional floating-point registers or executing PFPO when the AFP-register-control bit is zero results in an AFP-register data exception (DXC 1). Attempting to execute any BFP instruction when the AFP-register-control bit is zero results in a BFP-instruction data exception (DXC 2). Attempting to execute any DFP instruction when the AFP-register-control bit is zero results in a DFP-instruction data exception (DXC 3).

DXC 2 and 3 are mutually exclusive and are of higher priority than any other DXC. Thus, for example, DXC 2 (BFP instruction) takes precedence over any IEEE exception; and DXC 3 (DFP instruction) takes precedence over any IEEE exception. As another example, if the conditions for both DXC 3 (DFP instruction) and DXC 1 (AFP register) exist, DXC 3 is reported.

If the conditions for both a data exception and a specification exception exist, it is unpredictable which exception is reported.

The initial value of the AFP-register-control bit is zero.

IEEE Computational Operations

Instructions which operate on BFP or DFP source operands or produce BFP or DFP results and recognize IEEE exceptions are called IEEE computational operations, and the associated operands and results are sometimes referred to as IEEE operands and IEEE results, respectively.

Execution of IEEE computational operations normally follows a certain pattern. Source operands are first tested for special cases and then processed to form a result. The priority for testing of special cases is SNaN, QNaN, infinity, and then finite number.

If a source operand is an SNaN, an IEEE-invalid-operation exception is recognized; and, in the non-trap case, the result is set to the corresponding QNaN.

In the absence of a source SNaN, if a source operand is a QNaN the result is set to that QNaN.

Handling of the case when more than one source operand is a NaN is covered in the individual instruction descriptions.

In the absence of a NaN, if an operand is an infinity, special “infinity arithmetic” rules are followed to produce the result.

Programming Note: PERFORM FLOATING-POINT OPERATION is an IEEE computational operation.

Intermediate Values

In the normal case (when all source operands are finite numbers), first, a *precise intermediate value* is produced, then a *rounded intermediate value*. In the absence of trapped overflow and trapped underflow, the rounded intermediate value is used as the *delivered value*. For trapped overflow and trapped underflow, the rounded intermediate value is divided by a *scale factor* to produce a *scaled value*, which is used as the delivered value. Finally, the delivered value is placed in the target location. For DFP, since the cohort for the delivered value may have multiple members, one member from the cohort must be selected. Selection is performed by choosing the member having a quantum closest to the *preferred*

quantum (or for trapped overflow and trapped underflow, a *scaled preferred quantum*).

Precise Intermediate Value

Every IEEE computational operation producing a numeric result computes a value called the *precise intermediate value*. This is the value that would have been computed were both the precision and exponent range unbounded.

Precision-Rounded Value

Except when otherwise specified, the precise intermediate value is rounded to the precision of the target format, but with unbounded exponent range. This process is called *target-precision-constrained rounding*, and the value selected by this type of rounding is called the *precision-rounded value*.

Denormalized Value

For IEEE targets, when the IEEE-underflow trap is disabled and the tininess condition exists, the precise intermediate value is rounded to fit in the destination format. That is, with both precision and exponent range of the target format. This process is called *denormalization rounding* and the value selected by this type of rounding is called the *denormalized value*.

Functionally-Rounded Value

The instructions CONVERT TO FIXED, LOAD FP INTEGER, QUANTIZE, and REROUND, as an integral part of the function, take a source operand and modify it to fit in a subset of the destination format. This process is called *functionally-constrained rounding* and the value selected by this type of rounding is called the *functionally-rounded value*.

Rounded Intermediate Value

In any particular instance, only one of the three rounding processes is performed and the value selected (precision-rounded value, denormalized value, or functionally-rounded value) is generically referred to as the *rounded intermediate value*.

Scaled Value

For both overflow and underflow when the associated trap is enabled, the precision-rounded value is scaled to bring it into the representable exponent range of the target. This is called the *scaled value* and is derived as explained in the next section.

Scale Factor (Ψ)

IEEE-overflow trap action and IEEE-underflow trap action produce a *scaled value* using a *scale factor*. The scaled value (z) is computed from the precision-rounded value (g) using the scale factor (Ψ) as follows:

$$z = g \div \Psi$$

Unsigned Scaling Exponent (α)

For BFP and DFP instructions, the scale factor (Ψ) is computed using the unsigned scaling exponent (α) and different formulas are used for overflow and underflow.

For overflow:

$$\Psi = b^{+\alpha}$$

For underflow:

$$\Psi = b^{-\alpha}$$

where b is the radix of the target (2 for BFP and 10 for DFP). The unsigned scaling exponent (α) depends on the type of operation and operand format.

For all BFP operations except LOAD ROUNDED, α depends on the target format and is 192 for short, 1536 for long, and 24576 for extended. For BFP LOAD ROUNDED, α depends on the source format, and is 512 for long and 8192 for extended.

For all DFP operations except LOAD ROUNDED, α depends on the target format and is 576 for long and 9216 for extended. For DFP LOAD ROUNDED, α depends on the source format, and is 192 for long and 3072 for extended.

Signed Scaling Exponent (Ω)

For PFPO convert floating-point radix, the scale factor (Ψ) is computed using the signed scaling exponent (Ω) and the formula for the scale factor (Ψ) is the same for overflow and underflow:

$$\Psi = b^{\Omega}$$

Where b is the radix (2, 10, and 16) for BFP, DFP and HFP, respectively. Thus, for overflow, the signed scaling exponent (Ω) is a positive value, and for underflow it is a negative value.

IEEE Rounding

Rounding takes an input value, and, using the effective rounding method, selects a value from a permissible set. The input value, considered to be infinitely-precise, may be an operand of an instruction or the numeric output from an arithmetic operation. The effective rounding method may be the current rounding method specified in the BFP-rounding-mode or DFP-rounding-mode field of the FPC register; or, for some instructions, an explicit rounding method is specified by a modifier field. For the PFPO-convert-floating-point-radix operation, rounding is specified by a field in an implicit register.

For target-precision-constrained rounding and denormalization rounding, the input is the precise intermediate value. For functionally-constrained rounding, the input is a source operand.

Permissible Set

Rounding selects a value from the permissible set. A permissible set is a set of values, and not representations; thus, for DFP, the selection of a member from the cohort is considered to be performed after rounding. A permissible set differs from the values representable in a particular format in the following ways:

1. A permissible set does not include infinity. Infinity is handled as a special case.
2. For target-precision-constrained rounding, the permissible set is considered to have an unbounded exponent range.
3. For denormalization rounding, the permissible set is limited to the values representable in a particular format.
4. For LOAD FP INTEGER, the permissible set contains only integral values.
5. For QUANTIZE, the permissible set is similar to LOAD FP INTEGER, containing only those values which are integral multiples of the requested quantum.
6. For REROUND, the permissible set contains only those values which can be represented within the requested precision.

Selection of Candidates

If a member of the permissible set is equal in value to the input value, then that member is selected; otherwise, two adjacent candidates with the same sign as

the input value are chosen from the permissible set. One candidate, called TZ (toward zero), is the member of the permissible set nearest to and smaller in magnitude than the input value; the other candidate, called AZ (away from zero), is the member of the permissible set nearest to and larger in magnitude than the input value. Which of the two candidates is selected depends on the rounding method.

Ties

Three rounding methods depend on a condition called a “tie.” This condition exists when the two candidates are equidistant from the input value.

Voting Digit and Common-Rounding-Point View

Two rounding methods depend on the value of the voting digit of each candidate. (Each “digit” is an integral value from zero to one less than the radix. Thus, a BFP digit is one bit, an HFP digit is four bits, and a DFP digit is a value from zero to nine.) The voting digit is the units digit of the significand when considered in the common-rounding-point view.

Without changing the value of a floating-point number, the significand may be viewed with the implied radix point in different positions, provided a corresponding adjustment is made to the exponent. In the common-rounding-point view, an implied radix point (called the common rounding point) and an associated exponent are selected for the input value and the two candidates, TZ and AZ. The common-rounding point is selected to satisfy the following requirements:

1. The input value and the two candidates all have the same exponent.
2. The significand of TZ is equal to the significand of the input value truncated at the rounding point.
3. The significand of AZ is one greater in magnitude than the significand of TZ.

Rounding Methods

Round to nearest with ties to even: The candidate nearest to the input value is selected. In case of a tie, the candidate selected is the one whose voting digit has an even value.

Round toward 0: The candidate that is smaller in magnitude is selected.

Round toward $+\infty$: The candidate that is algebraically greater is selected.

Round toward $-\infty$: The candidate that is algebraically less is selected.

Round to nearest with ties away from 0: The candidate nearest to the input value is selected. In case of a tie, the candidate selected is the one that is larger in magnitude.

Round to nearest with ties toward 0: The candidate nearest to the input value is selected. In case of a tie, the candidate selected is the one that is smaller in magnitude.

Round away from 0: The candidate that is greater in magnitude is selected.

Round to prepare for shorter precision: For a BFP or HFP permissible set, the candidate selected is the one whose voting digit has an odd value. For a DFP permissible set, the candidate that is smaller in magnitude is selected, unless its voting digit has a value of either 0 or 5; in that case, the candidate that is greater in magnitude is selected.

Programming Notes:

1. The two candidates are defined as: “nearest to and smaller” and “nearest to and larger”. Note that the simpler definition, “the two closest values,” is incorrect, as it is possible that the two closest values to the input value can be on the same side (toward zero). To illustrate this, Figure 9-9 on page 9-15 gives an example of rounding the integer value 10,000,003 to DFP short. Several members in the permissible set near this value are shown in three views: left-units view, right-units view, and integer view. The fourth column shows the distance of each member from the input value. In this example, the four closest members are all smaller in magnitude than the input value. Although the example is shown using DFP, corresponding examples can be shown for both BFP and HFP. In the extreme case, the number of members in the permissible set which are closer than candidate AZ, is 2, 10, and 16, for BFP, DFP, and HFP, respectively.
2. Except for rounding methods using the voting digit, rounding can normally be considered to be performed using values independent of the representation and the view. Figure 9-10 on

page 9-15 is an example, using the instruction LOAD FP INTEGER, to illustrate the concepts of different representations, views, rounding point, common-rounding-point view, and voting digit. In the example, the input value (source operand) is 29.5 and the two candidates are 29 and 30. The figure shows all three of these values using the left-units, right-units, and common-rounding-point views in both decimal and binary representations. Note that as shown in the left-units and right-units view columns, the rightmost digit is odd for both TZ and AZ. (But as represented in all BFP formats, these values would have an even rightmost digit; and as represented in DFP formats, these values could have an even or odd rightmost digit, depending on the quantum.) The voting digit can be identified in the common-rounding-point view as the digit to the immediate left of the rounding point. In both the decimal and binary representations, candidate TZ has an odd voting digit and candidate AZ has an even voting digit.

LOAD FP INTEGER produces consistent results between decimal and binary for five of the six rounding methods currently supported by the BFP instruction; for the round-to-prepare-for-shorter-precision rounding method, this is not always the case. Figure 9-11 on page 9-16 shows results of LOAD FP INTEGER for several input values for all eight rounding methods. Only in the round-to-prepare-for-shorter-precision method are the results different for decimal and binary.

3. An example of LOAD ROUNDED (long to short DFP) is shown in Figure 9-12 on page 9-16. In this example, the integer 99,999,995 is rounded from DFP long to DFP short. The cohort for candidate TZ in this case, has only one member and is odd. The cohort of candidate AZ has seven members, one of which is odd and the other six are even. The common-rounding-point view makes it clear that in this case, candidate AZ should be considered to be even.
4. The rightmost digit and the voting digit of a candidate are not necessarily the same digit. Figure 9-13 on page 9-16 and Figure 9-14 on page 9-16 give two examples using the instruction REROUND (long DFP). In both examples, the requested significance is one (Result Digits=1). The input value is 95 in the first example and 150 in the second example. Note that candidate AZ in the first example is the same value as

candidate TZ in the second example. In the first example, this member is considered to be even,

and in the second example it is considered to be odd.

Value in Permissible Set			Distance from 10000003	Notes
Left-Units View	Right-Units View	Integer View		
9.999996×10^6	9999996×10^0	9999996	-7	
9.999997×10^6	9999997×10^0	9999997	-6	*
9.999998×10^6	9999998×10^0	9999998	-5	*
9.999999×10^6	9999999×10^0	9999999	-4	*
1.000000×10^7	1000000×10^1	10000000	-3	TZ
1.000001×10^7	1000001×10^1	10000010	+7	AZ
1.000002×10^7	1000002×10^1	10000020	+17	
Explanation: * Closer than AZ. AZ Selected as candidate AZ (away from zero). TZ Selected as candidate TZ (toward zero).				

Figure 9-9. Choosing Candidates (short DFP)

Participant	Decimal			Binary ¹		
	Left-Units View	Right-Units View	Common-Rounding-Point View	Left-Units View	Right-Units View	Common-Rounding-Point View
Input Value	2.95×10^1	295×10^{-1}	29.5×10^0	1.11011×2^4	111011×2^{-1}	11101.1×2^0
Candidate TZ	2.9×10^1	29×10^0	29.0×10^0	1.1101×2^4	11101×2^0	11101.0×2^0
Candidate AZ	3.0×10^1	3×10^1	30.0×10^0	1.111×2^4	1111×2^1	11110.0×2^0
Explanation: ¹ Significand is shown in binary, exponent in decimal.						

Figure 9-10. LOAD FP INTEGER - Decimal and Binary

Input Value	Result When Effective Rounding Method Is								
	RNTE	RZ	RP	RM	RNTA	RNTZ	RA	RFS	
								Dec	Bin
-9.5	-10	-9	-9	-10	-10	-9	-10	-9	-9
-5.5	-6	-5	-5	-6	-6	-5	-6	-6	-5
-2.5	-2	-2	-2	-3	-3	-2	-3	-2	-3
-1.5	-2	-1	-1	-2	-2	-1	-2	-1	-1
-0.5	-0	-0	-0	-1	-1	-0	-1	-1	-1
+0.5	+0	+0	+1	+0	+1	+0	+1	+1	+1
+1.5	+2	+1	+2	+1	+2	+1	+2	+1	+1
+2.5	+2	+2	+3	+2	+3	+2	+3	+2	+3
+5.5	+6	+5	+6	+5	+6	+5	+6	+6	+5
+9.5	+10	+9	+10	+9	+10	+9	+10	+9	+9
Explanation:									
RNTE	Round to nearest with ties to even								
RZ	Round toward 0								
RP	Round toward $+\infty$								
RM	Round toward $-\infty$								
RNTA	Round to nearest with ties away from 0								
RNTZ	Round to nearest with ties toward 0								
RA	Round away from 0								
RFS	Round to prepare for shorter precision								

Figure 9-11. LOAD FP INTEGER Examples for All Rounding Methods

Participant	Left-Units View	Right-Units View	Common-Rounding-Point View
Input Value	9.9999995×10^7	99999995×10^0	9999999.5×10^1
Candidate TZ	9.999999×10^7	9999999×10^0	9999999.0×10^1
Candidate AZ	1.0×10^8	1×10^8	10000000.0×10^1

Figure 9-12. LOAD ROUNDED (long to short DFP)

Participant	Left-Units View	Right-Units View	Common-Rounding-Point View
Input Value	9.5×10^1	95×10^0	9.5×10^1
Candidate TZ	9.0×10^1	9×10^1	9.0×10^1
Candidate AZ	1.0×10^2	1×10^2	10.0×10^1

Figure 9-13. REROUND (long DFP), Result Digits=1, Input Value=95

Participant	Left-Units View	Right-Units View	Common-Rounding-Point View
Input Value	1.5×10^2	15×10^1	1.5×10^2
Candidate TZ	1.0×10^2	1×10^2	1.0×10^2
Candidate AZ	2.0×10^2	2×10^2	2.0×10^2

Figure 9-14. REROUND (long DFP), Result Digits=1, Input Value=150

Explicit Rounding Methods

The floating-point-support instruction CONVERT HFP TO BFP includes an M_3 modifier field which can specify any of five rounding methods. One HFP instruction (CONVERT TO FIXED); three BFP instructions (CONVERT TO FIXED, DIVIDE TO INTEGER, and LOAD FP INTEGER); and five DFP instructions (CONVERT TO FIXED, LOAD FP INTEGER, LOAD ROUNDED, QUANTIZE, and REROUND) also include either an M_3 modifier field or a similar M_4 modifier field. When the floating-point extension facility is installed, two more BFP instructions (CONVERT FROM LOGICAL and LOAD ROUNDED) and seven more DFP instructions (ADD, CONVERT FROM FIXED, CONVERT FROM LOGICAL, CONVERT TO LOGICAL, DIVIDE, MULTIPLY, and SUBTRACT) also include an M_3 or M_4 modifier field. The handling of an M_3 or M_4 value of zero depends on the type of instruction. For BFP instructions, an M_3 or M_4 value of zero causes rounding to be performed according to the current BFP rounding mode specified in the FPC register. For DFP instructions, an M_3 or M_4 value of zero causes rounding to be performed according to the current DFP rounding mode specified in the FPC register. Most floating-point-support and HFP instructions treat an M_3 or M_4 of zero the same as 5, that is, round toward zero. For

PFPO, a zero in the rounding field causes the current DFP rounding mode to be used.

Summary of Rounding Action

Figure 9-15 on page 9-17 summarizes the rounding action for floating-point-support (FPS), BFP, DFP, and HFP instructions.

Instruction	Rounding Action For			
	FPS Inst.	HFP Inst.	BFP Inst.	DFP Inst.
ADD	—	—	CBRM	MDD
ADD NORMALIZED	—	GD	—	—
ADD UNNORMALIZED	—	GD	—	—
CONVERT BFP TO HFP	E	—	—	—
CONVERT FROM FIXED	—	RTZ	MBB	MDD
CONVERT FROM LOGICAL	—	—	MA	MD
CONVERT HFP TO BFP	MC	—	—	—
CONVERT TO FIXED	—	M	MB	MD
CONVERT TO LOGICAL	—	—	MA	MD
DIVIDE	—	RTZ	CBRM	MDD
DIVIDE TO INTEGER	—	—	MB	—
HALVE	—	RTZ	—	—
LOAD FP INTEGER	—	RTZ	MB	MD
LOAD ROUNDED	—	RNTA	MBB	MD
MULTIPLY	—	RTZ	CBRM	MDD
MULTIPLY AND ADD	—	RTZ	CBRM	—
MULTIPLY AND ADD UNNORMALIZED	—	RTZ	—	—
MULTIPLY AND SUBTRACT	—	RTZ	CBRM	—
MULTIPLY UNNORMALIZED	—	RTZ	—	—
PERFORM FLOATING-POINT OPERATION	MBD	—	—	—
QUANTIZE	—	—	—	MD
REROUND	—	—	—	MD
SQUARE ROOT	—	RNTA	CBRM	—
SUBTRACT	—	—	CBRM	MDD
SUBTRACT NORMALIZED	—	GD	—	—
SUBTRACT UNNORMALIZED	—	GD	—	—

Figure 9-15. Comparison of Rounding Action

Instruction	Rounding Action For			
	FPS Inst.	HFP Inst.	BFP Inst.	DFP Inst.
Explanation:				
CBRM	Rounded according to current BFP rounding mode.			
CDRM	Rounded according to current DFP rounding mode.			
E	Result is exact, no rounding is required.			
GD	Round using a guard digit; see the instruction definition. This is almost, but not quite, round toward 0.			
M	Rounding is specified by a modifier field in the instruction. Five rounding methods are supported.			
MA	Rounding is specified by a modifier field in the instruction. Six rounding methods are supported; current BFP rounding mode can also be specified.			
MB	Rounding is specified by a modifier field in the instruction. When the floating-point extension facility is not installed, five rounding methods are supported; when the floating-point extension facility is installed, six rounding modes are supported; current BFP rounding mode can also be specified.			
MBB	When the floating-point extension facility is not installed, rounding is specified by the current BFP rounding mode; four rounding modes are supported. When the floating-point extension facility is installed, rounding is specified by a modifier field in the instruction; six rounding modes are supported; current BFP rounding mode can also be specified.			
MBD	Rounding is specified by a field in general register 0. Eight rounding methods are supported; current BFP rounding mode and current DFP rounding mode can also be specified.			
MC	Rounding is specified by a modifier field in the instruction. When the floating-point extension facility is not installed, five rounding methods are supported; when the floating-point extension facility is installed, six rounding modes are supported; current BFP rounding mode cannot be specified.			
MD	Rounding is specified by a modifier field in the instruction. Eight rounding methods are supported; current DFP rounding mode can also be specified.			
MDD	When the floating-point extension facility is not installed, rounding is specified by the current DFP rounding mode. When the floating-point extension facility is installed, rounding is specified by a modifier field in the instruction; eight rounding modes are supported; current DFP rounding mode can also be specified.			
RNTA	Round to nearest with ties away from 0.			
RTZ	Round toward 0.			

Figure 9-15. Comparison of Rounding Action (Continued)

IEEE Exceptions

This section defines handling of IEEE exceptions for most IEEE computational operations. For some instructions the action may differ from the general

rules; including, for example, a special controls to suppress recognition of certain IEEE exceptions. These differences are described in the individual instruction descriptions.

The action taken for each IEEE exception is controlled by a mask bit in the FPC register. When an IEEE exception is recognized, one of two actions is taken:

- If the corresponding mask bit in the FPC register is zero, a default action, called IEEE nontrap action, is taken, as specified for each condition, and the corresponding flag bit in the FPC register is set to one. Program execution then continues normally.
- If the corresponding mask bit in the FPC register is one, an action, called IEEE trap action, is taken for that exception, a program interruption for a data exception occurs, and the operation is suppressed or completed, depending on the exception, and the data-exception code (DXC) assigned for that exception is provided. For PFPO, a control bit in GR0 can select alternate exception handling, where overflow or underflow can be reflected by completion with a distinguished condition code, avoiding the interruption.

Concurrent IEEE Exceptions

IEEE-overflow or IEEE-underflow exception, IEEE-inexact exception, and quantum exception can coincide concurrently.

When the action for IEEE overflow (or underflow) is a nontrap action, the IEEE-overflow (or IEEE-underflow) flag bit in the FPC register is set to one and then the action for IEEE inexact (which could be a nontrap or trap action) occurs.

When the action for IEEE overflow (or underflow) is a trap action, the inexact exception is not recognized and not reported directly; instead, the DXC is set to indicate whether the result is exact, inexact and truncated, or inexact and incremented.

When the action for all concurrent IEEE exceptions, except the quantum exception, is a nontrap action, the flag bits in the FPC register corresponding to these nontrap concurrent exceptions are set to one, and then the action for the quantum exception (which could be a nontrap or trap action) occurs.

When the action for any concurrent IEEE exception, except the quantum exception, is a trap action, the quantum exception is not recognized, not reported directly, and not indicated in the DXC.

IEEE Invalid Operation

An IEEE-invalid-operation exception is recognized when, in the execution of an IEEE computational operation, any of the following occurs:

1. An SNaN is encountered in an IEEE computational operation.
2. A QNaN is encountered in an unordered-signaling comparison (COMPARE AND SIGNAL with a QNaN operand).
3. An IEEE difference is undefined (addition of infinities of opposite sign, or subtraction of infinities of like sign).
4. An IEEE product is undefined (zero times infinity).
5. An IEEE quotient is undefined (DIVIDE instruction with both operands zero or both operands infinity).
6. A BFP remainder is undefined (DIVIDE TO INTEGER with a dividend of infinity or a divisor of zero).
7. A BFP square root is undefined (negative non-zero operand).
8. Any other IEEE computational operation whose result is either undefined or not representable in the target format.

Even though an invalid-operation condition exists, the exception is not recognized if recognition of the exception is suppressed by means of an IEEE-invalid-operation-exception control (XiC).

IEEE-Invalid-Operation Nontrap Action: IEEE-invalid-operation nontrap action occurs when the IEEE-invalid-operation exception is recognized and the IEEE-invalid-operation mask bit in the FPC register is zero. The operation is completed and the IEEE-invalid-operation flag bit in the FPC register is set to one. The result of the operation depends on the type of operation and the operands.

If the instruction performs a comparison, the comparison result is *unordered*.

If the instruction is one that produces an IEEE result and none of the operands is a NaN, the result is the default QNaN.

If the instruction is one that produces an IEEE result and one of the operands is a NaN, that operand becomes the result unchanged, except that an SNaN is first converted to the corresponding QNaN and, for DFP, NaNs are canonicalized.

IEEE-Invalid-Operation Trap Action: IEEE-invalid-operation trap action occurs when the IEEE-invalid-operation exception is recognized and the IEEE-invalid-operation mask bit in the FPC register is one. The operation is suppressed, and the exception is reported as a program interruption for a data exception with DXC 80 hex.

IEEE Division-By-Zero

An IEEE-division-by-zero exception is recognized when in IEEE division the divisor is zero and the dividend is a nonzero finite number.

IEEE-Division-By-Zero Nontrap Action: IEEE-division-by-zero nontrap action occurs when the IEEE-division-by-zero exception is recognized and the IEEE-division-by-zero mask bit in the FPC register is zero. The operation is completed and the IEEE-division-by-zero flag bit in the FPC register is set to one. The result is set to an infinity with a sign that is the exclusive or of the dividend and divisor signs.

IEEE-Division-By-Zero Trap Action: IEEE-division-by-zero trap action occurs when the IEEE-division-by-zero exception is recognized and the IEEE-division-by-zero mask bit in the FPC register is one. The operation is suppressed, and the exception is reported as a program interruption for a data exception with DXC 40 hex.

IEEE Overflow

An IEEE-overflow exception is recognized for an IEEE target when the precision-rounded value of an IEEE computational operation is greater in magnitude than the largest finite number (Nmax) representable in the target format.

IEEE-Overflow Nontrap Action: IEEE-overflow nontrap action occurs when the IEEE-overflow exception is recognized and the IEEE-overflow mask bit in the FPC register is zero.

The operation is completed and the IEEE-overflow flag bit in the FPC register is set to one. The result of the operation depends on the sign of the precise intermediate value and on the effective rounding method:

1. For all round-to-nearest methods and round-away-from-0, the result is infinity with the sign of the precise intermediate value.
2. For round-toward-0 and round-to-prepare-for-shorter-precision, the result is the largest finite number of the format, with the sign of the precise intermediate value.
3. For round toward $+\infty$, the result is $+\infty$, if the sign is plus, or it is the negative finite number with the largest magnitude if the sign is minus.
4. For round toward $-\infty$, the result is the largest positive finite number if the sign is plus or $-\infty$, if the sign is minus.

IEEE-Overflow Trap Action: IEEE-overflow trap action occurs when the IEEE-overflow exception is recognized and the IEEE-overflow mask bit in the FPC register is one.

The operation is completed by setting the result to the scaled value and the exception is reported as a program interruption for a data exception with DXC 20, 28, or 2C hex, depending on whether the delivered value is exact, inexact and truncated, or inexact and incremented, respectively.

The result of the operation is derived from the precision-rounded value, the scale factor, and, for DFP, on the scaled preferred quantum. The value of the scale factor depends on the type of operation and operand format. The scaled preferred quantum for a particular operation is equal to the preferred quantum for that operation divided by the scale factor for that operation.

The delivered value is equal to the precision-rounded value divided by the scale factor. For DFP targets, the cohort member with the quantum nearest to the scaled preferred quantum is selected.

IEEE Underflow

An IEEE-underflow exception is recognized for an IEEE target when the tininess condition exists and either: (1) the IEEE-underflow mask bit in the FPC register is zero and the denormalized value is inex-

act, or (2) the IEEE-underflow mask bit in the FPC register is one.

The tininess condition exists when the precise intermediate value of an IEEE computational operation is nonzero and smaller in magnitude than the smallest normal number (Nmin) representable in the target format.

The denormalized value is inexact if it is not equal to the precise intermediate value.

IEEE-Underflow Nontrap Action: IEEE-underflow nontrap action occurs when the IEEE-underflow exception is recognized and the IEEE-underflow mask bit in the FPC register is zero.

The operation is completed and the IEEE-underflow flag bit in the FPC register is set to one. The result is set to the denormalized value. For DFP targets, the cohort member with the smallest quantum is selected.

IEEE-Underflow Trap Action: IEEE-underflow trap action occurs when the IEEE-underflow exception is recognized and the IEEE-underflow mask bit in the FPC register is one.

The operation is completed by setting the result to the scaled value and the exception is reported as a program interruption for a data exception with DXC 10, 18, or 1C hex, depending on whether the delivered value is exact, inexact and truncated, or inexact and incremented, respectively.

The result of the operation is derived from the precision-rounded value, the scale factor and, for DFP, on the scaled preferred quantum. The value of the scale factor depends on the type of operation and operand format. The scaled preferred quantum for a particular operation is equal to the preferred quantum for that operation divided by the scale factor for that operation.

The result is set to the precision-rounded value divided by the scale factor. For DFP targets, the cohort member with the quantum nearest to the scaled preferred quantum is selected.

IEEE Inexact

An IEEE-inexact exception is recognized when, for an IEEE computational operation, an inexact condition exists, recognition of the exception is not sup-

pressed, and neither IEEE-overflow trap action nor IEEE-underflow trap action occurs.

An inexact condition exists when the rounded intermediate value differs from the precise intermediate value. The condition also exists when IEEE-overflow nontrap action occurs. When the inexact condition exists, the delivered value and the result are said to be inexact.

Even though an inexact condition exists, the IEEE-inexact exception is not recognized if recognition of the exception is suppressed by means of an IEEE-inexact-exception control (XxC) or if IEEE overflow or IEEE underflow trap action occurs. When an inexact condition exists and the conditions for an IEEE-overflow trap action or IEEE-underflow trap action also apply, the trap action takes precedence and the inexact condition is reported in the DXC.

IEEE-Inexact Nontrap Action: IEEE-inexact nontrap action occurs when the IEEE-inexact exception is recognized and the IEEE-inexact mask bit in the FPC register is zero.

In the absence of another IEEE nontrap action, the operation is completed using the rounded intermediate value and the IEEE-inexact flag bit in the FPC register is set to one. For DFP targets, except for QUANTIZE and REROUND, the cohort member with the smallest quantum is selected.

When an IEEE-inexact nontrap action and another IEEE nontrap action coincide, the operation is completed using the result specified for the other exception and the flag bits for both exceptions are set to one.

IEEE-Inexact Trap Action: IEEE-inexact trap action occurs when the IEEE-inexact exception is recognized and the IEEE-inexact mask bit in the FPC register is one. The operation is completed and the exception is reported as a program interruption for a data exception with DXC 08 or 0C hex, depending on whether the result is inexact and truncated or inexact and incremented, respectively.

In the absence of a coincident IEEE nontrap action, the delivered value is set to the rounded intermediate value. For DFP targets, the cohort member with the smallest quantum is selected.

When the IEEE-inexact trap action coincides with an IEEE nontrap action, the operation is completed

using the result specified for the IEEE nontrap action, the flag bit for the nontrap exception is set to one, and the IEEE-inexact trap action takes place.

Quantum Exception

A quantum exception is recognized when floating-point extension facility is installed and when, for an IEEE computational operation, a quantum-exception condition exists, recognition of the exception is not suppressed, and none of IEEE-overflow trap action, IEEE-underflow trap action, and IEEE-inexact trap action occurs.

For computational operations except DIVIDE, LOAD FP INTEGER, PERFORM FLOATING-POINT OPERATION, QUANTIZE, and REROUND, a quantum-exception condition exists when the delivered DFP result is inexact, or when the delivered DFP result is exact and finite, but the delivered quantum is different from the preferred quantum.

For DIVIDE, a quantum-exception condition exists (1) when the delivered DFP result is inexact, or (2) when the dividend is a finite number and the divisor is an infinity, or (3) when the delivered DFP result is exact and is a finite number, the delivered quantum is the maximum quantum and is not the preferred quantum, and the rightmost significand digit is zero, or (4) when the delivered DFP result is exact and is a finite number, the delivered quantum is the minimum quantum and is greater than the preferred quantum. Note that the maximum quantum is the value of $1 \times 10^{Q_{\max}}$ and the minimum quantum is the value of $1 \times 10^{Q_{\min}}$, where Q_{\max} and Q_{\min} are the maximum and minimum right-units-view exponents, respectively.

For LOAD FP INTEGER, QUANTIZE, and REROUND, a quantum-exception condition exists when the delivered DFP result is a finite number, but the delivered quantum is different from the quantum of the source operand (the second operand for LOAD FP INTEGER, and the third operand for QUANTIZE and REROUND).

For PERFORM FLOATING-POINT OPERATION, a quantum-exception condition exists when the delivered quantum exceeds the preferred quantum of 1.

When floating-point extension facility is not installed, no quantum exception is recognized by executing any instruction.

Even though a quantum-exception condition exists, the exception is not recognized if recognition of the exception is suppressed or if IEEE-overflow, IEEE-underflow, or IEEE-inexact trap action occurs. When a quantum-exception condition exists and the conditions for an IEEE-overflow, IEEE-underflow, or IEEE-inexact trap action also apply, the trap action takes precedence and the quantum-exception condition is not reported.

Quantum-Exception Nontrap Action: Quantum-exception nontrap action occurs when the quantum exception is recognized and the quantum-exception mask bit in the FPC register is zero.

In the absence of another IEEE nontrap action, the operation is completed using the rounded intermediate value and the quantum-exception flag bit in the FPC register is set to one. The delivered result, if it is a finite number, is the cohort member with the quantum closest to the preferred quantum.

When a quantum-exception nontrap action and other concurrent IEEE nontrap actions coincide, the operation is completed using the result specified for the other exceptions and the flag bits for all exceptions are set to one.

Quantum-Exception Trap Action: Quantum-exception trap action occurs when the quantum-exception is recognized and the quantum-exception mask bit in the FPC register is one. The operation is completed and the exception is reported as a program interruption for a data exception with DXC 04 hex.

In the absence of a coincident IEEE nontrap action, the delivered value is set to the rounded intermediate value, and the delivered result, if it is a finite number, is the cohort member with the quantum closest to the preferred quantum.

When the quantum-exception trap action coincides with other IEEE nontrap actions, the operation is completed using the result specified for the other IEEE nontrap actions, the flag bits for the other IEEE nontrap exceptions are set to one, and the quantum-exception trap action takes place.

Programming Notes:

1. IEEE traps are reported by means of a program interruption for a data exception with a data-exception code.¹ The use of data exception pro-

vides the application program with a convenient interface since this exception is one of the original 15 exceptions in the System/360 architecture and is supported by most control programs that support the z/Architecture.

2. ANSI/IEEE Standard 754-1985 includes recommendations for the trap handler. When a system traps, the trap handler should be able to determine:

- a. Which exception(s) were recognized on this operation.
- b. The kind of operation that was being performed.
- c. The destination's format.
- d. For overflow, underflow, and inexact exceptions, the correctly rounded result, including information that might not fit in the destination's format.
- e. For invalid-operation and divide-by-zero exceptions, the operand values.

Items a and d are supplied as part of the interruption action. Items b, c, and e can be obtained starting with the instruction address in the old PSW and from this finding the instruction (which indicates the operation and format) and then the operands.

3. The description of underflow is one of the most difficult parts of the standard to understand. This is because:
 - a. The condition is described as two “correlated events” — “tininess” and “loss of accuracy”.
 - b. For tininess, ANSI/IEEE Standard 754-1985 provides two options for detection: “after rounding” or “before rounding”.
 - c. For loss of accuracy, ANSI/IEEE Standard 754-1985 provides two options for detection: “denormalization loss” or “inexact result”.
 - d. Implementation of the trap is optional.
 - e. The conditions to signal underflow are different depending on whether or not the trap is taken.

Each of the above items is discussed below.

- a. Tininess refers to a nonzero number strictly between $\pm N_{\min}$. (All subnormal numbers are in this range.) Loss of accuracy means that the result cannot be represented exactly.
- b. Detection of tininess after or before rounding differs only for the case when “rounding” would increase the magnitude of the result to exactly $\pm N_{\min}$. It must be noted, however, that the action which ANSI/IEEE Standard 754-1985 here calls “rounding” is not the rounding to produce the delivered result but rounding to compute an intermediate value having the precision of the result but “as though the exponent range were unbounded”. In fact, it is possible that the delivered result may not be tiny even though the intermediate value “after rounding” is tiny.

The option selected in the z/Architecture (and the Power architecture) is to detect tininess before rounding.

- c. The difference between detection of loss of accuracy as a denormalization loss or as an inexact result can best be understood by considering three intermediate values: (1) the precise intermediate value, which has unbounded precision and unbounded exponent range, (2) the precision-rounded value, which is obtained by rounding the precise intermediate value to the precision of the target format but with unbounded exponent range, and (3) the denormalized value, which is obtained by rounding the precise intermediate value to the precision and exponent range of the target format. An inexact condition is said to exist when the denormalized value (3) differs from the precise intermediate value (1). A denormalization loss is said to exist when the denormalized value (3) differs from the precision-rounded value (2). The two options differ in the case when the denormalized value (3) is equal to the precision-rounded value (2) but these are not equal to the precise intermediate value (1). Although ANSI/IEEE Standard 754-1985 uses the term “denormalization loss”, this

1. PFPO provides an option for trap action without the program interruption.

condition includes a case in which the delivered result is normal.

The option selected in the z/Architecture (and the Power architecture) is to detect “loss of accuracy” as an inexact result.

- d. Although ANSI/IEEE Standard 754-1985 does not require traps to be implemented for underflow or the other IEEE exceptions, it does state that “with each exception should be associated a trap under user control”. Since it also defines “should” as “that which is strongly recommended as being in keeping with the intent of the standard”, the z/Architecture provides traps by means of program interruptions.
- e. When the underflow trap is enabled, underflow is to be signaled when tininess is detected regardless of loss of accuracy. When the underflow trap is not enabled, the underflow flag bit is to be set only when both tininess and loss of accuracy have been detected. Add and subtract can result in tiny or inexact results, but not both. Thus, when underflow is disabled, add and subtract never set the underflow flag bit.

Suppression of Certain IEEE Exceptions

For IEEE-invalid-operation exception, IEEE-inexact exception, and quantum exception, a special control may be provided for an IEEE computational operation to control whether recognition of the exception is suppressed. These controls are called IEEE-invalid-operation-exception control (XiC), IEEE-inexact-exception control (XxC), and quantum-exception control (XqC), respectively.

When a special control is zero, recognition of the designated exception is not suppressed, and normal handling of the exception is performed. And, when the exception is recognized, it further depends on the mask bit in the FPC register to determine if a trap action occurs.

When a special control is one, recognition of the designated exception is suppressed, and no trap or non-trap action for the exception occurs. This suppression affects only recognition of the exception for setting the status flag or for causing the trap action. For

example, the IEEE-inexact-exception control (XxC) has no effect on the DXC; that is, the DXC for IEEE-overflow or IEEE-underflow exceptions along with the detail for exact, inexact and truncated, or inexact and incremented, is reported according to the actual condition. Also, it has no effect on recognition of a quantum exception, which may depend on the actual inexact condition regardless of the setting of IEEE-inexact-exception control.

Multiple exception controls may be provided to a single IEEE computational operation. In this case, the effect and setting of each control is independent of other controls. The specific exception controls that are provided are described in the individual instruction descriptions.

IEEE Same-Radix Format Conversion

The instructions LOAD LENGTHENED and LOAD ROUNDED perform conversions of data between the short, long, and extended formats, where the source and target operands are in the same radix. (For mixed-radix conversions, see PFPO.) For BFP and DFP formats, same-radix conversion involves adjustments to both the significand and the biased exponent. Conversion to a narrower format requires rounding of the significand and an IEEE-inexact exception may result.

When converting to a narrower format, adjustment of the biased exponent causes IEEE underflow if the resultant left-units-view (LUV) exponent would be less than the minimum LUV exponent (E_{min}), or IEEE overflow if the resultant LUV exponent would be greater than the maximum LUV exponent (E_{max}) for the new format.

Programming Notes:

1. When a NaN is converted to a narrower format, for BFP the appropriate number of payload bits on the right (or for DFP, the appropriate number of payload digits on the left) are simply dropped with no indication. This is unlike the conversion of nonzero numbers, where the loss of nonzero significand digits causes an IEEE-inexact exception. Thus, programs which encode NaN payloads for specific purposes must ensure that the distinguishing bits (or digits) are placed in the left part

of the payload for BFP and in the right part of the payload for DFP.

- For BFP, converting a NaN to a narrower format cannot turn the NaN into an infinity because an SNaN either causes an interruption or turns into a QNaN, and all QNaNs have a leftmost fraction bit of one.

IEEE Comparison

Comparisons are always exact and cannot cause IEEE-overflow or IEEE-underflow exceptions.

Comparison ignores the sign of zero, that is, $+0$ equals -0 .

Infinities with like sign compare equal, that is, $+\infty$ equals $+\infty$, and $-\infty$ equals $-\infty$.

A NaN compares as *unordered* with any other operand, whether a finite number, an infinity, or another NaN, including itself.

Two sets of instructions are provided: COMPARE and COMPARE AND SIGNAL. In the absence of QNaNs, these instructions work the same. These instructions work differently only when both of the following are true:

- Neither operand of the instruction is an SNaN
- At least one operand of the instruction is a QNaN

In this case, COMPARE simply sets condition code 3, but COMPARE AND SIGNAL recognizes the IEEE-invalid-operation exception. If any operand is an SNaN, both instructions recognize the IEEE-invalid-operation exception.

The action when the IEEE-invalid-operation exception is recognized depends on the IEEE-invalid-operation mask bit in the FPC register. If the mask bit is zero, then the instruction execution is completed by setting condition code 3, and the IEEE-invalid-operation flag in the FPC register is set to one. If the mask bit is one, then the exception is reported as a program interruption for a data exception with DXC 80 hex (IEEE invalid operation).

Programming Note: A compiler can select either COMPARE or COMPARE AND SIGNAL for a comparison, depending on whether it is desired that a QNaN to be recognized as an exception.

Condition Codes for IEEE Instructions

For those operations which set the condition code to indicate the value of an IEEE result, condition codes 0, 1, and 2 are set to indicate that the result is a zero of either sign, less than zero, or greater than zero, respectively. The condition-code setting depends only on an inspection of the rounded result. For comparison operations, condition codes 0, 1, and 2 indicate equal, low, or high, respectively. These settings are the same as for the HFP instructions.

Condition code 3 can also be set. After an arithmetic operation, condition code 3 indicates a NaN result of either sign. After a comparison, it indicates that a NaN was involved in the comparison (the unordered condition). See Figure 9-16.

CC	Arithmetic	Comparison
0	± 0	Equal
1	<0	Low
2	>0	High
3	$\pm \text{NaN}$	Unordered

Figure 9-16. Condition Codes

Instructions

The floating-point-support instructions and their mnemonics and operation codes are listed in Figure 9-17 on page 9-25. The figure indicates, in the column labeled “Characteristics”, the instruction format, when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

All floating-point-support instructions are subject to the AFP-register-control bit, bit 45 of control register 0. The AFP-register-control bit must be one when an AFP register is specified as an operand location; otherwise, an AFP-register data exception, DXC 1, is recognized.

Mnemonics for the floating-point instructions have an R as the last letter when the instruction is in the RR, RRE, or RRF format. Certain letters are used for floating-point instructions to represent operand-format length, as follows:

D Long
E Short
X Extended

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For a register-to-register operation using LOAD (short), for example, LER is the mnemonic and R_1, R_2 the operand designation.

Programming Notes:

1. The following additional floating-point-support instruction is available when the decimal-floating-point facility is installed:
 - SET DFP ROUNDING MODE (SRNMT)
2. The following additional floating-point-support instructions are available when the floating-point-support-sign-handling facility is installed:
 - COPY SIGN (CPSDR)
 - LOAD COMPLEMENT (LCDFR)
 - LOAD NEGATIVE (LNDFR)
 - LOAD POSITIVE (LPDFR)
3. The following additional floating-point-support instructions are available when the FPR-GR-transfer facility is installed:
 - LOAD FPR FROM GR (LDGR)
 - LOAD GR FROM FPR (LGDR)
4. The following additional floating-point-support instructions are available when the IEEE-exception-simulation facility is installed:
 - LOAD FPC AND SIGNAL (LFAS)
 - SET FPC AND SIGNAL (SFASR)
5. The following additional floating-point-support instructions are available when the long-displacement facility is installed:

- LOAD (LDY, LEY)
- STORE (STDY, STEY)

6. The following additional floating-point-support instruction is available when the PFPO facility is installed:

- PERFORM FLOATING POINT OPERATION (PFPO)

7. The following additional floating-point-support instruction and features are available when the floating-point extension facility is installed:

- A new floating-point-support instruction, SET BFP ROUNDING MODE (SRNMB), is added.
- A new exception, the quantum exception, is defined for some computational operations.
- Bit 5 of the floating-point-control (FPC) register is assigned to the quantum-exception mask
- Bit 13 of the FPC register is assigned to the quantum-exception flag.
- Data-exception code (DXC) 04 (hex) is assigned to the quantum exception.
- DXC 07 (hex) is assigned to the simulated quantum exception.
- The BFP-rounding-mode field in the FPC register is changed to 3 bits to support one additional BFP rounding mode, round to prepare for shorter precision.
- One new value of the effective rounding method field is assigned to support the round to prepare for shorter precision rounding method for CONVERT HFP TO BFP.
- For PFPO with a DFP result, bit 58 of general register 0 is assigned to be the DFP quantum-permission control.

Name	Mnemonic	Characteristics				Op Code	Page
CONVERT BFP TO HFP (long)	THDR	RRE C		Da		B359	9-27
CONVERT BFP TO HFP (short to long)	THDER	RRE C		Da		B358	9-27
CONVERT HFP TO BFP (long)	TBDR	RRF-e C	SP	Da		B351	9-28
CONVERT HFP TO BFP (long to short)	TBEDR	RRF-e C	SP	Da		B350	9-28
COPY SIGN (long)	CPSDR	RRF-b FS		Da		B372	9-29
EXTRACT FPC	EFPC	RRE		Db		B38C	9-30
LOAD (extended)	LXR	RRE	SP	Da		B365	9-30

Figure 9-17. Summary of Floating-Point-Support Instructions

Name	Mne- monic	Characteristics						Op Code	Page
LOAD (long)	LDR	RR			Da			28	9-30
LOAD (long)	LD	RX-a		A	Da		B ₂	68	9-30
LOAD (long)	LDY	RXY-a	LD	A	Da		B ₂	ED65	9-30
LOAD (short)	LER	RR			Da			38	9-30
LOAD (short)	LE	RX-a		A	Da		B ₂	78	9-30
LOAD (short)	LEY	RXY-a	LD	A	Da		B ₂	ED64	9-30
LOAD COMPLEMENT (long)	LCDFR	RRE	FS		Da			B373	9-30
LOAD FPC	LFPC	S		A	SP	Db	B ₂	B29D	9-31
LOAD FPC AND SIGNAL	LFAS	S	XF	A	SP	Dt Xg	B ₂	B2BD	9-31
LOAD FPR FROM GR (64 to long)	LDGR	RRE	FG		Da			B3C1	9-33
LOAD GR FROM FPR (long to 64)	LGDR	RRE	FG		Da			B3CD	9-33
LOAD NEGATIVE (long)	LNDFR	RRE	FS		Da			B371	9-33
LOAD POSITIVE (long)	LPDFR	RRE	FS		Da			B370	9-34
LOAD ZERO (extended)	LZXR	RRE			SP	Da		B376	9-34
LOAD ZERO (long)	LZDR	RRE			Da			B375	9-34
LOAD ZERO (short)	LZER	RRE			Da			B374	9-34
PERFORM FLOATING-POINT OPERATION	PFPO	E	PF		SP	Da Xi Xo Xu Xx Xq		010A	9-34
SET BFP ROUNDING MODE (2 bit)	SRNM	S			Db			B299	9-45
SET BFP ROUNDING MODE (3 bit)	SRNMB	S	F		SP	Db		B2B8	9-45
SET DFP ROUNDING MODE	SRNMT	S	TR		Dt			B2B9	9-45
SET FPC	SFPC	RRE			SP	Db		B384	9-46
SET FPC AND SIGNAL	SFASR	RRE	XF		SP	Dt Xg		B385	9-46
STORE (long)	STD	RX-a		A	Da		ST	B ₂	60
STORE (long)	STDY	RXY-a	LD	A	Da		ST	B ₂	ED67
STORE (short)	STE	RX-a		A	Da		ST	B ₂	70
STORE (short)	STEY	RXY-a	LD	A	Da		ST	B ₂	ED66
STORE FPC	STFPC	S		A	Db		ST	B ₂	B29C

Explanation:

A	Access exceptions for logical addresses.
B ₂	B ₂ field designates an access register in the access-register mode.
C	Condition code is set.
Da	AFP-register data exception.
Dt	DFP-instruction data exception.
F	Floating-point extension facility.
FG	FPR-GR-transfer facility.
FS	Floating-point-support-sign-handling facility.
LD	Long-displacement facility.
PF	PFPO facility.
RR	RR instruction format.
RRE	RRE instruction format.
RRF	RRF instruction format.
RX	RX instruction format.
RXY	RXY instruction format.
SP	Specification exception.
ST	PER storage-alteration event.
TR	Decimal-floating-point-rounding facility.
XF	IEEE-exception-simulation facility.
Xg	Simulated IEEE exception.
Xi	IEEE invalid-operation exception.
Xo	IEEE overflow exception.
Xq	Quantum exception, if the floating-point extension facility is installed.
Xu	IEEE underflow exception.
Xx	IEEE inexact exception.

Figure 9-17. Summary of Floating-Point-Support Instructions

CONVERT BFP TO HFP

Mnemonic	R_1, R_2	[RRE]
Op Code	////////	R_1 R_2
0	16	24 28 31

Mnemonic	Op Code	Operands
THDER	'B358'	Short BFP operand, long HFP result
THDR	'B359'	Long BFP operand, long HFP result

The second operand (the source operand) is converted from the binary-floating-point (BFP) format to the hexadecimal-floating-point (HFP) format, and the normalized result is placed at the first-operand location. The sign and magnitude of the source operand are tested to determine the setting of the condition code.

For numeric operands, the sign of the result is the sign of the source operand. If the source operand has a sign bit of one and all other operand bits are zeros, the result also is a one followed by all zeros.

When, for THDR, the characteristic of the result would be negative, the result is made all zeros but with the same sign as that of the source operand, and condition code 1 or 2 is set to indicate the sign of the source operand.

When, for THDR, the characteristic of the hexadecimal intermediate result is too large to fit into the target format, the result is set to all ones (that is, the largest-in-magnitude representable finite number) but with the same sign as that of the source operand, and condition code 3 is set.

See Figure 9-18 for a detailed description of the results of this instruction.

Resulting Condition Code:

- 0 Source was zero
- 1 Source was less than zero
- 2 Source was greater than zero
- 3 Special case

Program Exceptions:

- Data with DXC 1, AFP register

Programming Notes:

1. The BFP-to-HFP conversion instructions are summarized in Figure 9-19.
2. CONVERT BFP TO HFP (THDER) converts BFP operands in the short format to HFP operands in the long format, rather than converting short to short, to retain full precision. Using this long HFP result subsequently as a short operand requires no extra conversion steps.

Source Operand (a)	Results
$-\infty \leq a < -H_{\max}$	$T(-H_{\max})$, cc3
$-H_{\max} \leq a \leq -H_{\min}$	$T(r)$, cc1
$-H_{\min} < a < 0$	$T(-0)^1$, cc1
-0	$T(-0)$, cc0
+0	$T(+0)$, cc0
$0 < a < +H_{\min}$	$T(+0)^2$, cc2
$+H_{\min} \leq a \leq +H_{\max}$	$T(r)$, cc2
$+H_{\max} < a \leq +\infty$	$T(+H_{\max})$, cc3
NaN	$T(+H_{\max})$, cc3

Explanation:

¹ Condition code 1 is set to indicate the source was less than zero.

² Condition code 2 is set to indicate the source was greater than zero.

ccn Condition code is set to n.

r The value derived when the BFP source value a is converted to the HFP format. This result is always exact.

Hmax Largest (in magnitude) representable number in the target HFP format.

Hmin Smallest (in magnitude) representable normalized number in the target HFP format.

T(x) The value x is placed at the target operand location.

Figure 9-18. Results: CONVERT BFP TO HFP

Instruction	Mnemonic	Source		Target			Overflow, Underflow Possible
		Format	Significant Bits	Format	Significant Bits	Result	
CONVERT BFP TO HFP	THDER	BFP short	24	HFP long	53-56	Exact	No
	THDR	BFP long	53	HFP long	53-56	Exact	Yes
CONVERT HFP TO BFP	TBEDR	HFP long	53-56	BFP short	24	Rounded	Yes
	TBDR	HFP long	53-56	BFP long	53	Rounded	No

Figure 9-19. Summary of BFP-to/from-HFP Conversion Instructions

CONVERT HFP TO BFP

Mnemonic R_1, M_3, R_2 [RRF-e]

Op Code	M_3	////	R_1	R_2
0	16	20	24	28 31

Mnemonic	Op Code	Operands
TBEDR	'B350'	Long HFP operand, short BFP result
TBDR	'B351'	Long HFP operand, long BFP result

The second operand (the source operand) is converted from the hexadecimal-floating-point (HFP) format to the binary-floating-point (BFP) format, and the result rounded according to the rounding method specified by the M_3 field is placed at the first-operand location. The sign and magnitude of the source operand are tested to determine the setting of the condition code.

The M_3 field contains a modifier specifying a rounding method, as follows:

M_3 Effective Rounding Method

- 0 Round toward 0
- 1 Round to nearest with ties away from 0
- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$

A modifier other than 0, 1, or 3-7 is invalid. If the floating-point extension facility is not installed, an M_3 modifier of 3 is also invalid.

The sign of the result is the sign of the second operand. If the second operand has a sign bit of one and all other operand bits are zeros, the result also is a one followed by all zeros.

See Figure 9-20 on page 9-29 for a detailed description of the results of this instruction.

If the M_3 field designates any of the following invalid modifier values: 2 and 8-15, then a specification exception is recognized. When the floating-point extension facility is not installed, if the M_3 field designates the invalid value 3, it is undefined whether a specification exception is recognized or an unpredictable rounding method is performed.

Resulting Condition Code:

- 0 Source was zero
- 1 Source was less than zero
- 2 Source was greater than zero
- 3 Special case

Program Exceptions:

- Data with DXC 1, AFP register
- Specification

Programming Notes:

- The HFP-to-BFP conversion instructions are summarized in Figure 9-19 on page 9-28.
- Conversion to short BFP data requires HFP operands in the long format; a short HFP operand should be extended to long by ensuring that the right half of the register is cleared. Thus, the entire register should be cleared before loading a short HFP operand into it for conversion to BFP. This avoids unrepeatable rounding errors in the BFP result due to data left over from previous use.

Source Operand (a)	Results
$a < -N_{\max}$	See Part 2 of this figure.
$-N_{\max} \leq a \leq -N_{\min}$	T(r), cc1
$-N_{\min} < a \leq -D_{\min}$	T(d), cc1
$-D_{\min} < a < 0$	T(d) ¹ , cc1
-0	T(-0), cc0
+0	T(+0), cc0
$0 < a < +D_{\min}$	T(d) ² , cc2
$+D_{\min} \leq a < +N_{\min}$	T(d), cc2
$+N_{\min} \leq a \leq +N_{\max}$	T(r), cc2
$+N_{\max} < a$	See Part 2 of this figure.

Figure 9-20. (Part 1 of 2) Results: CONVERT HFP to BFP

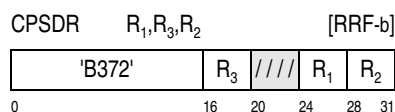
Source Operand (a)	Results for Rounding Method Specified in M_3					
	Round to Nearest with Ties away from 0	Round to Nearest with Ties to Even	Round to Prepare for Shorter Precision	Round toward 0	Round toward $+\infty$	Round toward $-\infty$
$a < -N_{\max}$	T($-\infty$), cc3	T($-\infty$), cc3	T($-N_{\max}$), cc3	T($-N_{\max}$), cc3	T($-N_{\max}$), cc3	T($-\infty$), cc3
$+N_{\max} < a$	T($+\infty$), cc3	T($+\infty$), cc3	T($+N_{\max}$), cc3	T($+N_{\max}$), cc3	T($+\infty$), cc3	T($+N_{\max}$), cc3

Explanation:

¹ Condition code 1 is set for this case, even when the rounded result is zero.
² Condition code 2 is set for this case, even when the rounded result is zero.
ccn Condition code is set to n.
d The denormalized value derived when the HFP source value a is rounded to the format of the target using the rounding method specified in the M_3 field.
r The value derived when the HFP source value a is rounded to the format of the target using the rounding method specified in the M_3 field.
Dmin Smallest (in magnitude) representable subnormal number in the target BFP format.
Nmax Largest (in magnitude) representable finite number in the target BFP format.
Nmin Smallest (in magnitude) representable normal number in the target BFP format.
T(x) The value x is placed at the target operand location.

Figure 9-20. (Part 2 of 2) Results: CONVERT HFP to BFP

COPY SIGN



The second operand is placed at the first-operand location with the sign bit set to the sign of the third operand. The first, second, and third operands are each in a 64-bit floating-point register. The sign bit of the second operand and bits 1-63 of the third operand are ignored.

Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 1, AFP register
- Operation (if the floating-point-support-sign-handling facility is not installed)

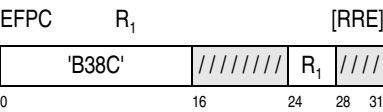
Programming Notes:

- COPY SIGN is radix independent and can be used to operate on HFP, BFP, or DFP operands or even to copy the sign between operands having different radices.
- Since the sign is in the same bit position (the left-most bit) in all widths for all radices, the third

operand can be a short, long, or extended format.

3. COPY SIGN can be used in conjunction with LOAD (LDR) to copy the sign of an operand in the extended format.

EXTRACT FPC



The contents of the FPC (floating-point-control) register are placed in bit positions 32-63 of the general register designated by R₁. Bit positions 0-31 of the general register remain unchanged.

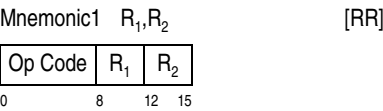
Condition Code: The code remains unchanged.

IEEE Exceptions: None.

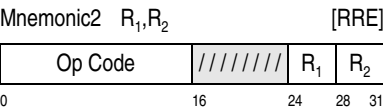
Program Exceptions:

- Data with DXC 2, BFP instruction

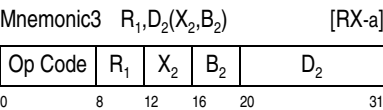
LOAD



Mnemonic1	Op Code	Operands
LER	'38'	Short
LDR	'28'	Long

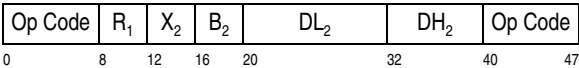


Mnemonic2	Op Code	Operands
LXR	'B365'	Extended



Mnemonic3	Op Code	Operands
LE	'78'	Short
LD	'68'	Long

Mnemonic4 R₁,D₂(X₂,B₂) [RXY-a]



Mnemonic4	Op Code	Operands
LEY	'ED64'	Short
LDY	'ED65'	Long

The second operand is placed unchanged at the first-operand location.

The operation is performed without inspecting the contents of the second operand; no arithmetic exceptions are recognized.

For LXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

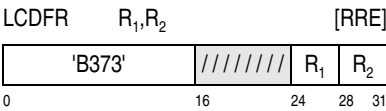
The displacement for LE and LD is treated as a 12-bit unsigned binary integer. The displacement for LEY and LDY is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of LE, LD, LEY, and LDY)
- Data with DXC 1, AFP register
- Operation (LEY and LDY, if the long-displacement facility is not installed)
- Specification (LXR only)

LOAD COMPLEMENT



The second operand is placed at the first-operand location with the sign bit inverted. Both the first and second operands are each in a 64-bit floating-point register.

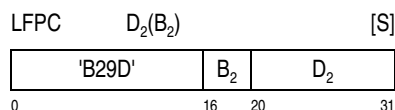
Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 1, AFP register
- Operation (if the floating-point-support-sign-handling facility is not installed)

Programming Notes:

1. LOAD COMPLEMENT (LCDFR) is radix independent and can be used to operate on HFP, BFP, or DFP operands.
2. LOAD COMPLEMENT (LCDFR) can be used in conjunction with LOAD (LDR) to set the sign of an operand in the extended format.

LOAD FPC

The four-byte second operand in storage is loaded into the FPC (floating-point-control) register.

Bits corresponding to unsupported bit positions in the FPC register must be zero; otherwise, a specification exception is recognized. For purposes of this checking, a bit position is considered to be unsupported only if it is either unassigned or assigned to a facility which is not installed in any architectural mode of the configuration.

When the floating-point extension facility is installed, the bits corresponding to the BFP rounding mode must specify a valid rounding mode; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

Program Exceptions:

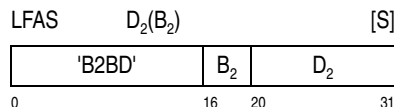
- Access (fetch, operand 2)
- Data with DXC 2, BFP instruction
- Specification

Programming Notes:

1. When the architectural mode of a CPU is changed without resetting the FPC register (by means of the set architecture signal-processor order, for example) the entire contents of the FPC register are preserved. This is true even when some of these bits are associated with a facility which is not available in both architectural modes. Checking for unsupported bit positions of

the FPC register is performed independently of the current architectural mode of the CPU; thus, it is always safe to restore the FPC register from a value previously saved on this CPU, even after the architectural mode has changed.

2. When the floating-point extension facility is installed, bits 29-31 of the second operand must specify a valid BFP rounding mode and bits 6-7, 14-15, 24, and 28 must be zero; otherwise, a specification exception is recognized.
3. When the floating-point extension facility is not installed, bits 5-7, 13-15, 24, and 28-29 are unassigned and must be zero; otherwise, a specification exception is recognized.

LOAD FPC AND SIGNAL

First, flags of byte 1 of the floating-point-control (FPC) register at the beginning of the operation are preserved to be used as signaling flags. Next, the contents of the source operand are placed in the FPC register; then the flags in the FPC register are set to the logical OR of the signaling flags and the source flags. Finally, the conditions for simulated-IEEE-exception trap action are examined.

The source operand is the second operand in storage.

If any signaling flag is one and the corresponding source mask is also one, simulated-IEEE-exception trap action occurs. The data-exception code (DXC) in the FPC register is updated to indicate the specific cause of the interruption and a data-exception program interruption occurs at completion of the instruction execution. The DXC for the interruption is shown in Figure 9-21.

If no signaling flag is enabled, the DXC in the FPC register remains as loaded from the source and instruction execution completes with no trap action.

See Figure 9-22 for a detailed description of the result of this instruction.

Bits in the source operand that correspond to unsupported bit positions in the FPC register must be zero;

otherwise, a specification exception is recognized. For purposes of this checking, a bit position is considered to be unsupported only if it is either unassigned or assigned to a facility which is not installed in any architectural mode of the configuration.

When the floating-point extension facility is installed, the bits corresponding to the BFP rounding mode must specify a valid rounding mode; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Data with DXC 3, DFP instruction
- Data with DXC for simulated IEEE exception
- Operation (if the IEEE-exception-simulation facility is not installed)
- Specification

Enabled ¹ Signaling Flags Bit 0 1 2 3 4 5	DXC (Binary)
1 - - - - -	1000 0011
0 1 - - - - -	0100 0011
0 0 1 - - - -	0010 w011
0 0 0 1 - - -	0001 w011
0 0 0 0 1 - -	0000 1011
0 0 0 0 0 1	0000 0111 ²

Explanation:

¹ The logical AND of the corresponding bit in the source masks and the signaling flags.

² Is only supported when the floating-point extension facility is installed.

- Don't care.

w Bit 4 of the signaling flags.

Figure 9-21. DXC for Simulated-IEEE-Exception Trap Action

Trap	Resulting FPC Register Contents				
	Masks	Flags	DXC	DRM	BRM
No	S	OR	S	S	S
Yes	S	OR	Xg	S	S

Explanation:

BRM BFP rounding mode

DRM DFP rounding mode

DXC Data-exception code

OR Set to the logical OR of signaling flags and source flags.

S Set to the contents of the corresponding field in the source operand.

Trap Simulated-IEEE-exception trap action. This action occurs when the logical AND of the signaling flags and source masks is nonzero.

Xg DXC for simulated IEEE exception. See Figure 9-21

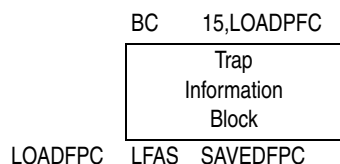
Figure 9-22. Result: LFAS and SFASR

Programming Notes:

1. When the floating-point extension facility is installed, bits 29-31 of the second operand must specify a valid BFP rounding mode and bits 6-7, 14-15, 24, and 28 must be zero; otherwise, a specification exception is recognized.
2. When the floating-point extension facility is not installed, bits 5-7, 13-15, 24, and 28-29 are unassigned and must be zero; otherwise, a specification exception is recognized.
3. The IEEE-exception-simulation instructions (LOAD FPC AND SIGNAL and SET FPC AND SIGNAL) are provided to facilitate program simulation of IEEE operations not provided directly in the machine. These instructions permit the simulation routine to restore the caller's masks, flags, and rounding modes; and, when appropriate, to simulate IEEE exceptions, including traps when enabled.
4. On entry, the simulation routine should first save the FPC register contents (containing the caller's masks, flags, and rounding modes), disable all traps, clear all flags, establish appropriate rounding modes, and then perform the necessary floating-point operations. Finally, the routine should set the current flags (called signaling flags) appropriately, and invoke either LOAD FPC AND SIGNAL or SET FPC AND SIGNAL.
5. When a program interruption for a data exception occurs, and both bit 6 and 7 of the DXC are ones, it indicates to the trap handler that the interruption was caused by a simulated-IEEE-

exception trap action. The remaining bits of the DXC indicate the type of IEEE exception. Additional information required by the trap handler can be determined by the contents of a trap-information block, located by convention, at a fixed offset from the location where the instruction causing the trap resides.

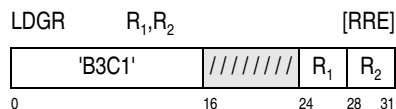
The following shows an example of this scheme. The block and the code below are inserted at an appropriate place where a trap can occur. In this example, the LOAD FPC AND SIGNAL instruction is used, and the caller's FPC register contents are at the location, SAVEDFPC.



The trap information block can contain the designation of floating-point registers and general registers containing the information (or addresses of the information) for the following types of items:

- The operation being simulated
- Format and values of source operands
- Format and value of the result
- Any additional information useful to the trap handler.

LOAD FPR FROM GR



The second operand is placed at the first-operand location. The second operand is in a general register, and the first operand is in a floating-point register.

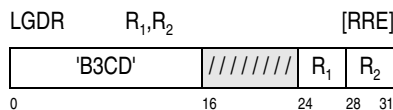
Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 1, AFP register

- Operation (if the FPR-GR-transfer facility is not installed)

LOAD GR FROM FPR



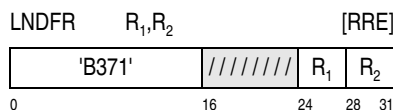
The second operand is placed at the first-operand location. The second operand is in a floating-point register, and the first operand is in a general register.

Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 1, AFP register
- Operation (if the FPR-GR-transfer facility is not installed)

LOAD NEGATIVE



The second operand is placed at the first-operand location with the sign bit set to one. Both the first and second operands are each in a 64-bit floating-point register.

Condition Code: The code remains unchanged.

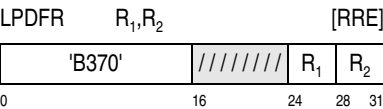
Program Exceptions:

- Data with DXC 1, AFP register
- Operation (if the floating-point-support-sign-handling facility is not installed)

Programming Notes:

- LOAD NEGATIVE (LNDFR) is radix independent and can be used to operate on HFP, BFP, or DFP operands.
- LOAD NEGATIVE (LNDFR) can be used in conjunction with LOAD (LDR) to set the sign of an operand in the extended format.

LOAD POSITIVE



The second operand is placed at the first-operand location with the sign bit set to zero. Both the first and second operands are each in a 64-bit floating-point register.

Condition Code: The code remains unchanged.

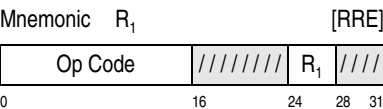
Program Exceptions:

- Data with DXC 1, AFP register
- Operation (if the floating-point-support-sign-handling facility is not installed)

Programming Notes:

1. LOAD POSITIVE (LPDFR) is radix independent and can be used to operate on HFP, BFP, or DFP operands.
2. LOAD POSITIVE (LPDFR) can be used in conjunction with LOAD (LDR) to set the sign of an operand in the extended format.

LOAD ZERO



Mnemonic	Op Code	Operands
LZER	'B374'	Short
LZDR	'B375'	Long
LZXR	'B376'	Extended

All bits of the first operand are set to zeros.

For LZXR, The R₁ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

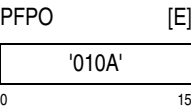
Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 1, AFP register
- Specification (LZXR only)

Programming Note: LOAD ZERO sets all bits of a register to zeros, which produces a positive zero value in the BFP, DFP, and HFP formats. For DFP, addition of an all-zero value to any normal value, X, normalizes X.

PERFORM FLOATING-POINT OPERATION



The operation specified by the function code in general register 0 is performed and the condition code is set to indicate the result. When there are no exceptional conditions, condition code 0 is set. When an IEEE nontrap exception is recognized, condition code 1 is set. When an IEEE trap exception with alternate action is recognized, condition code 2 is set. A 32-bit return code is placed in bits 32-63 of general register 1; bits 0-31 of general register 1 remain unchanged.

The PERFORM FLOATING-POINT OPERATION (PFPO) instruction is subject to the AFP-register-control bit, bit 45 of control register 0. For PFPO to be executed successfully, the AFP-register-control bit must be one; otherwise, an AFP-register data exception, DXC 1, is recognized.

Bit 32 of general register 0 is the test bit. When bit 32 is zero, the function specified by bits 33-63 of general register 0 is performed; each field in bits 33-63 must be valid and the combination must be a valid and installed function; otherwise a specification exception is recognized. When bit 32 is one, the function specified by bits 33-63 is not performed but, instead, the condition code is set to indicate whether these bits specify a valid and installed function; the condition code is set to 0 if the function is valid and installed, or to 3 if the function is invalid or not installed. This will be useful if additional functions are assigned in the future. This definition is written as if the test bit is zero except when stated otherwise.

Bits 33-39 of GR0 specify the operation type. Only one operation type is currently defined: 01, hex, is PFPO Convert Floating-Point Radix.

For the PFPO-convert-floating-point-radix operation, other fields in general register 0 include first-operand

format, second operand format, control flags, and rounding method. For the PFPO-convert-floating-point-radix operation, the second operand is converted to the format of the first operand and placed at the first-operand location, a return code is placed in bits 32-63 of general register 1, and the condition code is set to indicate whether an exceptional condition was recognized.

The first and second operands are in implicit floating-point registers. The first operand is in FPR0 (paired with FPR2 for extended). The second operand is in FPR4 (paired with FPR6 for extended).

General Register 0 (GR0)

The following figures summarize fields and bits in general register 0 (GR0).

GR0 Bit(s)	Name and Detail
0-31	Reserved/Ignored
32	Test Bit
33-39	PFPO-operation-type code
40-47	PFPO-operand-format code for operand 1
48-55	PFPO-operand-format code for operand 2
56	Inexact-suppression control
57	Alternate-exception-action control
58-59	Target-radix-dependent controls
60-63	PFPO rounding method

Figure 9-23. Fields in GR0 for PFPO

Op Type Code (hex)	Operation
00	Reserved/Invalid
01	PFPO Convert Floating-Point Radix
02-7F	Reserved/Invalid

Figure 9-24. PFPO-Operation-Type Code (GR0 bits 33-39)

Code (hex)	Format
00	HFP short
01	HFP long
02	HFP extended
03	Reserved/Invalid
04	Reserved/Invalid
05	BFP short
06	BFP long
07	BFP extended
08	DFP short
09	DFP long
0A	DFP extended
0B-FF	Reserved/Invalid

Figure 9-25. PFPO-Operand-Format Codes (GR0 bits 40-47 and 48-55)

Radix of Target	GR0 Bit	Meaning
HFP	58	HFP-overflow control
HFP	59	HFP-underflow control
BFP	58	Reserved, must be zero
BFP	59	Reserved, must be zero
DFP	58	Reserved, must be zero when the floating-point extension facility is not installed; it is the DFP quantum-permission control (DQPC) when the facility is installed.
DFP	59	DFP preferred-quantum control (DPQC)

Figure 9-26. PFPO-Target-Radix-Dependent Controls (GR0, bits 58-59)

Value	Method*
0	According to current DFP rounding mode
1	According to current BFP rounding mode
2-7	Reserved/Invalid

Figure 9-27. PFPO-Rounding Method (GR0 bits 60-63)

Value	Method*
8	Round to nearest with ties to even
9	Round toward 0
10	Round toward $+\infty$
11	Round toward $-\infty$
12	Round to nearest with ties away from 0
13	Round to nearest with ties toward 0
14	Round away from 0
15	Round to prepare for shorter precision
* All rounding methods, other than those listed as reserved, are valid for any conversion function, regardless of the radix of the result.	

Figure 9-27. PFPO-Rounding Method (GR0 bits 60-63)

Figure 9-28 shows the basic PFPO functions, that is, the minimum set of combinations of the operation-type, first-operand-format, and second-operand-format fields which are valid and installed functions when the PFPO instruction is installed.

GR0 bits 33-55 (hex)	Function	Exceptions
010500	Convert HFP short to BFP short	SP Da Xo Xu Xx
010600	Convert HFP short to BFP long	SP Da
010700	Convert HFP short to BFP extended	SP Da
010800	Convert HFP short to DFP short	SP Da Xx Xq
010900	Convert HFP short to DFP long	SP Da Xx Xq
010A00	Convert HFP short to DFP extended	SP Da Xx Xq
010501	Convert HFP long to BFP short	SP Da Xo Xu Xx
010601	Convert HFP long to BFP long	SP Da Xx
010701	Convert HFP long to BFP extended	SP Da
010801	Convert HFP long to DFP short	SP Da Xx Xq
010901	Convert HFP long to DFP long	SP Da Xx Xq
010A01	Convert HFP long to DFP extended	SP Da Xx Xq
010502	Convert HFP extended to BFP short	SP Da Xo Xu Xx
010602	Convert HFP extended to BFP long	SP Da Xx
010702	Convert HFP extended to BFP extended	SP Da
010802	Convert HFP extended to DFP short	SP Da Xx Xq
010902	Convert HFP extended to DFP long	SP Da Xx Xq
010A02	Convert HFP extended to DFP extended	SP Da Xx Xq
010005	Convert BFP short to HFP short	SP Da Xi Xx
010105	Convert BFP short to HFP long	SP Da Xi
010205	Convert BFP short to HFP extended	SP Da Xi
010805	Convert BFP short to DFP short	SP Da Xi Xx Xq
010905	Convert BFP short to DFP long	SP Da Xi Xx Xq
010A05	Convert BFP short to DFP extended	SP Da Xi Xx Xq
010006	Convert BFP long to HFP short	SP Da Xi Xo Xu Xx
010106	Convert BFP long to HFP long	SP Da Xi Xo Xu Xx
010206	Convert BFP long to HFP extended	SP Da Xi Xo Xu Xx
010806	Convert BFP long to DFP short	SP Da Xi Xo Xu Xx Xq
010906	Convert BFP long to DFP long	SP Da Xi Xx Xq
010A06	Convert BFP long to DFP extended	SP Da Xi Xx Xq
010007	Convert BFP extended to HFP short	SP Da Xi Xo Xu Xx
010107	Convert BFP extended to HFP long	SP Da Xi Xo Xu Xx
010207	Convert BFP extended to HFP extended	SP Da Xi Xo Xu Xx
010807	Convert BFP extended to DFP short	SP Da Xi Xo Xu Xx Xq
010907	Convert BFP extended to DFP long	SP Da Xi Xo Xu Xx Xq
010A07	Convert BFP extended to DFP extended	SP Da Xi Xx Xq
010008	Convert DFP short to HFP short	SP Da Xi Xo Xu Xx
010108	Convert DFP short to HFP long	SP Da Xi Xo Xu Xx

Figure 9-28. Basic PFPO Functions (GR0 bits 33-55)

GR0 bits 33-55 (hex)	Function	Exceptions
010208	Convert DFP short to HFP extended	SP Da Xi Xo Xu Xx
010508	Convert DFP short to BFP short	SP Da Xi Xo Xu Xx
010608	Convert DFP short to BFP long	SP Da Xi Xx
010708	Convert DFP short to BFP extended	SP Da Xi Xx
010009	Convert DFP long to HFP short	SP Da Xi Xo Xu Xx
010109	Convert DFP long to HFP long	SP Da Xi Xo Xu Xx
010209	Convert DFP long to HFP extended	SP Da Xi Xo Xu Xx
010509	Convert DFP long to BFP short	SP Da Xi Xo Xu Xx
010609	Convert DFP long to BFP long	SP Da Xi Xo Xu Xx
010709	Convert DFP long to BFP extended	SP Da Xi Xx
01000A	Convert DFP extended to HFP short	SP Da Xi Xo Xu Xx
01010A	Convert DFP extended to HFP long	SP Da Xi Xo Xu Xx
01020A	Convert DFP extended to HFP extended	SP Da Xi Xo Xu Xx
01050A	Convert DFP extended to BFP short	SP Da Xi Xo Xu Xx
01060A	Convert DFP extended to BFP long	SP Da Xi Xo Xu Xx
01070A	Convert DFP extended to BFP extended	SP Da Xi Xo Xu Xx
Explanation: Da AFP-register data exception. SP Specification exception. Xi IEEE-invalid-operation exception. Xo IEEE-overflow exception. Xq Quantum exception, if the floating-point extension facility is installed. Xu IEEE-underflow exception. Xx IEEE-inexact exception.		

Figure 9-28. Basic PFPO Functions (GR0 bits 33-55)

In the detailed descriptions that follow there are many references to quantum exceptions. Those apply only when the floating-point extension facility is installed, and bit 58 of general register 0 is one.

Inexact-Suppression Control: Bit 56 of general register 0 is the inexact-suppression control. When the inexact-suppression control is zero, IEEE-inexact exceptions are recognized and reported in the normal manner. When the inexact-suppression control is one, IEEE-inexact exceptions are not recognized, and no trap or nontrap action for the exceptions occurs. The inexact-suppression control has no effect on the DXC; that is, the DXC for IEEE-overflow or IEEE-underflow exceptions along with the detail for exact, inexact and truncated, or inexact and incremented, is reported according to the actual condition.

Alternate-Exception-Action Control: Bit 57 of general register 0 is the alternate-exception-action control. The setting of this control affects the action taken for IEEE-overflow and IEEE-underflow trap exceptions.

When the alternate-exception-action control is zero, IEEE-overflow and IEEE-underflow trap exceptions are reported in the normal manner. That is, the appropriate data exception code (DXC) is placed in byte 2 of the floating-point control register, the operation is completed, and a program interruption for a

data exception occurs. (As part of the program interruption, the DXC is stored at location 147.) This is called an *IEEE trap exception with normal action*.

When the alternate-exception-action control is one, the DXC is placed in byte 2 of the floating-point control register, the operation is completed, condition code 2 is set, and program execution continues with the next sequential instruction. (There is no program interruption and the DXC is not stored at location 147.) This is called an *IEEE trap exception with alternate action*.

HFP-Overflow Control: Bit 58 of general register 0 is the HFP-overflow control. When the HFP-overflow control is zero, an HFP-overflow condition is reported as an IEEE-invalid-operation exception and is subject to IEEE-invalid-operation mask. When the HFP-overflow control is one, an HFP-overflow condition is reported as an IEEE-overflow exception and is subject to the IEEE-overflow mask.

The HFP-overflow control is defined only for HFP targets; when other than an HFP target is specified, this bit must be zero.

HFP-Underflow Control: For HFP targets, bit 59 of general register 0 is the HFP alternate underflow control. When the HFP-underflow control is zero, HFP underflow causes the result to be set to a true zero with the same sign as the source and underflow is not reported. (The result in this case is inexact and subject to the inexact-suppression control.) When the HFP-underflow control is one, the condition is reported as an IEEE-underflow exception and is subject to the IEEE-underflow mask.

Bit 59 of general register 0 is defined only for HFP and DFP targets; when a BFP target is specified, this bit must be zero.

DFP Quantum-Permission Control (DQPC):

When the floating-point extension facility is installed, for DFP targets, bit 58 of general register 0 is the DFP quantum-permission control (DQPC). If this control is zero, recognition of the quantum exception is suppressed so that no trap or nontrap action for the exception occurs; if the control is one, recognition of the quantum exception is not suppressed.

DFP Preferred Quantum Control (DPQC): For DFP targets, bit 59 of general register 0 is the DFP preferred quantum control (DPQC). For radix conver-

sion with DFP targets, if the delivered value is inexact, the cohort member with the smallest quantum is selected; if the delivered value is exact, selection depends on the value of bit 59 of general register 0, the DFP preferred quantum control (DPQC). When the delivered value is exact and the DPQC bit is zero, the cohort member with the largest quantum is selected. When the delivered value is exact and the DPQC bit is one, the preferred quantum is one and the cohort member with the quantum closest to one is selected.

Return Code

Regardless of what condition code is set, and independent of whether the test bit is one, a 32-bit return code is placed in bits 32-63 of general register 1; bits 0-31 of general register 1 remain unchanged. A return code is also placed in general register 1 when a program interruption occurs for an IEEE trap exception that completes; general register 1 is not updated when a program interruption occurs for an IEEE trap exception that suppresses. Thus, general register 1 is updated on a program interruption for IEEE-overflow, IEEE-underflow, IEEE-inexact, and quantum-exceptiontrap exceptions, but is not updated on a program interruption for an IEEE-invalid-operation trap exception.

Except where otherwise specified, the return code is a value of zero.

Sign Preservation

For PFPO convert floating-point radix, the sign of the result is the same as the sign of the source. The only exception to this is when the source is a NaN and the target is HFP; in this case, the result is the largest representable number in the target HFP format (Hmax) with the sign set to plus.

Preferred Quantum

For radix conversion with DFP targets, if the delivered value is inexact, the cohort member with the smallest quantum is selected; if the delivered value is exact, selection depends on the value of bit 59 of general register 0, the DFP preferred quantum control (DPQC). When the delivered value is exact and the DPQC bit is zero, the cohort member with the largest quantum is selected. When the delivered value is exact and the DPQC bit is one, the preferred quantum is one and the cohort member with the quantum closest to one is selected.

NaN Conversion

When converting between DFP and BFP, the sign of the NaN is always preserved, and the value of the payload is preserved, when possible. If the value of the source payload exceeds the maximum value of the target payload, the target is set to the default QNaN, but with the same sign as the source.

When traps are disabled, an SNaN is converted to the corresponding QNaN, and the payload is preserved, when possible; that is, SNaN(*x*) is converted to QNaN(*x*), where *x* is the value of the payload. For DFP, both QNaN(0) and SNaN(0) can be represented; but in BFP, there is a representation for QNaN(0), but not for SNaN(0).

Scaled Value and Signed Scaling Exponent (Ω) for PFPO

When, for the PFPO-convert-floating-point-radix operation, IEEE-overflow trap action or IEEE-underflow trap action occurs, the scaled value is computed using the following steps:

$$\Psi = b^{\Omega}$$

$$z = g \div \Psi$$

Where Ω is the signed scaling exponent, *b* is the target radix (2, 10, or 16), Ψ is the scale factor, *g* is the precision-rounded value, and *z* is the scaled value.

The signed scaling exponent (Ω) is selected to make the magnitude of the value of the scaled result (*z*) lie in the range:

$$1 \leq |z| < b.$$

The value of the signed scaling exponent (Ω), treated as a 32-bit signed binary integer, is placed in bits 32-63 of general register 1; bits 0-31 of general register 1 remain unchanged.

The scaled value is used as the delivered value and is placed in the result location. For DFP targets, the cohort member with the quantum nearest to the scaled preferred quantum is selected. (But it should be noted that for all currently supported conversions, the result is always inexact, so the cohort member with the smallest quantum is selected.) For BFP targets, there are no redundant representations, there is only one member in a cohort. For HFP targets, the result is normalized.

HFP Values

Unnormalized HFP values are accepted on input, but all HFP results are normalized. If an HFP result would be less than the smallest (in magnitude) representable normalized number, an HFP underflow condition exists.

HFP Zero Result

For PFPO-convert-BFP-to-HFP or PFPO-convert-DFP-to-HFP functions, if the source operand is zero, the result has the same sign as that of the source operand, and has all zeros in fraction and characteristic.

HFP Overflow and Underflow for PFPO

For an HFP target of a PFPO-convert-floating-point-radix operation, the handling of overflow and underflow conditions is controlled by the HFP-overflow control and the HFP-underflow control, respectively.

HFP Overflow: An HFP-overflow condition exists when an HFP target precision's largest number (*Hmax*) is exceeded in magnitude by the precision-rounded value. That is, when the characteristic of a normalized HFP result would exceed 127 and the fraction is not zero.

When the HFP-overflow control is zero, HFP-overflow is reported as an IEEE-invalid-operation exception and is subject to the IEEE-invalid-operation mask in the FPC register. This is called an *HFP-overflow-as-IEEE-invalid-operation condition*.

When the HFP-overflow control is one, HFP overflow is reported as an IEEE-overflow exception and is subject to the IEEE-overflow mask in the FPC register. This is called an *HFP-overflow-as-IEEE-overflow condition*.

HFP Underflow: An HFP-underflow condition exists when the precision-rounded value is nonzero and less in magnitude than the HFP target precision's smallest normalized number, *Hmin*. That is, when the characteristic of a normalized HFP result would be less than zero and the fraction is not zero. The result is set to a true zero with the same sign as the source. Reporting of the HFP-underflow condition is subject to the HFP-underflow control. The result in this case, however, is inexact and is subject to the controls for that condition.

When the HFP-underflow control is zero, the HFP-underflow condition is not reported.

When the HFP-underflow control is one, HFP underflow is reported as an IEEE-underflow exception and is subject to the IEEE-underflow mask in the FPC register. This is called an *HFP-underflow-as-IEEE-underflow condition*.

IEEE Exceptions for PFPO

Except where otherwise stated, the following sections on IEEE exceptions apply to both BFP and DFP.

IEEE Invalid Operation: An IEEE-invalid-operation exception is recognized when any of the following occurs:

1. An SNaN is encountered in an IEEE source.
2. In an IEEE-to-HFP conversion, a NaN or an infinity is encountered in the IEEE source.
3. An HFP-overflow-as-IEEE-invalid-operation condition exists.

IEEE-invalid-operation exceptions are recognized as either IEEE-invalid-operation nontrap exceptions or IEEE-invalid-operation trap exceptions.

IEEE-Invalid-Operation Nontrap Action

IEEE-invalid-operation nontrap action occurs when an IEEE-invalid-operation exception is recognized and the IEEE-invalid-operation mask bit in the FPC register is zero. The operation is completed, the IEEE-invalid-operation flag bit in the FPC register is set to one, and condition code 1 is set. The result is as follows:

- When the target is IEEE and the source is an IEEE SNaN, the result is the source NaN converted to the corresponding canonical QNaN in the target format.
- When the target is HFP and the source is an IEEE NaN, the result is the largest representable number in the target HFP format (Hmax) with the sign set to plus.
- When the target is HFP and the source is an IEEE infinity, the result is Hmax with the same sign as the source.
- When an HFP-overflow-as-IEEE-invalid-operation condition exists, the result is Hmax with the same sign as the source.

IEEE-Invalid-Operation Trap Action

IEEE-invalid-operation trap action occurs when an IEEE-invalid-operation exception is recognized and

the IEEE-invalid-operation mask bit in the FPC register is one. The operation is suppressed, and the exception is reported as a program interruption for a data exception with DXC 80 hex.

IEEE Overflow: For IEEE targets, an IEEE-overflow exception is recognized when the precision-rounded value is greater in magnitude than the largest finite number (Nmax) representable in the target format. For HFP targets, an IEEE-overflow exception is recognized when the HFP-overflow condition exists and the HFP-overflow control is one.

IEEE-Overflow Nontrap Action

IEEE-overflow nontrap action occurs when the IEEE-overflow exception is recognized and the IEEE-overflow mask bit in the FPC register is zero.

The operation is completed and the IEEE-overflow flag bit in the FPC register is set to one. For IEEE targets, the result of the operation depends on the sign of the precise intermediate value and on the effective rounding method:

1. For all round-to-nearest methods and round-away-from-0, the result is infinity with the sign of the precise intermediate value.
2. For round-toward-0 and round-to-prepare-for-shorter-precision, the result is the largest finite number of the format, with the sign of the precise intermediate value.
3. For round toward $+\infty$, the result is $+\infty$ if the sign is plus, or it is the negative finite number with the largest magnitude if the sign is minus.
4. For round toward $-\infty$, the result is the largest positive finite number if the sign is plus or $-\infty$ if the sign is minus.

For HFP targets, the result is set to the largest representable number in the target HFP format (Hmax) with the same sign as the source.

Additional action depends on whether there is also an IEEE-inexact exception or a quantum exception.

When IEEE-overflow nontrap action occurs, and neither IEEE-inexact exception nor quantum exception has been recognized, the IEEE-overflow flag bit in the FPC register is set to one and condition code 1 is set.

When an IEEE-overflow nontrap action occurs together with an IEEE-inexact nontrap action, or with a quantum-exception nontrap action, or with nontrap actions for both IEEE-inexact and quantum-exception, the IEEE flag bits in the FPC register for all the recognized exceptions are set to ones and condition code 1 is set.

When IEEE-overflow nontrap action and IEEE-inexact trap action occur, the condition code is not set, the IEEE-overflow flag bit in the FPC register is set to one, and the IEEE-inexact exception is reported as a program interruption for a data exception with DXC 08 or 0C hex, depending on whether the result is inexact and truncated or inexact and incremented, respectively.

When IEEE-overflow nontrap action and quantum-exception trap action occur, the condition code is not set, the IEEE-overflow flag bit in the FPC register is set to one, and the quantum exception is reported as a program interruption for a data exception with DXC 04 hex. When in addition an IEEE-inexact nontrap action occurs, the IEEE-inexact flag bit in the FPC register is set to one.

IEEE-Overflow Trap Action

IEEE-overflow trap action occurs when the IEEE-overflow exception is recognized and the IEEE-overflow mask bit in the FPC register is one.

The operation is completed by setting the result to the scaled value; placing the value of the signed scaling exponent (Ω), treated as a 32-bit signed binary integer in bits 32-63 of general register 1; and setting DXC 20, 28, or 2C hex, depending on whether the delivered value is exact, inexact and truncated, or inexact and incremented, respectively.

For DFP targets, the delivered value is always inexact and the cohort member with the smallest quantum is selected.

Additional action depends on the value of the alternate-exception-action control.

When the alternate-exception-action control is zero, the condition code is not set and the exception is reported as a program interruption for a data exception.

When the alternate-exception-action control is one, condition code 2 is set and no program interruption occurs.

IEEE Underflow: For IEEE targets, an IEEE-underflow exception is recognized when the tininess condition exists and either: (1) the IEEE-underflow mask bit in the FPC register is zero and the denormalized value is inexact, or (2) the IEEE-underflow mask bit in the FPC register is one.

The tininess condition exists when the precise intermediate value of an IEEE computational operation is nonzero and smaller in magnitude than the smallest normal number (N_{min}) representable in the target format.

The denormalized value is inexact if it is not equal to the precise intermediate value.

For HFP targets, an IEEE-underflow exception is recognized when the HFP-underflow condition exists and the HFP-underflow control is one.

IEEE-Underflow Nontrap Action

IEEE-underflow nontrap action occurs when the IEEE-underflow exception is recognized and the IEEE-underflow mask bit in the FPC register is zero.

The operation is completed and the IEEE-underflow flag bit in the FPC register is set to one.

For IEEE targets, the result is set to the denormalized value. For DFP targets, the cohort member with the smallest quantum is selected.

For HFP targets, the result is set to a true zero with the same sign as the source.

Additional action depends on whether there is also an IEEE-inexact exception or a quantum exception.

When IEEE-underflow nontrap action occurs, and neither IEEE-inexact exception nor quantum exception has been recognized, the IEEE-underflow flag bit in the FPC register is set to one and condition code 1 is set.

When an IEEE-underflow nontrap action occurs together with an IEEE-inexact nontrap action, or with a quantum-exception nontrap action, or with nontrap actions for both IEEE-inexact and quantum-exception, the IEEE flag bits in the FPC register for all the recognized exceptions are set to ones and condition code 1 is set.

When IEEE-underflow nontrap action and IEEE-inexact trap action occur, the condition code is not set,

the IEEE-underflow flag bit in the FPC register is set to one, and the IEEE-inexact trap exception is reported as a program interruption for a data exception with DXC 08 or 0C hex, depending on whether the result is inexact and truncated or inexact and incremented, respectively.

When IEEE-underflow nontrap action and quantum-exception trap action occur, the condition code is not set, the IEEE-underflow flag bit in the FPC register is set to one, and the quantum exception is reported as a program interruption for a data exception with DXC 04 hex. When in addition an IEEE-inexact nontrap action occurs, the IEEE-inexact flag bit in the FPC register is set to one.

IEEE-Underflow Trap Action

IEEE-underflow trap action occurs when the IEEE-underflow exception is recognized and the IEEE-underflow mask bit in the FPC register is one.

The operation is completed by setting the result to the scaled value; placing the value of the signed scaling exponent (Ω), treated as a 32-bit signed binary integer in bits 32-63 of general register 1; and setting DXC 10, 18, or 1C hex, depending on whether the result is exact, inexact and truncated, or inexact and incremented, respectively.

For DFP targets, the delivered value is always inexact and the cohort member with the smallest quantum is selected.

Additional action depends on the value of the alternate-exception-action control.

When the alternate-exception-action control is zero, the condition code is not set and the exception is reported as a program interruption for a data exception.

When the alternate-exception-action control is one, condition code 2 is set and no program interruption occurs.

IEEE Inexact: An IEEE-inexact exception is recognized when, for a PFPO-convert-floating-point-radix operation, an inexact condition exists, recognition of the exception is not suppressed, and neither IEEE-overflow trap action nor IEEE-underflow trap action occurs.

An inexact condition exists when the rounded intermediate value differs from the precise intermediate

value. The condition also exists when IEEE-overflow nontrap action occurs.

Even though an inexact condition exists, the IEEE-inexact exception is not recognized if the inexact-suppression control is one or if IEEE overflow or IEEE underflow trap action occurs. When an inexact condition exists and the conditions for an IEEE-overflow trap action or IEEE-underflow trap action also apply, the trap action takes precedence and the inexact condition is reported in the DXC.

IEEE-Inexact Nontrap Action

IEEE-inexact nontrap action occurs when the IEEE-inexact exception is recognized and the IEEE-inexact mask bit in the FPC register is zero.

In the absence of another IEEE nontrap action, the operation is completed using the rounded intermediate value, condition code 1 is set, and the IEEE-inexact flag bit in the FPC register is set to one. For DFP targets, the cohort member with the smallest quantum is selected.

When an IEEE-inexact nontrap action and an IEEE-overflow or IEEE-underflow nontrap action coincide, the operation is completed using the result specified for the other exception and the flag bits for both exceptions are set to one, and condition code 1 is set. When in addition, a quantum-exception nontrap action occurs, the quantum-exception flag bit in the FPC register is also set to one.

IEEE-Inexact Trap Action

IEEE-inexact trap action occurs when the IEEE-inexact exception is recognized and the IEEE-inexact mask bit in the FPC register is one. The operation is completed, the condition code is not set, and the exception is reported as a program interruption for a data exception with DXC 08 or 0C hex, depending on whether the result is inexact and truncated or inexact and incremented, respectively.

In the absence of a coincident IEEE nontrap action, the delivered value is set to the rounded intermediate value. For DFP targets, the cohort member with the smallest quantum is selected.

When the IEEE-inexact trap action coincides with an IEEE nontrap action, the operation is completed using the result specified for the IEEE nontrap action, the flag bit for the nontrap exception is set to one, and the IEEE-inexact trap action takes place.

Quantum Exception: A quantum exception is recognized when the floating-point extension facility is installed, and when a quantum-exception condition exists, recognition of the exception is not suppressed ($DQPC = 1$), and none of IEEE-overflow trap action, IEEE-underflow trap action, and IEEE-inexact trap action occurs.

For a PFPO-convert-floating-point-radix operation with DFP result, a quantum-exception condition exists when the delivered DFP result is inexact, or when the delivered DFP result is exact and finite, but the delivered quantum exceeds the preferred quantum of 1.

When the floating-point extension facility is not installed, no quantum exception is recognized.

Even though a quantum-exception condition exists, the quantum exception is not recognized if recognition of the exception is suppressed ($DQPC = 0$), or if IEEE-overflow, IEEE-underflow, or IEEE-inexact trap action occurs. When a quantum-exception condition exists, and the conditions for an IEEE-overflow, IEEE-underflow, or IEEE-inexact trap action also apply, the trap action takes precedence and the quantum exception is not reported in status flag or DXC.

Quantum-Exception Nontrap Action

Quantum-exception nontrap action occurs when the quantum exception is recognized and the quantum-exception mask bit in the FPC register is zero.

In the absence of another IEEE nontrap action, the operation is completed using the rounded intermediate value, condition code 1 is set, and the quantum-exception flag bit in the FPC register is set to one.

When a quantum-exception nontrap action and another one or two IEEE nontrap actions coincide, the operation is completed using the result specified for the other exceptions and the flag bits for all two or three exceptions are set to one, and condition code 1 is set.

Quantum-Exception Trap Action

Quantum-exception trap action occurs when the quantum exception is recognized and the quantum-exception mask bit in the FPC register is one. The operation is completed, the condition code is not set, and the exception is reported as a program interruption for a data exception with DXC 04 hex.

In the absence of a coincident IEEE nontrap action, the delivered value is set to the rounded intermediate value.

When the quantum-exception trap action coincides with other IEEE nontrap actions, the operation is completed using the result specified for the other IEEE nontrap actions, the flag bits for the nontrap exceptions are set to one, and the quantum-exception trap action takes place.

Resulting Condition Code (when test bit is zero):

- 0 Normal result
- 1 Nontrap exception
- 2 Trap exception with alternate action
- 3 --

Resulting Condition Code (when test bit is one):

- 0 Function is valid
- 1 --
- 2 --
- 3 Function is invalid

IEEE Exceptions:

- Invalid operation
- Overflow
- Underflow
- Inexact

Program Exceptions:

- Data with DXC 1, AFP register
- Data with DXC for IEEE exception
- Operation (if the PFPO facility is not installed)
- Specification
- Quantum (if the floating-point extension facility is installed)

Programming Notes:

1. The PFPO-convert-floating-point-radix operation performs "correct rounding"; that is, the result is accurately obtained from the precise intermediate value using the effective rounding method. This is in contrast to some radix conversion programs, which may produce results with larger rounding errors.
2. Note that a value of zero in the rounding method field (GR0 bits 60-63) specifies rounding according to the current DFP rounding mode (FPC 3.1-

- 3) regardless of whether the target is DFP, BFP, or HFP.
- If the program does not supply a default QNaN (payload = zero) as a source, then its presence as a result indicates that the machine either created a new QNaN or the program attempted to convert a payload which was too large for this format.
 - If the program restricts payloads to values within the smallest payload capacity of any format to be converted to, then payloads are preserved across radix conversions.
 - In the absence of suppression, bits 32-63 of general register 1 are always set to a return code, even though, in most cases, the return code is zero. Future extensions of PFPO may make more use of nonzero return codes. With the current definition, the only nonzero return codes are set for IEEE-overflow trap exceptions and IEEE-underflow trap exceptions. In this case, the value of the signed scaling exponent (Ω), is placed in bits 32-63 of general register 1. For IEEE-inexact trap exceptions, the return code is set to zero, thus indicating that no scaling has occurred.
 - To display a value in scientific notation, many programming languages, calculators, and spread sheets use the letter e (or E) to separate the significand from the exponent. This is referred to as e-form notation. In this document, e-form notation, along with other forms of conventional numeric notation, is used to represent a value without regard to quantum. To indicate that both value and quantum are being represented, a variation of e-form notation, called q-form notation, is used. In q-form notation, the letter q replaces the letter e and the value shown after q is the right-units-view exponent, that is, the base 10 logarithm of the quantum. Thus, for example, the seven members of the cohort for the value 1e6 in the DFP short format are: 1q6, 10q5, 100q4, 1000q3, 10000q2, 100000q1, and 1000000q0.
 - As an example of the distinction between e-form and q-form notation, consider the representation of the members of the cohort of zero. The DFP short format, for example, can represent 198 exact powers of 10, ranging in value from 1e-101 to 1e96; but can represent only 192 values for a quantum, ranging from 1e-101 to 1e90. Thus, the 192 members of the cohort for zero range from 0q-101 to 0q90.
 - The effect of the DFP preferred quantum control is shown in Figure 9-29.

Rounded Intermediate Value	Result If Inexact	Result If Exact and DPQC = 0	Result If Exact and DPQC = 1
0	0q-101	0q90	0q0
0.125	1250000q-7	125q-3	125q-3
1	1000000q-6	1q0	1q0
10	1000000q-5	1q1	10q0
100	1000000q-4	1q2	100q0
1000	1000000q-3	1q3	1000q0
9999	9999000q-3	9999q0	9999q0
10000	1000000q-2	1q4	10000q0
10001	1000100q-2	10001q0	10001q0
1e5	1000000q-1	1q5	100000q0
1e6	1000000q0	1q6	1000000q0
1e10	1000000q4	1q10	1000000q4
1e15	1000000q9	1q15	1000000q9

Figure 9-29. Effect of DPQC on DFP Short Results

- When the floating-point extension facility is installed, for a PFPO-convert-floating-point-radix operation with DFP result, if DPQC is zero and the DFP result is finite and exact, no quantum exception is recognized. This is because the operation always delivers the DFP result with the preferred quantum in this case.

PFPO Action Figures

Symbol	Meaning
Caa	Alternate-exception-action control
Cho	HFP-overflow control
Chu	HFP-underflow control
Csx	Inexact-suppression control
DQPC	DFP quantum-permission control
Hmax	The largest (in magnitude) representable number in this HFP format with the same sign as the source.
Ho	HFP-overflow condition
Hu	HFP-underflow condition
IMi	IEEE-invalid-operation mask
IMo	IEEE-overflow mask
IMq	Quantum-exception mask
IMu	IEEE-underflow mask
IMx	IEEE-inexact mask
N	No
N'	No (true by virtue of conditions to the left of this column)
PI	Program interruption
Rq	Quantum-exception condition
Rx	Inexact-result condition
SFi	IEEE-invalid-operation flag
SFo	IEEE-overflow flag
SFq	Quantum-exception flag
SFu	IEEE-underflow flag
SFx	IEEE-inexact flag
Xi	IEEE-invalid-operation exception
Xo	IEEE-overflow exception
Xu	IEEE-underflow exception
Y	Yes
Y'	Yes (true by virtue of conditions to the left of this column)
+Hmax	Hmax with a plus sign
Tz	True zero with the same sign as the source.

Figure 9-30. Symbols Used in PFPO Action Figures

Condition	Rx	Csx	IMx	Rq	DQPC	IMq	Action
Normal	N	-	-	N	-	-	complete, cc=0
Normal	N	-	-	Y	1	0	complete, cc=1, SFq=1
Normal	N	-	-	Y	1	1	complete, PI, DXC=00000100
Normal	N	-	-	Y	0	-	complete, cc=0
Normal	Y	0	0	Y'	1	0	complete, cc=1, SFx=1, SFq=1
Normal	Y	0	0	Y'	0	-	complete, cc=1, SFx=1
Normal	Y	1	-	Y'	1	0	complete, cc=1, SFq=1
Normal	Y	1	-	Y'	0	-	complete, cc=0
Normal	Y	0	0	Y'	1	1	complete, PI, SFx=1, DXC=00000100
Normal	Y	1	-	Y'	1	1	complete, PI, DXC=00000100
Normal	Y	0	1	N'	-	-	complete, PI, DXC=00001y00

Figure 9-31. Action for PFPO IEEE Targets (Normal)

Condition	IMu	Rx	Csx	IMx	Rq	DQPC	IMq	Caa	Action
Xu	0	N	-	-	N	-	-	-	complete, cc=0
Xu	0	N	-	-	Y	1	0	-	complete, cc=1, SFq=1
Xu	0	N	-	-	Y	0	-	-	complete, cc=0
Xu	0	N	-	-	Y	1	1	-	complete, PI, DXC=00000100
Xu	0	Y	1	-	Y'	0	-	-	complete, cc=1, SFu=1
Xu	0	Y	0	0	Y'	0	-	-	complete, cc=1, SFu=1, SFx=1
Xu	0	Y	1	-	Y'	1	0	-	complete, cc=1, SFu=1, SFq=1
Xu	0	Y	0	0	Y'	1	0	-	complete, cc=1, SFu=1, SFx=1, SFq=1
Xu	0	Y	0	0	Y'	1	1	-	complete, PI, SFu=1, SFx=1, DXC=00000100
Xu	0	Y	1	-	Y'	1	1	-	complete, PI, SFu=1, DXC=00000100
Xu	0	Y	0	1	N'	-	-	-	complete, PI, SFu=1, DXC=00001y00
Xu	1	N	-	-	N'	-	-	0	Scaled, PI, DXC=00010000
Xu	1	Y	-	-	N'	-	-	0	scaled, PI, DXC=0001xy00
Xu	1	N	-	-	N'	-	-	1	Scaled, cc=2, DXC=00010000
Xu	1	Y	-	-	N'	-	-	1	scaled, cc=2, DXC=0001xy00

Figure 9-32. Action for PFPO IEEE Targets (Underflow)

Condition	IMo	Rx	Csx	IMx	Rq	DQPC	IMq	Caa	Action
Xo	0	Y'	1	-	Y'	0	-	-	complete, cc=1, SFo=1
Xo	0	Y'	0	0	Y'	0	-	-	complete, cc=1, SFo=1, SFx=1
Xo	0	Y'	1	-	Y'	1	0	-	complete, cc=1, SFo=1, SFq=1
Xo	0	Y'	0	0	Y'	1	0	-	complete, cc=1, SFo=1, SFx=1, SFq=1
Xo	0	Y'	0	0	Y	1	1	-	complete, PI, SFo=1, SFx=1, DXC=00000100
Xo	0	Y'	1	-	Y'	1	1	-	complete, PI, SFo=1, DXC=00000100
Xo	0	Y'	0	1	N'	-	-	-	complete, PI, SFo=1, DXC=00001y00
Xo	1	N	-	-	N'	-	-	0	scaled, PI, DXC=00100000
Xo	1	Y	-	-	N'	-	-	0	scaled, PI, DXC=0010xy00
Xo	1	N	-	-	N'	-	-	1	scaled, cc=2, DXC=00100000
Xo	1	Y	-	-	N'	-	-	1	scaled, cc=2, DXC=0010xy00

Figure 9-33. Action for PFPO IEEE Targets (Overflow)

Condition	IMI	Rx	Action
Inf	-	N'	canonical infinity, cc=0
QNaN	-	N'	converted QNaN, cc=0
SNaN	0	N'	converted QNaN, cc=1, SFi=1
SNaN	1	N'	suppress, PI, DXC=10000000

Figure 9-34. Action for PFPO IEEE Targets (Infinity or NaN source)

Rx	Csx	IMx	Action
N	-	-	complete, cc=0
Y	1	-	complete, cc=0
Y	0	0	complete, cc=1, SFx=1
Y	0	1	complete, PI, DXC=00001y00

Figure 9-35. Action for PFPO HFP Targets (Normal)

Condition	Cho	IMi	IMo	Rx	Csx	IMx	Caa	Action
Ho	0	0	-	-	-	-	-	Hmax, cc=1, SFi=1
Ho	0	1	-	-	-	-	-	suppress, PI, DXC=10000000
Ho	1	-	0	Y'	1	-	-	Hmax, cc=1, SFo=1
Ho	1	-	0	Y'	0	0	-	Hmax, cc=1, SFo=1, SFx=1
Ho	1	-	0	Y'	0	1	-	Hmax, PI, SFo=1, DXC=00001y00
Ho	1	-	1	-	-	-	0	scaled, PI, DXC=0010xy00
Ho	1	-	1	-	-	-	1	scaled, cc=2, DXC=0010xy00

Figure 9-36. Action for PFPO HFP Targets (HFP Overflow)

Condition	Chu	IMu	Rx	Csx	IMx	Caa	Action
Hu	0	-	Y'	1	-	-	TZ, cc=0
Hu	0	-	Y'	0	0	-	TZ, cc=1, SFx=1
Hu	0	-	Y'	0	1	-	TZ, PI, DXC=00001y00
Hu	1	0	Y'	1	-	-	TZ, cc=1, SFu=1
Hu	1	0	Y'	0	0	-	TZ, cc=1, SFu=1, SFx=1
Hu	1	0	Y'	0	1	-	TZ, PI, SFu=1, DXC=00001y00
Hu	1	1	-	-	-	0	scaled, PI, DXC=0001xy00
Hu	1	1	-	-	-	1	scaled, cc=2, DXC=0001xy00

Figure 9-37. Action for PFPO HFP Targets (HFP Underflow)

Condition	Chu	Action
Inf	0	Hmax, cc=1, SFi=1
Inf	1	suppress, PI, DXC=10000000
NaN	0	+Hmax, cc=1, SFi=1
NaN	1	suppress, PI, DXC=10000000

Figure 9-38. Action for PFPO HFP Targets (Infinity or NaN Source)

SET BFP ROUNDING MODE

Mnemonic $D_2(B_2)$ [S]

Op Code		B ₂	D ₂
0	16	20	31

Mnemonic Op Code Operands

SRNM 'B299' 2-bit BFP rounding mode
 SRNMB 'B2B8' 3-bit BFP rounding mode

The BFP rounding-mode bits in the FPC register are set from the second-operand address.

or SRNM, the second-operand address is not used to address data; instead, bits 30-31 of the FPC register are set to bits 62 and 63, respectively, of the address. Bits other than 62 and 63 of the second-operand

address are ignored. When the floating-point extension facility is installed, bit 29 of the FPC register is set to zero.

For SRNMB, the second-operand address is not used to address data; instead, bits 29-31 of the FPC register are set to bits 61-63, respectively, of the address. Bits 0-55 of the second-operand address are ignored, bits 56-60 must be zero, and bits 61-63 must designate a valid rounding mode; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 2, BFP instruction
- Operation (SRNMB only if the floating-point extension facility is not installed)
- Specification (SRNMB only)

Programming Note: SRNMB is a complete superset of the functionality of SRNM. It is recommended that newly developed software use the SRNMB instruction exclusively.

SET DFP ROUNDING MODE

SRNMT $D_2(B_2)$ [S]

'B2B9'	B ₂	D ₂	
0	16	20	31

The DFP rounding-mode bits are set from the second-operand address.

The second-operand address is not used to address data; instead, the DFP rounding-mode bits in the FPC register are set with bits 61-63 of the address.

Bits other than 61-63 of the second-operand address are ignored.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

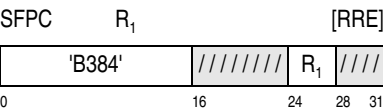
Program Exceptions:

- Data with DXC 3, DFP instruction

- Operation (if the decimal-floating-point-rounding facility is not installed)

2. See also the programming notes under LOAD FPC.

SET FPC



The contents of bit positions 32-63 of of the general register designated by R₁ are placed in the FPC (floating-point-control) register.

All of bits 32-63 corresponding to unsupported bit positions in the FPC register must be zero; otherwise, a specification exception is recognized. For purposes of this checking, a bit position is considered to be unsupported only if it is either unassigned or assigned to a facility which is not installed in any architectural mode of the configuration. Bits 0-31 of the general register are ignored.

When the floating-point extension facility is installed, the bits corresponding to the BFP rounding mode must specify a valid rounding mode; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

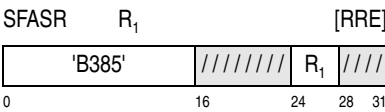
Program Exceptions:

- Data with DXC 2, BFP instruction
- Specification

Programming Notes:

1. When the architectural mode of a CPU is changed without resetting the FPC register (by means of the set architecture signal-processor order, for example) the entire contents of the FPC register are preserved. This is true even when some of these bits are associated with a facility which is not available in both architectural modes. Checking for unsupported bit positions of the FPC register is performed independently of the current architectural mode of the CPU; thus, it is always safe to restore the FPC register from a value previously saved on this CPU, even after the architectural mode has changed.

SET FPC AND SIGNAL



First, bits 0-4 of byte 1 of the floating-point-control (FPC) register at the beginning of the operation are preserved to be used as signaling flags. Next, the contents of the source operand are placed in the FPC register; then the flags in the FPC register are set to the logical OR of the signaling flags and the source flags. Finally, the conditions for simulated-IEEE-exception trap action are examined.

The source operand is in bits 32-63 of the general register designated by R₁; bits 0-31 of the general register are ignored.

If any signaling flag is one and the corresponding source mask is also one, simulated-IEEE-exception trap action occurs. The data-exception code (DXC) in the FPC register is updated to indicate the specific cause of the interruption and a data-exception program interruption occurs at completion of the instruction execution. The DXC for the interruption is shown in Figure 9-21 on page 9-32.

If no signaling flag is enabled, the DXC in the FPC register remains as loaded from the source and instruction execution completes with no trap action.

See Figure 9-22 on page 9-32 for a detailed description of the result of this instruction.

Bits in the source operand that correspond to unsupported bit positions in the FPC register must be zero; otherwise, a specification exception is recognized. For purposes of this checking, a bit position is considered to be unsupported only if it is either unassigned or assigned to a facility which is not installed in any architectural mode of the configuration.

When the floating-point extension facility is installed, the bits corresponding to the BFP rounding mode must specify a valid rounding mode; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for simulated IEEE exception
- Operation (if the IEEE-exception-simulation facility is not installed)
- Specification

Programming Notes:

1. See also the programming notes under LOAD FPC AND SIGNAL.
2. Bits 0-31 of the general register designated by the R_1 field are reserved for future extension and should be set to zeros.

STORE

Mnemonic $R_1, D_2(X_2, B_2)$ [RX-a]

Op Code	R ₁	X ₂	B ₂	D ₂	
0	8	12	16	20	31

Mnemonic	Op Code	Operands
STE	'70'	Short
STD	'60'	Long

Mnemonic2 $R_1, D_2(X_2, B_2)$ [RXY-a]

Op Code	R ₁	X ₂	B ₂	DL ₂	DH ₂	Op Code	
0	8	12	16	20	32	40	47

Mnemonic2	Op Code	Operands
STEY	'ED66'	Short
STDY	'ED67'	Long

The first operand is placed unchanged in storage at the second-operand location.

The displacement for STE and STD is treated as a 12-bit unsigned binary integer. The displacement for STEY and STDY is treated as a 20-bit signed binary integer.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Data with DXC 1, AFP register

- Operation (STEY and STDY, if the long-displacement facility is not installed)

STORE FPC

STFPC	$D_2(B_2)$	[S]	
'B29C'	B_2	D_2	
0	16	20	31

The contents of the FPC (floating-point-control) register are placed in storage at the second-operand location.

The operand is four bytes in length. All 32 bits of the FPC register are stored.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

Program Exceptions:

- Access (store, operand 2)
- Data with DXC 2, BFP instruction

Summary of All Floating-Point Instructions

Figures 9-39 through 9-45 on the following pages show mnemonics for all floating-point instructions (except for PFPO) arranged in various categories of operand format and type of operation. Figure 9-39 shows those instructions which operate on the FPC register. Figure 9-40 on page 9-49 shows the floating-point instructions which have all operands in the same format (same radix and same length), including those FPS (floating-point support) instructions which operate on floating-point operands in a radix-independent manner. Figure 9-41 on page 9-50 and Figure 9-42 on page 9-50 show the floating-point instructions in which all operands are in the same radix but not all of the same length. Figure 9-43 on page 9-50 and Figure 9-44 on page 9-51 show the floating-point instructions in which the source or the result is in a general register. Figure 9-45 on

page 9-51 shows the floating-point instructions other than PFPO which perform radix conversion.

Instruction Name	Mnemonic	Type
EXTRACT FPC	EFPC	BFP
LOAD FPC	LFPC	BFP
LOAD FPC AND SIGNAL	LFAS	DFP
SET BFP ROUNDING MODE	SRNM, SRNMB ¹	BFP
SET DFP ROUNDING MODE	SRNMT	DFP
SET FPC	SFPC	BFP
SET FPC AND SIGNAL	SFASR	DFP
STORE FPC	STFPC	BFP
Explanation: Mnemonics in bold indicate a register-and-storage operation. ¹ The instruction is only available if the floating-point extension facility is installed. BFP The instruction is considered to be part of the BFP facility. An attempt to execute this instruction when the AFP-register-control bit is zero results in a BFP-instruction data exception, DXC 2. DFP The instruction is considered to be part of the DFP facility. An attempt to execute this instruction when the AFP-register-control bit is zero results in a DFP-instruction data exception, DXC 3.		

Figure 9-39. Floating-Point-Control-Register Instructions

Instruction Name	FPS			HFP			BFP			DFP		
	Short	Long	Ext.	Short	Long	Ext.	Short	Long	Ext.	Short	Long	Ext.
ADD (NORMALIZED)				AER AE	ADR AD	AXR	AEBR AEB	ADBR ADB	AXBR		ADTR(A)	AXTR(A)
ADD UNNORMALIZED				AUR AU	AWR AW							
COMPARE				CER CE	CDR CD	CXR	CEBR CEB	CDBR CDB	CXBR		CDTR	CXTR
COMPARE AND SIGNAL							KEBR KEB	KDBR KDB	KXBR		KDTR	KXTR
COMPARE BIASED EXPONENT											CEDTR	CEXTR
COPY SIGN		CPSDR										
DIVIDE				DER DE	DDR DD	DXR	DEBR DEB	DDBR DDB	DXBR		DDTR(A)	DXTR(A)
DIVIDE TO INTEGER							DIEBR	DIDBR				
HALVE				HER	HDR							
LOAD	LER LE LEY	LDR LD LDY	LXR									
LOAD AND TEST				LTER	LTDR	LTXR	LTEBR	LTDBR	LTXBR		LTDR	LTXTR
LOAD COMPLEMENT		LCDFR		LCER	LCDR	LCXR	LCEBR	LCDBR	LCXBR			
LOAD FP INTEGER				FIER	FIDR	FIXR	FIEBR(A)	FIDBR(A)	FIXBR(A)		FIDTR	FIXTR
LOAD NEGATIVE		LNDFR		LNER	LNDR	LNXR	LNEBR	LNDBR	LNGBR			
LOAD POSITIVE		LPDFR		LPER	LPDR	LPXR	LPEBR	LPDBR	LPGBR			
LOAD ZERO	LZER	LZDR	LZXR									
MULTIPLY				MEER MEE	MDR MD	MXR	MEEBR MEEB	MDBR MDB	MXBR		MDTR(A)	MXTR(A)
MULTIPLY AND ADD				MAER MAE	MADR MAD		MAEBR MAEB	MADBR MADB				
MULTIPLY AND ADD UNNORMALIZED					MAYHR MAYH MAYLR MAYL							
MULTIPLY AND SUBTRACT				MSER MSE	MSDR MSD		MSEBR MSEB	MSDBR MSDB				
MULTIPLY UNNORMALIZED					MYHR MYH MYLR MYL							
QUANTIZE											QADTR	QAXTR
REROUND											RRDTR	RRXTR
SHIFT SIGNIFICAND LEFT											SLDT	SLXT
SHIFT SIGNIFICAND RIGHT											SRDT	SRXT
SQUARE ROOT				SQER SQE	SQDR SQD	SQXR	SQEBR SQEB	SQDBR SQDB	SQGBR			
STORE	STE STEY	STD STDY										
SUBTRACT (NORMALIZED)				SER SE	SDR SD	SXR	SEBR SEB	SDBR SDB	SXBR		SXTR(A)	SDTR(A)
SUBTRACT UNNORMALIZED				SUR SU	SWR SW							
TEST DATA CLASS							TCEB	TCDB	TCXB	TDCET	TDCDT	TDCXT
TEST DATA GROUP										TDGET	TDGDT	TDGXT
Explanation: Mnemonics in bold indicate a register-and-storage operation. When the last letter of mnemonic is A, the instruction is an alternate instruction, which uses additional modifier fields not available to the original instruction.												

Figure 9-40. Floating-Point Instructions: All Operands Same Format

Instruction Name	HFP			BFP			DFP	
	Short to Long	Long to Ext.	Short to Ext.	Short to Long	Long to Ext.	Short to Ext.	Short to Long	Long to Ext.
LOAD LENGTHENED	LDER LDE	LXDR LXD	LXER LXE	LDEBR LDEB	LXDBR LXDB	LXEBR LXEB	LDETR	LXDTR
MULTIPLY	MDER MDE	MXDR MXD		MDEBR MDEB	MXDBR MXDB			
MULTIPLY AND ADD UNNORMALIZED		MAYR MAY						
MULTIPLY UNNORMALIZED		MYR MY						
Explanation: Mnemonics in bold indicate a register-and-storage operation.								

Figure 9-41. Floating-Point Instructions: Result Longer than Source

Instruction Name	HFP			BFP			DFP	
	Long to Short	Ext. to Long	Ext. to Short	Long to Short	Ext. to Long	Ext. to Short	Long to Short	Ext. to Long
LOAD ROUNDED	LEDR	LDXR	LEXR	LEDBR(A)	LDXBR(A)	LEXBR(A)	LEDTR	LDXTR

Figure 9-42. Floating-Point Instructions: Result Shorter than Source

Instruction Name	Source GR Size (bits)	Result									
		FPS	HFP				BFP			DFP	
		Long	Short	Long	Ext.	Short	Long	Ext.	Long	Ext.	
CONVERT FROM FIXED	32		CEFR	CDFR	CXFR	CEFBR(A)	CDFBR(A)	CXFBR(A)	CDFTR	CXFTR	
CONVERT FROM FIXED	64		CEGR	CDGR	CXGR	CEGBR(A)	CDGBR(A)	CXGBR(A)	CDGTR(A)	CXGTR(A)	
CONVERT FROM LOGICAL	32					CELFBR	CDLFBR	CXLFBR	CDLFTR	CXLFTR	
CONVERT FROM LOGICAL	64					CELGBR	CDLGBR	CXLGBR	CDLGTR	CXLGTR	
CONVERT FROM SIGNED PACKED	64								CDSTR		
CONVERT FROM SIGNED PACKED	128									CXSTR	
CONVERT FROM UNSIGNED PACKED	64								CDUTR		
CONVERT FROM UNSIGNED PACKED	128									CXUTR	
INSERT BIASED EXPONENT	64								IEDTR	IEXTR	
LOAD FPR FROM GR	64	LDGR									

Figure 9-43. Floating-Point Instructions: General Register Source

Instruction Name	Result GR Size (bits)	Result									
		FPS	HFP				BFP			DFP	
		Long	Short	Long	Ext.	Short	Long	Ext.	Long	Ext.	
CONVERT TO FIXED	32		CFER	CFDR	CFXR	CFEBR(A)	CFDBR(A)	CFXBR(A)	CFDTR	CFXTR	
CONVERT TO FIXED	64		CGER	CGDR	CGXR	CGEBR(A)	CGDBR(A)	CGXBR(A)	CGDTR(A)	CGXTR(A)	
CONVERT TO LOGICAL	32					CLFEBR	CLFDBR	CLFXBR	CLFDTR	CLFXTR	
CONVERT TO LOGICAL	64					CLGEBR	CLGDBR	CLGXBR	CLGDTR	CLGXTR	
CONVERT TO SIGNED PACKED	64								CSDTR		
CONVERT TO SIGNED PACKED	128									CSXTR	
CONVERT TO UNSIGNED PACKED	64								CUDTR		
CONVERT TO UNSIGNED PACKED	128									CUXTR	
EXTRACT BIASED EXPONENT	64								EEDTR	EEXTR	
EXTRACT SIGNIFICANCE	64								ESDTR	ESXTR	
LOAD GR FROM FPR	64	LGDR									

Figure 9-44. Floating-Point Instructions: General Register Result

Instruction Name	Source	Result		
		HFP Long	BFP	
			Short	Long
CONVERT BFP TO HFP	BFP Short	THDER		
CONVERT BFP TO HFP	BFP Long	THDR		
CONVERT HFP TO BFP	HFP Long		TBEDR	TBDR

Figure 9-45. Floating-Point Instructions (other than PFPO): Radix Conversion

Impacts on ESA/390

When the decimal-floating-point facility or the floating-point extension facility is installed in the z/Architecture architectural mode, the behavior of some instructions in the ESA/390 mode is deviated from the definition in the ESA/390 architecture.

The following summarizes the different behavior of affected ESA/390 instructions caused by those two facilities in the z/Architecture architectural mode.

Impacts of the Decimal-Floating-Point Facility

When the decimal-floating-point facility is installed in the z/Architecture architectural mode, it causes the following deviations from the ESA/390 architecture.

- For LOAD FPC (LFPC) and SET FPC (SFPC), if bits 25-27 of the source operand contain a non-

zero value, it is unpredictable whether a specification exception is recognized or the corresponding bits in the FPC register are set to this nonzero value.

Impacts of the Floating-Point Extension Facility

When the floating-point extension facility is installed in the z/Architecture architectural mode, it causes the following deviations from the ESA/390 architecture.

- For LOAD FPC (LFPC) and SET FPC (SFPC), if bit 5 of the source operand is one, it is unpredictable whether a specification exception is recognized or the corresponding bit in the FPC register is set to one; if bits 29-31 of the source operand contain all ones, is unpredictable whether a specification exception is recognized or the corresponding bits in the FPC register are set to all ones.

2. For CONVERT HFP TO BFP (TBEDR, TBDR), if the M_3 field designates the value 3, it is unpredictable whether a specification exception is recognized or an undefined rounding method is performed.
3. For BFP CONVERT TO FIXED (CFEBR, CFDBR, CFXBR), if the M_3 field designates the value 3, it is unpredictable whether a specification exception is recognized or an undefined rounding method is performed. In addition, if bits 20-23 of the instruction contain a nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.
4. For BFP CONVERT FROM FIXED (CEFBR, CDFBR, CXFBR), if bits 16-19 of the instruction contain a nonzero value, it is unpredictable whether a specification exception is recognized or an undefined rounding method is performed. In addition, if bits 20-23 of the instruction contain a nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.
5. For BFP DIVIDE TO INTEGER (DIEBR, DIDBR), if the M_3 field designates the value 3, it is unpredictable whether a specification exception is recognized or an undefined rounding method is performed.
6. For BFP LOAD FP INTEGER (FIEBR, FIDBR, FIXBR), if the M_3 field designates the value 3, it is unpredictable whether a specification exception is recognized or an undefined rounding method is performed. In addition, if bits 20-23 of the instruction contain a nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.
7. For BFP LOAD ROUNDED (LEDBR, LDXBR, LEXBR), if bits 16-19 of the instruction contain a nonzero value, it is unpredictable whether a specification exception is recognized or an undefined rounding method is performed. In addition, if bits 20-23 of the instruction contain a nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.

Chapter 10. Control Instructions

BRANCH AND SET AUTHORITY	10-6	PCKMO-Encrypt-AES-256-Key (Function Code 20)	10-69
BRANCH AND STACK	10-10	PERFORM FRAME MANAGEMENT FUNCTION	10-70
BRANCH IN SUBSPACE GROUP	10-13	PERFORM TIMING FACILITY FUNCTION ..	10-73
COMPARE AND SWAP AND PURGE	10-17	PERFORM TOPOLOGY FUNCTION	10-80
DIAGNOSE	10-19	Operation of Function Codes 0 and 1	10-81
EXTRACT AND SET EXTENDED AUTHORITY	10-20	Operation of Function Code 2:	10-81
EXTRACT PRIMARY ASN	10-20	PROGRAM CALL	10-81
EXTRACT PRIMARY ASN AND INSTANCE	10-20	PROGRAM RETURN	10-94
EXTRACT SECONDARY ASN	10-21	PROGRAM TRANSFER	10-98
EXTRACT SECONDARY ASN AND INSTANCE	10-21	PROGRAM TRANSFER WITH INSTANCE ..	10-98
EXTRACT STACKED REGISTERS	10-21	PURGE ALB	10-107
EXTRACT STACKED STATE	10-23	PURGE TLB	10-107
INSERT ADDRESS SPACE CONTROL ...	10-25	RESET REFERENCE BIT EXTENDED ...	10-107
INSERT PSW KEY	10-26	RESET REFERENCE BITS MULTIPLE ...	10-108
INSERT STORAGE KEY EXTENDED	10-26	RESUME PROGRAM	10-108
INSERT VIRTUAL STORAGE KEY	10-27	SET ADDRESS SPACE CONTROL	10-110
INVALIDATE DAT TABLE ENTRY	10-28	SET ADDRESS SPACE CONTROL FAST ..	10-111
INVALIDATE PAGE TABLE ENTRY	10-32	SET CLOCK	10-112
LOAD ADDRESS SPACE PARAMETERS ..	10-34	SET CLOCK COMPARATOR	10-113
LOAD CONTROL	10-44	SET CLOCK PROGRAMMABLE FIELD ...	10-113
LOAD PAGE TABLE ENTRY ADDRESS ..	10-44	SET CPU TIMER	10-114
LOAD PSW	10-47	SET PREFIX	10-114
LOAD PSW EXTENDED	10-48	SET PSW KEY FROM ADDRESS	10-115
LOAD REAL ADDRESS	10-49	SET SECONDARY ASN	10-115
LOAD USING REAL ADDRESS	10-53	SET SECONDARY ASN WITH INSTANCE	10-115
MODIFY STACKED STATE	10-53	SET STORAGE KEY EXTENDED	10-121
MOVE PAGE	10-54	Setting Storage Keys in Multiple 4K-Byte Blocks	10-123
MOVE TO PRIMARY	10-56	SET SYSTEM MASK	10-124
MOVE TO SECONDARY	10-56	SIGNAL PROCESSOR	10-124
MOVE WITH DESTINATION KEY	10-58	STORE CLOCK COMPARATOR	10-126
MOVE WITH KEY	10-59	STORE CONTROL	10-126
MOVE WITH OPTIONAL SPECIFICATIONS	10-60	STORE CPU ADDRESS	10-126
MOVE WITH SOURCE KEY	10-63	STORE CPU ID	10-127
PAGE IN	10-64	STORE CPU TIMER	10-128
PAGE OUT	10-65	STORE FACILITY LIST	10-128
PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATIONS	10-66	STORE PREFIX	10-128
PCKMO-Query (Function Code 0)	10-67	STORE REAL ADDRESS	10-129
PCKMO-Encrypt-DEA-Key (Function Code 1)	10-68	STORE SYSTEM INFORMATION	10-130
PCKMO-Encrypt-TDEA-128-Key (Function Code 2)	10-68	CPU Topology Overview	10-145
PCKMO-Encrypt-TDEA-192-Key (Function Code 3)	10-68	CPU Slack	10-147
PCKMO-Encrypt-AES-128-Key (Function Code 18)	10-68	STORE THEN AND SYSTEM MASK	10-149
PCKMO-Encrypt-AES-192-Key (Function Code 19)	10-69	STORE THEN OR SYSTEM MASK	10-149
		STORE USING REAL ADDRESS	10-150
		TEST ACCESS	10-150
		TEST BLOCK	10-152
		TEST PROTECTION	10-155
		TRACE	10-157

This chapter includes all privileged and semiprivileged instructions described in this publication, except the input/output instructions, which are described in “I/O Instructions” on page 14-1.

Privileged instructions may be executed only when the CPU is in the supervisor state. An attempt to execute a privileged instruction in the problem state generates a privileged-operation exception.

The semiprivileged instructions are those instructions that can be executed in the problem state when certain authority requirements are met. An attempt to execute a semiprivileged instruction in the problem state when the authority requirements are not met generates a privileged-operation exception or some other program-interruption condition depending on the particular requirement which is violated. Those requirements which cause a privileged-operation exception to be generated in the problem state are not enforced when execution is attempted in the supervisor state.

The control instructions and their mnemonics, formats, and operation codes are listed in Figure 10-1 on page 10-3. The figure also indicates which instructions are new in z/Architecture as compared to ESA/390, when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

The instructions that are new in z/Architecture are indicated in Figure 10-1 by “N.”

When the operands of an instruction are 32-bit operands, the mnemonic for the instruction does not include a letter indicating the operand length. If there is an instruction with the same name but with 64-bit operands, its mnemonic includes the letter “G.” In Figure 10-1, when there is an instruction with 32-bit operands and another instruction with the same name but with “G” added in its mnemonic, the first instruction has “(32)” after its name, and the other instruction has “(64)” after its name.

For those control instructions which have special rules regarding the handling of exceptional situations, a section called “Special Conditions” is included. This section indicates the type of ending

(suppression, nullification, or completion) only for those exceptions for which the ending may vary.

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For LOAD PSW, for example, LPSW is the mnemonic and $D_2(B_2)$ the operand designation.

Programming Notes:

1. The following additional control instructions are available when the DAT-enhancement facility is installed:

- COMPARE AND SWAP AND PURGE (CSPG)
- INVALIDATE DAT TABLE ENTRY

CSPG operates on a doubleword in storage, in contrast to the previously existing instruction COMPARE AND SWAP AND PURGE (CSP), which operates on a word in storage.

2. The long-displacement facility uses new instruction formats, named RSY, RXY, and SIY, to provide 20-bit signed displacements. In connection with the long-displacement facility, all previously existing control instructions of the RSE or RXE format are changed to be of format RSY or RXY, respectively, where the new formats differ from the old by using a previously unused byte, now named DH, in the instructions. When the long-displacement facility is installed, the displacement for an instruction operand address is formed by appending DH on the left of the previous displacement field, now named DL, of the instruction. When the long-displacement facility is not installed, eight zero bits are appended on the left of DL, and DH is ignored.

The following additional control instruction is available when the long-displacement facility is installed:

- LOAD REAL ADDRESS (LRAY)

3. The following additional control instructions are available when the ASN-and-LX-reuse facility is installed:

- EXTRACT PRIMARY ASN AND INSTANCE

- EXTRACT SECONDARY ASN AND INSTANCE
 - PROGRAM TRANSFER WITH INSTANCE
 - SET SECONDARY ASN WITH INSTANCE
4. The LOAD PAGE TABLE ENTRY ADDRESS control instruction is available when the DAT-enhancement facility 2 is installed.
 5. The PERFORM TIMING FACILITY FUNCTION control instruction is available when the TOD-clock-steering facility is installed.
 6. The MOVE WITH OPTIONAL SPECIFICATIONS control instruction is available when the move-with-optional-specifications facility is installed.
 7. The PERFORM TOPOLOGY FUNCTION control instruction is available when the configuration-topology facility is installed.
 8. The PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATIONS control instruction is available when the message-security-assist extension 3 is installed.
 9. The RESET REFERENCE BITS MULTIPLE control instruction is available when the reset-reference-bits-multiple facility is installed.

Name	Mnemonic	Characteristics						Op-code	Page
BRANCH AND SET AUTHORITY	BSA	RRE	Q A ¹	SO T	B			B25A	10-6
BRANCH AND STACK	BAKR	RRE	A ¹	Z ⁵ T	B ST			B240	10-10
BRANCH IN SUBSPACE GROUP	BSG	RRE	A ¹	SO T	B	R ₂		B258	10-13
COMPARE AND SWAP AND PURGE	CSP	RRE C	P A ¹ SP	\$	ST	R ₂		B250	10-17
COMPARE AND SWAP AND PURGE	CSPG	RRE C DE	P A ¹ SP	\$	ST	R ₂		B98A	10-17
DIAGNOSE		DM	P DM			MD		83	10-19
EXTRACT AND SET EXTENDED AUTHORITY	ESEA	RRE N	P					B99D	10-20
EXTRACT PRIMARY ASN	EPAR	RRE	Q	SO				B226	10-20
EXTRACT PRIMARY ASN AND INSTANCE	EPAIR	RRE RA	Q	SO				B99A	10-20
EXTRACT SECONDARY ASN	ESAR	RRE	Q	SO				B227	10-21
EXTRACT SECONDARY ASN AND INSTANCE	ESAIR	RRE RA	Q	SO				B99B	10-21
EXTRACT STACKED REGISTERS (32)	EREG	RRE	A ¹	SE		U ₁ U ₂		B249	10-21
EXTRACT STACKED REGISTERS (64)	EREGG	RRE N	A ¹	SE		U ₁ U ₂		B90E	10-21
EXTRACT STACKED STATE	ESTA	RRE C	A ¹ SP	SE				B24A	10-23
INSERT ADDRESS SPACE CONTROL	IAC	RRE C	Q	SO				B224	10-25
INSERT PSW KEY	IPK	S	Q	G2				B20B	10-26
INSERT STORAGE KEY EXTENDED	ISKE	RRE	P A ¹					B229	10-26
INSERT VIRTUAL STORAGE KEY	IVSK	RRE	Q A ¹	SO		R ₂		B223	10-27
INVALIDATE DAT TABLE ENTRY	IDTE	RRF-b DE	P A ¹ SP	\$				B98E	10-28
INVALIDATE PAGE TABLE ENTRY	IPTE	RRF-a	P A ¹ SP	\$				B221	10-32
LOAD ADDRESS SPACE PARAMETERS	LASP	SSE C	P A ¹ SP	SO		B ₁		E500	10-34
LOAD CONTROL (32)	LCTL	RS-a	P A SP			B ₂		B7	10-44
LOAD CONTROL (64)	LCTLG	RSY-a N	P A SP			B ₂		EB2F	10-44
LOAD PAGE TABLE ENTRY ADDRESS	LPTEA	RRF-b C D2	P A ¹ SP	SO		R ₂		B9AA	10-44
LOAD PSW	LPSW	S L	P A SP	¢		B ₂		82	10-47
LOAD PSW EXTENDED	LPSWE	S L N	P A SP	¢		B ₂		B2B2	10-48
LOAD REAL ADDRESS (32)	LRA	RX-a C	P A ¹	SO		BP		B1	10-49
LOAD REAL ADDRESS (32)	LRAY	RXY-a C LD	P A ¹	SO		BP		E313	10-49
LOAD REAL ADDRESS (64)	LRAG	RXY-a C N	P A ¹			BP		E303	10-49
LOAD USING REAL ADDRESS (32)	LURA	RRE	P A ¹ SP					B24B	10-53
LOAD USING REAL ADDRESS (64)	LURAG	RRE N	P A ¹ SP					B905	10-53
MODIFY STACKED STATE	MSTA	RRE	A ¹ SP	SE	ST			B247	10-53
MOVE PAGE	MVPG	RRE C	Q A SP	G0	ST	R ₁ R ₂		B254	10-54
MOVE TO PRIMARY	MVCP	SS-d C	Q A	SO ¢	ST			DA	10-56
MOVE TO SECONDARY	MVCS	SS-d C	Q A	SO ¢	ST			DB	10-56

Figure 10-1. Summary of Control Instructions (Part 1 of 4)

Name	Mne- monic	Characteristics						Op- code	Page
MOVE WITH DESTINATION KEY	MVCDK	SSE		Q A		GM	ST	B ₁ B ₂	E50F 10-58
MOVE WITH KEY	MVCK	SS-d C		Q A			ST	B ₁ B ₂	D9 10-59
MOVE WITH OPTIONAL SPECIFICATIONS	MVCOS	SSF C MO		Q A	SO	G0	ST	B ₁ B ₂	C80 10-60
MOVE WITH SOURCE KEY	MVCSK	SSE		Q A		GM	ST	B ₁ B ₂	E50E 10-63
PAGE IN	PGIN	RRE C ES	P A ¹			¢			B22E 10-64
PAGE OUT	PGOUT	RRE C ES	P A ¹			¢			B22F 10-65
PERFORM CRYPTOGRAPHIC KEY MGMT. OPERATIONS	PCKMO	RRE	M3	P A SP		GM	ST		B928 10-66
PERFORM FRAME MANAGEMENT FUNCTION	PFMF	RRE	ED	P A ¹ SP	II	¢ ³			B9AF 10-70
PERFORM TIMING FACILITY FUNCTION	PTFF	E C TS	Q A SP			GM	ST		0104 10-73
PERFORM TOPOLOGY FUNCTION	PTF	RRE C CT	P SP						B9A2 10-80
PROGRAM CALL	PC	S		Q A ¹	Z ¹ T	¢ GM	B ST		B218 10-81
PROGRAM RETURN	PR	E L		A ¹ SP	Z ⁴ T	¢ ²	B ST		0101 10-94
PROGRAM TRANSFER	PT	RRE		Q A ¹ SP	Z ² T	¢	B		B228 10-98
PROGRAM TRANSFER WITH INSTANCE	PTI	RRE	RA	Q A ¹ SP	Z ⁶ T	¢	B		B99E 10-98
PURGE ALB	PALB	RRE		P		\$			B248 10-107
PURGE TLB	PTLB	S		P		\$			B20D 10-107
RESET REFERENCE BIT EXTENDED	RRBE	RRE C		P A ¹					B22A 10-107
RESET REFERENCE BITS MULTIPLE	RRBM	RRE	RB	P A ¹					B9AE 10-108
RESUME PROGRAM	RP	S L		Q A SP	WE T		B	B ₂	B277 10-108
SET ADDRESS SPACE CONTROL	SAC	S		Q SP	SW	¢			B219 10-110
SET ADDRESS SPACE CONTROL FAST	SACF	S		Q SP	SW				B279 10-111
SET CLOCK	SCK	S C		P A SP				B ₂	B204 10-112
SET CLOCK COMPARATOR	SCKC	S		P A SP				B ₂	B206 10-113
SET CLOCK PROGRAMMABLE FIELD	SCKPF	E		P SP		G0			0107 10-113
SET CPU TIMER	SPT	S		P A SP				B ₂	B208 10-114
SET PREFIX	SPX	S		P A SP		\$		B ₂	B210 10-114
SET PSW KEY FROM ADDRESS	SPKA	S		Q					B20A 10-115
SET SECONDARY ASN	SSAR	RRE		A ¹	Z ³ T	¢			B225 10-115
SET SECONDARY ASN WITH INSTANCE	SSAIR	RRE	RA	A ¹	Z ⁷ T	¢			B99F 10-115
SET STORAGE KEY EXTENDED	SSKE	RRF-c C		P A ¹	II	¢			B22B 10-121
SET SYSTEM MASK	SSM	S		P A SP	SO			B ₂	80 10-124
SIGNAL PROCESSOR	SIGP	RS-a C		P		\$			AE 10-124
STORE CLOCK COMPARATOR	STCKC	S		P A SP			ST	B ₂	B207 10-126
STORE CONTROL (32)	STCTL	RS-a		P A SP			ST	B ₂	B6 10-126
STORE CONTROL (64)	STCTG	RSY-a N		P A SP			ST	B ₂	EB25 10-126
STORE CPU ADDRESS	STAP	S		P A SP			ST	B ₂	B212 10-126
STORE CPU ID	STIDP	S		P A SP			ST	B ₂	B202 10-127
STORE CPU TIMER	STPT	S		P A SP			ST	B ₂	B209 10-128
STORE FACILITY LIST	STFL	S N3		P					B2B1 10-128
STORE PREFIX	STPX	S		P A SP			ST	B ₂	B211 10-128
STORE REAL ADDRESS	STRAG	SSE N	P A ¹ SP				ST	B ₁ BP	E502 10-129
STORE SYSTEM INFORMATION	STSI	S C	P A SP			GM	ST	B ₂	B27D 10-130
STORE THEN AND SYSTEM MASK	STNSM	SI	P A				ST	B ₁	AC 10-149
STORE THEN OR SYSTEM MASK	STOSM	SI	P A SP				ST	B ₁	AD 10-149
STORE USING REAL ADDRESS (32)	STURA	RRE	P A ¹ SP				SU		B246 10-150
STORE USING REAL ADDRESS (64)	STURG	RRE N	P A ¹ SP				SU		B925 10-150
TEST ACCESS	TAR	RRE C	A ¹					U ₁	B24C 10-150
TEST BLOCK	TB	RRE C	P A ¹	II	\$	G0			B22C 10-152
TEST PROTECTION	TPROT	SSE C	P A ¹					B ₁	E501 10-155
TRACE (32)	TRACE	RS-a	P A SP		T	¢		B ₂	99 10-157

Figure 10-1. Summary of Control Instructions (Part 2 of 4)

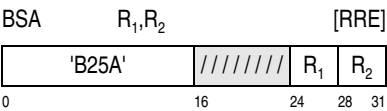
Name	Mne- monic	Characteristics							Op- code	Page		
TRACE (64)	TRACG	RSY-a	N	P	A	SP	T	¢	B ₂	EB0F	10-157	
TRAP	TRAP2	E			A		SO	T	B	ST	01FF	10-159
TRAP	TRAP4	S			A		SO	T	B	ST	B2FF	10-159
Explanation:												
¢	Causes serialization and checkpoint synchronization.											
¢ ²	Causes serialization and checkpoint synchronization when the state entry to be unstacked is a program-call state entry.											
¢ ³	Causes serialization and checkpoint synchronization when the set-key control is one.											
\$	Causes serialization.											
A	Access exceptions for logical addresses.											
A ¹	Access exceptions; not all access exceptions may occur; see instruction description for details.											
B	PER branch event.											
B ₁	B ₁ field designates an access register in the access-register mode.											
B ₂	B ₂ field designates an access register in the access-register mode.											
BP	B ₂ field designates an access register when PSW bits 16 and 17 have the value 01 binary.											
B†	B ₁ field designates an access register when bit 47 of GR0 is zero, and bits 16-17 of the current PSW are 01 binary; or when bit 47 of GR0 is one, and bits 40-41 of GR0 are 01 binary.											
B‡	B ₂ field designates an access register when bit 63 of GR0 is zero, and bits 16-17 of the current PSW are 01 binary; or when bit 63 of GR0 is one, and bits 56-57 of GR0 are 01 binary.											
C	Condition code is set.											
CT	Configuration-topology facility.											
DE	DAT-enhancement facility.											
DM	Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code.											
D2	DAT-enhancement facility 2.											
E	E instruction format.											
ED	Enhanced-DAT facility.											
ES	Expanded-storage facility.											
FC	Designation of access registers depends on the function code of the instruction.											
G0	Instruction execution includes the implied use of general register 0.											
G2	Instruction execution includes the implied use of general register 2.											
GM	Instruction execution includes the implied use of multiple general registers: General registers 0 and 1 for MOVE WITH DESTINATION KEY, MOVE WITH SOURCE KEY, PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATIONS, PERFORM TIMING FACILITY FUNCTION, and STORE SYSTEM INFORMATION. General registers 3, 4, and 14, and, when the ASN-and-LX-reuse facility is installed, general register 15 for PROGRAM CALL.											
II	Interruptible instruction (for SSKE, the instruction is interruptible when the multiple-block control is one).											
L	New condition code is loaded.											
LD	Long-displacement facility.											
MD	Designation of access registers in the access-register mode is model-dependent.											
MO	Move-with-optional-specifications facility.											
M3	Message-security assist extension 3.											
N	Instruction is new in z/Architecture as compared to ESA/390.											
N3	Instruction is new in z/Architecture and has been added to ESA/390.											
P	Privileged-operation exception.											
Q	Privileged-operation exception for semiprivileged instructions.											
R ₁	R ₁ field designates an access register in the access-register mode.											
R ₂	R ₂ field designates an access register in the access-register mode.											

Figure 10-1. Summary of Control Instructions (Part 3 of 4)

	Name	Mne- monic	Characteristics	Op- code	Page
I	RA		Reusable-ASN-and-LX facility.		
	RB		Reset-reference-bits-multiple facility.		
	RRE		RRE instruction format.		
	RS		RS instruction format.		
	RSY		RSY instruction format.		
	RX		RX instruction format.		
	RXY		RXY instruction format.		
	S		S instruction format.		
	SE		Special-operation, stack-empty, stack-specification, and stack-type exceptions.		
	SF		Special-operation, stack-full, and stack-specification exceptions.		
	SI		SI instruction format.		
	SO		Special-operation exception.		
	SP		Specification exception.		
	SS		SS instruction format.		
	SSE		SSE instruction format.		
	SSF		SSF instruction format.		
	ST		PER storage-alteration event.		
	SU		PER store-using-real-address event.		
	SW		Special-operation exception and space-switch event.		
	T		Trace exceptions (which include trace table, addressing, and low-address protection).		
	TS		TOD-clock-steering facility.		
	U ₁		R ₁ field designates an access register unconditionally.		
	U ₂		R ₂ field designates an access register unconditionally.		
	WE		Space-switch event.		
	Z ¹		Additional exceptions and events for PROGRAM CALL (which include ASX-translation, EX-translation, LFX-translation, LSTE-sequence, LSX-translation, LX-translation, PC-translation-specification, special-operation, stack-full, stack-specification and subspace-replacement exceptions and space-switch event).		
	Z ²		Additional exceptions and events for PROGRAM TRANSFER (which include AFX-translation, ASX-translation, primary-authority, special-operation, and subspace-replacement exceptions and space-switch event).		
	Z ³		Additional exceptions for SET SECONDARY ASN (which include AFX translation, ASX translation, secondary authority, special operation and subspace replacement).		
	Z ⁴		Additional exceptions and events for PROGRAM RETURN (which include AFX-translation, ASTE-instance, ASX-translation, secondary-authority, special-operation, stack-empty, stack-operation, stack-specification, stack-type, and subspace-replacement exceptions and space-switch event).		
	Z ⁵		Additional exceptions for BRANCH AND STACK (which include special operation, stack full, and stack specification)		
	Z ⁶		Additional exceptions and events for PROGRAM TRANSFER WITH INSTANCE (which include AFX-translation, ASTE-instance, ASX-translation, primary-authority, special-operation, and subspace-replacement exceptions and space-switch event).		
	Z ⁷		Additional exceptions for SET SECONDARY ASN WITH INSTANCE (which include AFX translation, ASTE instance, ASX translation, secondary authority, special operation, and subspace replacement).		

Figure 10-1. Summary of Control Instructions (Part 4 of 4)

BRANCH AND SET AUTHORITY

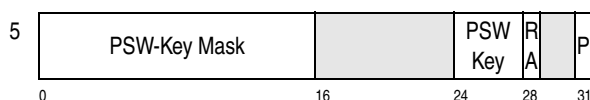


If the dispatchable unit is in the base-authority state and the 24-bit or 31-bit addressing mode: bits 32 and 97-127 of the current PSW, the basic-addressing-mode bit and bits 33-63 of the updated instruction address, are saved in the dispatchable-unit control table (DUCT); the PSW-key mask (PKM), PSW key, and problem-state bit also are saved in the DUCT;

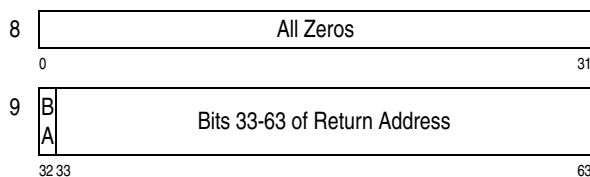
the PKM and PSW key are replaced using the contents of general register R_1 ; the problem-state bit is set to one; bits 32 and 97-127 of the PSW are replaced using the contents of general register R_2 ; and the dispatchable unit is placed in the reduced-authority state. In the 64-bit addressing mode, the action is the same except that bits 64-127 of the current PSW are saved in the DUCT and replaced from general register R_2 , and bit 32 of the PSW is neither saved nor replaced.

If the dispatchable unit is in the reduced-authority state and the 24-bit or 31-bit addressing mode: bits 32 and 97-127 of the current PSW are saved in general register R_1 if R_1 is not zero; bits 32 and 97-127 of the PSW and the PKM, PSW key, and problem-state bit are replaced by values saved in the DUCT; and the dispatchable unit is placed in the base-authority state. In the 64-bit addressing mode, the action is the same except that bits 64-127 of the current PSW are saved in general register R_1 if R_1 is not zero, those bits in the PSW are replaced from the DUCT, and bit 32 of the PSW is neither saved nor replaced.

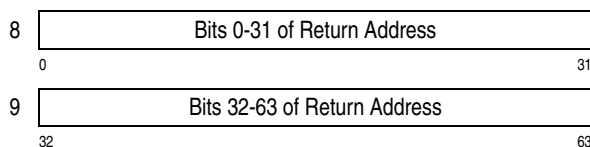
Words 5, 8, and 9 of the DUCT are used by this instruction. The contents of those words are as follows:



In the 24-Bit or 31-Bit Addressing Mode



In the 64-Bit Addressing Mode



The fields in words 5, 8, and 9 of the DUCT are allocated as follows:

PSW-Key Mask: Bit positions 0-15 of word 5 contain the PSW-key mask (PKM), bits 32-47 of control

register 3, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The PKM is restored to control register 3 by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

PSW Key: Bit positions 24-27 of word 5 contain the PSW key, bits 8-11 of the PSW, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The PSW key is restored to the PSW by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

Reduced Authority (RA): Bit 28 of word 5 indicates, when zero, that the dispatchable unit associated with the DUCT is in the base-authority state or, when one, that the dispatchable unit is in the reduced-authority state. Bit 28 is set to one by BRANCH AND SET AUTHORITY executed in the base-authority state, and it is set to zero by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

Problem State (P): Bit position 31 of word 5 contains the problem-state bit, bit 15 of the PSW, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The problem-state bit is restored to the PSW by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

Basic Addressing Mode (BA): In the 24-bit or 31-bit addressing mode, bit position 0 of word 9 contains the basic-addressing-mode bit, bit 32 of the PSW, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The basic-addressing-mode bit is restored to the PSW from the DUCT by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

Return Address: In the 24-bit or 31-bit addressing mode, bit positions 1-31 of word 9 contain bits 33-63 of the updated instruction address, bits 97-127 of the PSW, saved by BRANCH AND SET AUTHORITY executed in the base-authority state. Bits 1-31 of word 9 of the DUCT are restored to bit positions 97-127 of the PSW by BRANCH AND SET AUTHORITY executed in the reduced-authority state. In the 64-bit addressing mode, words 8 and 9 contain the updated instruction address saved by BRANCH AND SET AUTHORITY executed in the base-authority state. The contents of words 8 and 9 are restored to bit positions 64-127 of the PSW by BRANCH AND SET AUTHORITY executed in the reduced-authority state.

In the 24-bit or 31-bit addressing mode, all zeros are stored in word 8 when saving occurs in the base-authority state. In any addressing mode, all zeros are stored in bit positions 16-23, 29, and 30 of word 5 when saving occurs in the base-authority state.

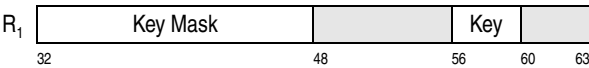
All other fields in words 5, 8, and 9 remain unchanged when bit 28 of word 5 is set to zero in the reduced-authority state.

The fetch, store, and update references to the DUCT are single-access references and appear to be word concurrent as observed by other CPUs. The words of the DUCT are accessed in no particular order.

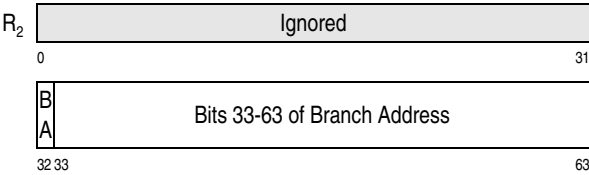
Base-Authority Operation

When BRANCH AND SET AUTHORITY is executed in the base-authority state, as indicated by the reduced-authority bit (RA) in the DUCT being zero, R₂ must be nonzero; otherwise, a special-operation exception is recognized. R₁ may be zero or nonzero.

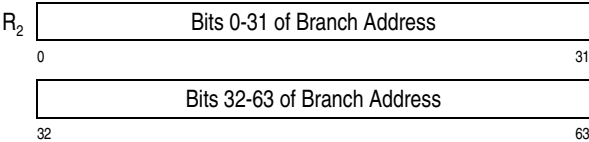
The contents of bit positions 32-63 of general register R₁ and of general register R₂ when the execution of the instruction begins in the base-authority state are as follows:



In the 24-Bit or 31-Bit Addressing Mode



In the 64-Bit Addressing Mode



In any addressing mode, the contents of bit positions 0-31 of general register R₁ are ignored. In the 24-bit or 31-bit addressing mode, the contents of bit positions 0-31 of general register R₂ are ignored.

In the 24-bit or 31-bit addressing mode, PSW bits 32 and 97-127 are saved in word 9 of the DUCT, and

zeros are stored in word 8. In the 64-bit addressing mode, PSW bits 64-127 are saved in words 8 and 9 of the DUCT. In any addressing mode, the PKM, the PSW key, and the problem-state bit are saved in word 5 of the DUCT, the RA bit in word 5 is set to one, and bits 16-23, 29, and 30 of word 5 are set to zeros.

Bits 56-59 of general register R₁ are placed in bit positions 8-11 of the PSW as the new PSW key. In the problem state, the new PSW key must be authorized by the PKM; otherwise, if the new PSW key is not authorized, a privileged-operation exception is recognized.

After the new PSW key has been placed in the PSW, bits 32-47 of general register R₁ are ANDed with the PKM in control register 3, and the result replaces the PKM in control register 3.

The problem-state bit in the PSW is set to one.

In the 24-bit or 31-bit addressing mode, bit 32 of general register R₂ is placed in bit position 32 of the PSW as the new basic-addressing-mode bit. A branch address is generated from bits 33-63 of general register R₂ under the control of the new basic addressing mode, and the result is placed in bit positions 64-127 of the PSW as the new instruction address.

In the 64-bit addressing mode, a branch address is generated from bits 0-63 of general register R₂ and is placed in bit positions 64-127 of the PSW as the new instruction address. Bit 32 of the PSW remains unchanged.

Bits 48-55 and 60-63 of general register R₁ may be used for future extensions and should be zeros; otherwise, the program may not operate compatibly in the future.

Reduced-Authority Operation

When BRANCH AND SET AUTHORITY is executed in the reduced-authority state, as indicated by the reduced-authority (RA) bit in the DUCT being one, R₂ must be zero; otherwise, a special-operation exception is recognized. R₁ may be zero or nonzero. The initial contents of general registers R₁ and R₂ are ignored.

If R₁ is nonzero in the 24-bit or 31-bit addressing mode, bits 32 and 97-127 of the current PSW, the basic-addressing-mode bit and bits 33-63 of the

updated instruction address, are placed in bit positions 32 and 33-63, respectively, of general register R_1 , and bits 0-31 of the register remain unchanged. If R_1 is nonzero in the 64-bit addressing mode, bits 64-127 of the current PSW are placed in bit positions 0-63 of general register R_1 . If R_1 is zero, general register 0 remains unchanged.

In the 24-bit or 31-bit addressing mode, bit 0 of word 9 of the DUCT is placed in PSW bit position 32, and bits 1-31 of word 9, with 33 leftmost zeros appended, are placed in PSW bit positions 64-127.

In the 64-bit addressing mode, the contents of words 8 and 9 of the DUCT are placed in PSW bit positions 64-127, and bit 32 of the PSW remains unchanged.

In any addressing mode, the PKM, the PSW key, and the problem-state bit are restored from the DUCT, and the RA bit is set to zero, as previously described. There is no test for whether the restored PSW key is authorized by the restored PKM.

Special Conditions

R_2 must be nonzero in the base-authority state and zero in the reduced-authority state. If either of these rules is violated, a special-operation exception is recognized, and the operation is suppressed.

In the problem state, the execution of the instruction in the base-authority state is subject to control by the PSW-key mask in control register 3. When the bit in the PSW-key mask corresponding to the PSW-key value to be set is one, the instruction is executed successfully. When the selected bit in the PSW-key mask is zero, a privileged-operation exception is recognized. In the supervisor state, any value for the PSW key is valid.

Key-controlled protection does not apply to any access made during the operation. Low-address protection does apply.

In the 24-bit or 31-bit addressing mode, the contents of word 9 of the DUCT are not checked for validity before they are loaded into the PSW. However, after loading, a specification exception is recognized, and a program interruption occurs, when the newly loaded PSW contains a zero in bit position 32 and the contents of bit positions 97-103 are not all zeros. In this case, the operation is completed, and the resulting instruction-length code is 0. The specification exception, which in this case is listed as a program exception in this instruction, is described in "Early Exception Recognition" on page 6-8. It may be considered as occurring early in the process of preparing to execute the following instruction.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-2 on page 10-10.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (dispatchable-unit control table)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state, base-authority operation only)
- Protection (low-address; dispatchable-unit control table)
- Special operation
- Specification

- Trace

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.	Access exceptions for second instruction halfword.
8.A	Trace exceptions.
8.B	Protection exception (low-address protection) for access to dispatchable-unit control table.
8.C.1	Addressing exception for access to dispatchable-unit control table.
8.C.2	Special-operation exception due to R_2 being zero in the base-authority state or R_2 being nonzero in the reduced-authority state.
8.C.3	Privileged-operation exception due to selected PSW-key-mask bit being zero (base-authority operation only).
9.	Specification exception due to bit 32 of the newly loaded PSW zero when bits 97-103 are not all zeros (reduced-authority operation only).

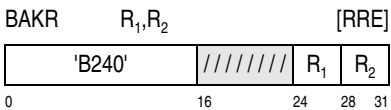
Figure 10-2. Priority of Execution: BRANCH AND SET AUTHORITY

Programming Notes:

1. BRANCH AND SET AUTHORITY can improve performance by replacing to-current-primary forms of PROGRAM TRANSFER (PT-cp) and basic (nonstacking) PROGRAM CALL (PC-cp) instructions. PT-cp and PC-cp are often used (within a single address space) to reduce the authority of the PSW-key mask (PKM) or change from supervisor state to problem state during a calling linkage made by PT-cp and then to restore the PKM authority or supervisor state during a return linkage made by PC-cp. Also, the PSW-key-setting operations of BRANCH AND SET AUTHORITY can be substituted for SET PSW KEY FROM ADDRESS instructions, and, since BRANCH AND SET AUTHORITY combines branching with PSW-key setting, it can be used to change the PSW key when branching from or to a fetch-protected program.
2. Only one base-authority state and one reduced-authority state are available to a dispatchable unit. Nested use of BRANCH AND SET AUTHORITY, that is, use within different subroutine levels, is not possible. The requirement that R_2 must be nonzero in the base-authority state and zero in the reduced-authority state provides detection of an attempt to use BRANCH AND SET AUTHORITY in the base-authority state when the dispatchable unit is already in the reduced-authority state because of a previous use of the instruction in the base-authority state.

3. BRANCH AND SET AUTHORITY in the base authority-state does not save an indication in the DUCT of whether the current addressing mode is the extended (64-bit) addressing mode or a basic (24-bit or 31-bit) addressing mode. The instruction, in either the base-authority state or the reduced-authority state, does not cause a switch between the extended addressing mode and a basic addressing mode. In the reduced-authority state, the contents of words 8 and 9 of the DUCT are interpreted based only on the current addressing mode. If saving occurs in the 31-bit addressing mode and then restoring occurs in the 64-bit addressing mode, bit 0 of word 9 of the DUCT will be used as an address bit instead of as the basic-addressing-mode bit. If saving occurs in the 64-bit addressing mode and then restoring occurs in the 24-bit addressing mode, an early specification exception may be recognized, after the instruction execution is completed, because bits 97-103 of the PSW may be nonzero when bit 32 is zero.
4. The instruction may be referred to as BSA-ba or BSA-ra depending on whether it is executed in the base-authority state or the reduced-authority state, respectively.

BRANCH AND STACK



A linkage-stack branch state entry is formed, and the current PSW, except with an unpredictable PER mask and with the addressing-mode bits and instruction address replaced from the first operand, is placed in the state entry. Subsequently, the updated instruction address in the current PSW is replaced from the second operand. Indications of the current addressing-mode bits and the new instruction address are placed in the state entry, and the PSW-key mask, PASN, SASN, EAX, and contents of general registers 0-15 and access registers 0-15 also are placed in the state entry. When the ASN-and-LX-reuse facility is installed and the ASN-and-LX-reuse control in control register 0 is one, the PASTEIN and SASTEIN also are placed in the state entry. The action associated with an operand is not performed if the R field designating the operand is zero.

When the R_1 field is nonzero, the contents of general register R_1 specify an address referred to as the return address.

When R_1 is nonzero and bit 63 of general register R_1 is zero, the return address is generated from the contents of the register under the control of the basic addressing mode specified by bit 32 of the register: 24-bit mode if bit 32 is zero, or 31-bit mode if bit 32 is one. Bit 32 of the register and the return address are substituted for the basic-addressing-mode bit, bit 32, and the updated instruction address, respectively, in the current PSW when the contents of that PSW are placed in the state entry. The extended-addressing-mode bit, bit 31, is set to zero in the PSW that is placed in the state entry. The contents of the current PSW are not changed.

When R_1 is nonzero and bit 63 of general register R_1 is one, the return address is generated from the contents of the register under the control of the 64-bit addressing mode. Bits 0-62 of the return address, with a zero appended on the right, are substituted for the updated instruction address in the current PSW when the contents of that PSW are placed in the state entry. Bits 31 and 32 are set to one in the PSW that is placed in the state entry. The contents of the current PSW are not changed.

When the R_1 field is zero, the current PSW is placed in the state entry without any change except for an unpredictable PER mask.

Subsequently, when the R_2 field is nonzero, the instruction address in the current PSW is replaced by the branch address. The branch address is generated from the contents of general register R_2 under the control of the current addressing mode. When the R_2 field is zero, the operation is performed without branching.

The branch state entry is formed and information is placed in it as described in "Stacking Process" on page 5-80.

In the 24-bit or 31-bit addressing mode, bits 33-63 of the branch address (or of the updated instruction address if the operation is performed without branching) are placed in bit positions 33-63 of bytes 144-151 in the state entry, bit 32 of the current PSW is placed in bit position 32 of those bytes, and zeros are placed in bit positions 0-31 of the bytes.

In the 64-bit addressing mode, bits 0-62 of the branch address (or of the updated instruction address if the operation is performed without branching) are placed in bit positions 0-62 of bytes 144-151 in the state entry, and a one is placed in bit position 63 of those bytes.

The entry-type code in the state entry is 0001100 binary.

Key-controlled protection does not apply to accesses to the linkage stack, but low-address and DAT protection do apply.

Special Conditions

The CPU must be in the primary-space mode or access-register mode; otherwise, a special-operation exception is recognized.

A stack-full or stack-specification exception may be recognized during the stacking process.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-3 on page 10-12.

Condition Code: The code remains unchanged.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off or the CPU being in secondary-space mode or home-space mode.
8.A	Trace exceptions (only if R_2 is nonzero).
8.B.1	Access exceptions (fetch) for entry descriptor of the current linkage-stack entry. Note: Exceptions 8.B.2-8.B.7 can occur only if there is not enough remaining free space in the current linkage-stack section.
8.B.2	Stack-specification exception due to remaining-free-space value in current linkage-stack entry not being a multiple of 8.
8.B.3	Access exceptions (fetch) for second word of the trailer entry of the current section. The entry is presumed to be a trailer entry; its entry-type field is not examined.
8.B.4	Stack-full exception due to forward-section validity bit in the trailer entry being zero.
8.B.5	Access exceptions (fetch) for entry descriptor of the header entry of the next section. This entry is presumed to be a header entry; its entry-type field is not examined.
8.B.6	Stack-specification exception due to not enough remaining free space in the next section.
8.B.7	Access exceptions (store) for second word of the header entry of the next section. If there is no exception, the header is now called the current entry.
8.B.8	Access exceptions (store) for entry descriptor of the current entry and for the new state entry.

Figure 10-3. Priority of Execution: *BRANCH AND STACK*

Program Exceptions:

- Access (fetch or store, except for key-controlled protection, linkage-stack entry)
- Special operation
- Stack full
- Stack specification
- Trace

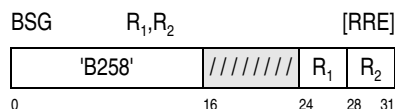
Programming Notes:

1. Examples of the use of the *BRANCH AND STACK* instruction are given in Appendix A, "Number Representation and Instruction-Use Examples."
2. In no case does *BRANCH AND STACK* change the current addressing mode.
3. The effect when the R_1 field is zero is that the return address, which would otherwise be specified by the R_1 general register, is the address of the next sequential instruction. In this case, *BRANCH AND STACK* provides a program-link-

age function that is comparable to the function of *BRANCH AND SAVE*.

4. *BRANCH AND STACK* with a nonzero R_1 field is intended for use at or near the entry point of a called program. The program may be called by means of *BRANCH AND LINK* (BALR) or *BRANCH AND SAVE* (BAS or BASR) from a program being executed in the 24-bit or 31-bit addressing mode, by means of *BRANCH AND SAVE AND SET MODE* from a program being executed in any addressing mode, or by means of a *BRANCH AND SET MODE* instruction located in a "glue module" and being executed in any addressing mode. In all of these cases when the nonzero R_1 field of the calling instruction is the same as the R_1 field of *BRANCH AND STACK*, and even when the addressing mode was changed during the calling linkage, *BRANCH AND STACK* correctly saves the addressing mode and return address of the calling program so that the subsequent execution of *PROGRAM RETURN* will return correctly to the calling program.

BRANCH IN SUBSPACE GROUP



Provided that the current primary address space is in the subspace group, if any, associated with the current dispatchable unit, the access-list-entry token (ALET) in access register R_2 is translated by means of a special form of access-register translation (ART) to locate a destination ASN-second-table entry (DASTE). If the DASTE specifies the base space of the subspace group, the primary ASCE (PASCE) in control register 1 is replaced by the ASCE in the DASTE. If the DASTE specifies a subspace of the group, bits 0-55 and 58-63 of the PASCE are replaced by the same bits of the ASCE in the DASTE. In either case, the following actions also occur.

In the 24-bit or 31-bit addressing mode, bits 32 and 97-127 of the current PSW, the basic-addressing-mode bit and bits 33-63 of the updated instruction address, are saved in bit positions 32 and 33-63, respectively, of general register R_1 , and bits 0-31 of the register remain unchanged. Subsequently, the basic-addressing-mode bit and bits 33-63 of the instruction address in the current PSW are replaced from bit positions 32-63 of general register R_2 , and bits 0-31 of the register are ignored.

In the 64-bit addressing mode, bits 64-127 of the current PSW, the updated instruction address, are saved in bit positions 0-63 of general register R_1 . Subsequently, the instruction address in the current PSW is replaced from bit positions 0-63 of general register R_2 . Bit 32 of the PSW remains unchanged.

In any addressing mode, general register 0 remains unchanged if the R_1 field is zero.

The secondary ASCE (SASCE) in control register 7 is set equal to the new PASCE in control register 1. Also, the secondary ASN (SASN), bits 48-63 of control register 3, is set equal to the primary ASN (PASN), bits 48-63 of control register 4. If the ASN-and-LX-reuse facility is installed and is enabled by the ASN-and-LX-reuse control in control register 0, the secondary ASTEIN (SASTEIN), bits 0-31 of control register 3, is set equal to the primary ASTEIN (PASTEIN), bits 0-31 of control register 4

The current primary address space is in the subspace group for the dispatchable unit if the current primary-ASTE origin (PASTE0), bits 33-57 of control register 5, designates the ASTE for the base space of the group. The PASTE0 designates the base-space ASTE if the PASTE0 is equal to the base-ASTE origin (BASTE0), bits 1-25 of word 0 of the dispatchable-unit control table (DUCT). For determining whether the PASTE0 equals the BASTE0, either the PASTE0 may be compared to the BASTE0 or the entire contents of bit positions 32-63 of control register 5 may be compared to the entire contents of word 0 of the DUCT.

Ordinary ART is described in "Access-Register-Translation Process" on page 5-55. The special ART performed by this instruction is contrasted to ordinary ART as follows:

1. The special ART is performed regardless of whether the CPU is in the access-register mode.
2. If the ALET being translated is 00000000 hex, called ALET 0, the DASTE is the ASTE for the base space. Bit 0 of the DASTE is ignored.
3. If the ALET is 00000001 hex, called ALET 1, the DASTE is the ASTE for the last subspace entered by the dispatchable unit by means of BRANCH IN SUBSPACE GROUP. That ASTE is designated by the subspace-ASTE origin (SSASTE0), bits 1-25 of word 1 of the DUCT. A special-operation exception is recognized if a subspace has not previously been entered, as indicated by an SSASTE0 of all zeros. An ASTE-validity exception is recognized if bit 0 of the DASTE is one. An ASTE-sequence exception is recognized if the ASTE sequence number (ASTESN) in the DASTE does not equal the subspace ASTESN (SSASTESN) in word 3 of the DUCT. The DASTE located because of ALET 1 is considered to specify a subspace even if, due to an error, the DASTE is the ASTE for the base space. That is, there is no comparison of the SSASTE0 to the BASTE0.
4. If the ALET is other than ALET 0 and ALET 1, an ASTE is located by obtaining its origin from an access-list entry (ALE) in a way similar to ordinary ART, and the DASTE is that located ASTE. In this case, as in ordinary ART:
 - An ALET-specification exception is recognized if bits 0-6 of the ALET are not zeros.

- An ALEN-translation exception is recognized if the ALE is outside the effective access list or bit 0 of the ALE is one.
- An ASTE-validity exception is recognized if bit 0 of the DASTE is one.
- An ASTE-sequence exception is recognized if the ASTE sequence number (ASTESN) in the DASTE does not equal the ASTESN in the ALE.

The operation differs from ordinary ART in that the ALE sequence number (ALESN) in the ALE is not compared to the ALESN in the ALET, and the private bit in the ALE is treated as zero. Thus, ALE-sequence and extended-authority exceptions cannot occur.

The fetch-only bit in the ALE is ignored.

When the ALET is other than ALET 0 and ALET 1, the special ART may be performed by using the ART-lookaside buffer (ALB).

The DASTE located due to an ALET other than ALET 0 and ALET 1 may be the ASTE for the base space of the subspace group associated with the dispatchable unit. The DASTE is the base-space ASTE if the DASTE origin (DASTEO) obtained from an ALE by ART equals the BASTEO in the DUCT. For determining whether the DASTEO equals the BASTEO, either the DASTEO may be compared to the BASTEO, or the DASTEO with one leftmost and six rightmost zeros appended may be compared to the entire contents of word 0 of the DUCT. If the DASTE is not the base-space ASTE, the DASTE is treated as the ASTE for a subspace of the dispatchable unit's subspace group provided that (1) the subspace-group bit, bit 54, in the ASCE in the DASTE is one, and (2) the DASTE does not specify the base space of another subspace group. The DASTE specifies the base space of another subspace group if the base-space bit, bit 31 of word 0 of the DASTE, is one. A special-operation exception is recognized if either of those two provisions is not met.

If the DASTE specifies the base space of the subspace group, the PASCE in control register 1 is replaced by the ASCE in the DASTE. If the DASTE specifies a subspace, bits 0-55 and 58-63 of the PASCE are replaced by the same bits of the ASCE in the DASTE, and bits 56 and 57 of the PASCE, the storage-alteration-event bit and space-switch-event-control bit, remain unchanged.

If R_1 is nonzero in the 24-bit or 31-bit addressing mode, bits 32 and 97-127 of the current PSW, the basic-addressing-mode bit and bits 33-63 of the updated instruction address, are placed in bit positions 32 and 33-63, respectively, of general register R_1 , and bits 0-31 of the register remain unchanged. If R_1 is nonzero in the 64-bit addressing mode, bits 64-127 of the current PSW, the updated instruction address, are placed in bit positions 0-63 of general register R_1 . If R_1 is zero, general register 0 remains unchanged.

Whether R_2 is nonzero or zero, in the 24-bit or 31-bit addressing mode, bits 32-63 of general register R_2 specify the new basic addressing mode and designate the branch address. Bit 32 of the register specifies the new basic addressing mode and replaces bit 32 of the current PSW, and the branch address is generated from the contents of bit positions 33-63 of the register under the control of the new basic addressing mode.

When R_2 is nonzero or zero in the 64-bit addressing mode, the contents of general register R_2 designate the branch address. The branch address is generated from the contents of the register under the control of the 64-bit addressing mode. Bit 32 of the PSW remains unchanged.

Regardless of the addressing mode, the new value for the PSW is computed before general register R_1 is changed.

The secondary ASCE (SASCE) in control register 7 is set equal to the new PASCE in control register 1. The secondary ASN (SASN), bits 48-63 of control register 3, is set equal to the primary ASN (PASN), bits 48-63 of control register 4. If the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, the secondary ASTEIN (SASTEIN), bits 0-31 of control register 3, is set equal to the primary ASTEIN (PASTEIN), bits 0-31 of control register 4.

If the DASTE specifies the base space, the subspace-active bit, bit 0 of word 1 of the DUCT, is set to zero, and bits 1-31 of word 1 remain unchanged. If the DASTE specifies a subspace by means of ALET 1, then (1) the subspace-active bit is set to one, (2) the SSASTEO in bit positions 1-25 of word 1 remains unchanged, and (3) bits 26-31 of word 1 either are set to zeros or remain unchanged. If the DASTE specifies a subspace by means of an ALET

other than ALET 1, then (1) the subspace-active bit is set to one, (2) the DASTEO is stored in bit positions 1-25 of word 1 as the SSASTEO, (3) zeros are stored in bit positions 26-31 of word 1, and (4) the ASTESN in the DASTE is stored in word 3 of the DUCT as the SSASTESN.

The fetch, store, and update references to the DUCT are single-access references and appear to be word concurrent as observed by other CPUs. The words of the DUCT are accessed in no particular order.

The operation, since it changes a translation parameter in control register 1, causes all copies of prefetched instructions to be discarded, except when in the home-space mode.

Special Conditions

DAT must be on; otherwise, a special-operation exception is recognized. A special-operation exception is also recognized if the current primary address space is not in a subspace group associated with the current dispatchable unit, if the ALET in access register R_2 is ALET 1 but a subspace has not previously been entered by the dispatchable unit by means of BRANCH IN SUBSPACE GROUP, or if the ALET used is other than ALET 0 and ALET 1 and the destination ASTE does not specify the base space or a subspace of the subspace group.

The primary space-switch-event-control bit, bit 57 of control register 1 either before or after the operation, does not cause a space-switch-event program interruption to occur.

Key-controlled protection does not apply to any access made during the operation. Low-address protection does apply.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: BRANCH IN SUBSPACE GROUP."

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (dispatchable-unit control table, effective access-list designation, access-list entry, destination ASN-second-table entry)

- ALET specification
- ALEN translation
- ASTE sequence
- ASTE validity
- Protection (low-address; dispatchable-unit control table)
- Special operation
- Trace

Programming Notes:

1. See the discussion of BRANCH IN SUBSPACE GROUP in "Subroutine Linkage without the Linkage Stack" on page 5-11. It is intended that there be a separate ASN-second-table entry (ASTE) for each of the base space and each subspace of a subspace group. The ASTEs for the subspaces can be "pseudo" ASTEs as described in the programming note in "Address-Space Number" on page 3-20. A subspace can contain a subset of the storage in the base space by having the DAT tables for the subspace designate a subset of the pages that are designated by the DAT tables for the base space. A dispatchable unit has access to a subspace if an access-list entry designating the ASTE for the subspace is in the primary-space or dispatchable-unit access list of the dispatchable unit.
2. BRANCH IN SUBSPACE GROUP can be used to give control from the base space to a subspace, from a subspace to another subspace, and from a subspace to the base space. The instruction can also be used to give control from the base space to the base space or from a subspace to the same subspace.
3. Since BRANCH IN SUBSPACE GROUP sets the secondary address-space-control element (ASCE) in control register 7 equal to the new primary ASCE in control register 1 (along with setting the secondary ASN in control register 3 equal to the primary ASN in control register 4), the program in an address space given control by BRANCH IN SUBSPACE GROUP does not have access to the calling program's address space by means of that address space being the secondary address space.
4. When a dispatchable unit has used BRANCH IN SUBSPACE GROUP to enter a subspace and has not subsequently used BRANCH IN SUBSPACE GROUP to return to the base space, the dispatchable unit is said to be "subspace active." When LOAD ADDRESS SPACE PARAMETERS,

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off.
8.A	Trace exceptions.
8.B	Protection exception (low-address protection) for access to dispatchable-unit control table.
8.C.1	Addressing exception for access to dispatchable-unit control table.
8.C.2	Special-operation exception due to current primary address space not being in a subspace group associated with the current dispatchable unit (primary-ASTE origin in control register 5 not equal to base-ASTE origin in dispatchable-unit control table). Note: Exception 8.C.3.A can occur only if the access-list-entry token (ALET) in access register R_2 is ALET 0.
8.C.3.A	Addressing exception for access to base ASTE (ASTE designated by base-ASTE origin in dispatchable-unit control table). Note: Exceptions 8.C.3.B.1-8.C.3.B.4 can occur only if the access-list-entry token (ALET) in access register R_2 is ALET 1.
8.C.3.B.1	Special-operation exception due to subspace-ASTE origin in dispatchable-unit control table being zero.
8.C.3.B.2	Addressing exception for access to subspace ASTE.
8.C.3.B.3	ASTE-validity exception due to bit 0 in subspace ASTE being one.
8.C.3.B.4	ASTE-sequence exception due to ASTE sequence number in subspace ASTE not being equal to subspace-ASTE sequence number in dispatchable-unit control table. Note: Exceptions 8.C.3.C.1-8.C.3.C.9 can occur only if the access-list-entry token (ALET) in access register R_2 is other than ALET 0 and ALET 1.
8.C.3.C.1	ALET-specification exception due to bits 0-6 of ALET not being all zeros.
8.C.3.C.2	Addressing exception for access to effective access-list designation.
8.C.3.C.3	ALLEN-translation exception due to access-list entry being outside the list.
8.C.3.C.4	Addressing exception for access to access-list entry.
8.C.3.C.5	ALLEN-translation exception due to I bit in access-list entry being one.
8.C.3.C.6	Addressing exception for access to destination ASTE.
8.C.3.C.7	ASTE-validity exception due to bit 0 in destination ASTE being one.
8.C.3.C.8	ASTE-sequence exception due to ASTE sequence number (ASTESN) in access-list entry not being equal to ASTESN in destination ASTE.
8.C.3.C.9	Special-operation exception due to destination-ASTE origin not equal to base-ASTE origin in dispatchable-unit control table and (1) subspace-group bit, bit 54 in address-space-control element in destination ASTE being zero or (2) base-space bit 31, in destination ASTE being one.

Figure 10-4. Priority of Execution: *BRANCH IN SUBSPACE GROUP*

PROGRAM CALL, PROGRAM RETURN, PROGRAM TRANSFER, PROGRAM TRANSFER WITH INSTANCE, SET SECONDARY ASN, or SET SECONDARY ASN WITH INSTANCE places an ASCE in control register 1 as the pri-

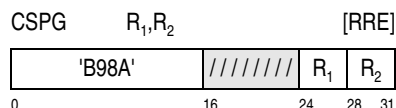
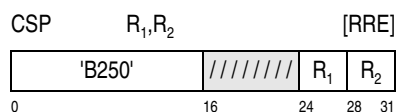
mary ASCE or in control register 7 as the secondary ASCE, and if (1) the ASCE has the subspace-group bit on in it, (2) the dispatchable unit is subspace active, and (3) the ASCE was obtained from the ASN-second-table entry

(ASTE) for the base space of the current dispatchable unit, then the instruction (any of the seven named instructions) replaces bits 0-55 and 58-63 of the ASCE in the control register with the same bits of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. Further details about the effects of the subspace-group facility on the seven named instructions are given in “Subspace-Replacement Operations” on page 5-65 and in the definitions of the instructions.

5. The use of BRANCH IN SUBSPACE GROUP (BSG) along with PROGRAM CALL (PC) and either PROGRAM TRANSFER (PT) (or PROGRAM TRANSFER WITH INSTANCE) or PROGRAM RETURN (PR) can produce results that may be unexpected. Consider the following sequence of operations:
 - a. Start in the base space
 - b. BSG to a subspace
 - c. PC (the first PC) to an address space that is not in the subspace group.
 - d. PC (the second PC) to the base space. Since the dispatchable unit is subspace active, control is given to the subspace.
 - e. BSG back to the base space.
 - f. PT or PR (paired with the second PC) back to the address space that is not in the subspace group.
 - g. PT or PR (paired with the first PC) back to the subspace group. Since the dispatchable unit is no longer subspace active, control is given to the base space even though the first PC was issued in the subspace.
6. BRANCH IN SUBSPACE GROUP does not perform the serialization or checkpoint-synchronization functions, but it does cause all copies of prefetched instructions to be discarded except when in the home-space mode.
7. Unlike the RR-format branch instructions, a value of zero in the R_2 field for BRANCH IN SUBSPACE GROUP designates general register 0, and branching occurs.
8. When the R_2 field designates access register 0, the access register is treated as containing ALET

0 regardless of the contents of the access register.

COMPARE AND SWAP AND PURGE



The first and second operands are compared. If they are equal, contents of general register $R_1 + 1$ are stored at the second-operand location, and a purging operation is performed. If they are unequal, the second operand is loaded at the first-operand location. The result of the comparison is indicated in the condition code.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

For COMPARE AND SWAP AND PURGE (CSP), the first operand is the contents of bit positions 32-63 of general register R_1 . The second operand is a word in storage.

For COMPARE AND SWAP AND PURGE (CSPG), the first operand is the contents of bit positions 0-63 of general register R_1 . The second operand is a doubleword in storage.

For both CSP and CSPG, the location of the leftmost byte of the second operand is designated by contents of general register R_2 .

The purging operation applies to ART-lookaside buffers (ALBs) and translation-lookaside buffers (TLBs) in all CPUs in the configuration. Either ALBs or TLBs, or both ALBs and TLBs, may be selected for purging. All entries are cleared from the selected buffers.

The purging operation is specified by means of bits 62 and 63 of general register R_2 . When bit 62 is one, entries are cleared from ALBs. When bit 63 is one, entries are cleared from TLBs. When bits 62 and 63

both are ones, entries are cleared from ALBs and TLBs. When bits 62 and 63 both are zeros, no entries are cleared.

The handling of the address in general register R_2 is dependent on the addressing mode. For CSP in the 24-bit addressing mode, the contents of bit positions 40-61 of general register R_2 , with two zeros appended on the right, constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-61 of the register, with two zeros appended on the right, constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-61 of the register, with two zeros appended on the right, constitute the address.

For CSPG in the 24-bit addressing mode, the contents of bit positions 40-60 of general register R_2 , with three zeros appended on the right, constitute the

address, and the contents of bit positions 0-39 and 61 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-60 of the register, with three zeros appended on the right, constitute the address, and the contents of bit positions 0-32 and 61 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-60 of the register, with three zeros appended on the right, constitute the address, and the contents of bit position 61 are ignored.

The contents of the registers just described are shown in Figure 10-5 on page 10-18 and Figure 10-6 on page 10-19. When an equal comparison occurs, the contents of bit positions 32-63 of general register $R_1 + 1$ for CSP, or of bit positions 0-63 for CSPG, are stored at the second-operand location. The fetch of the second operand for purposes of comparison and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs.

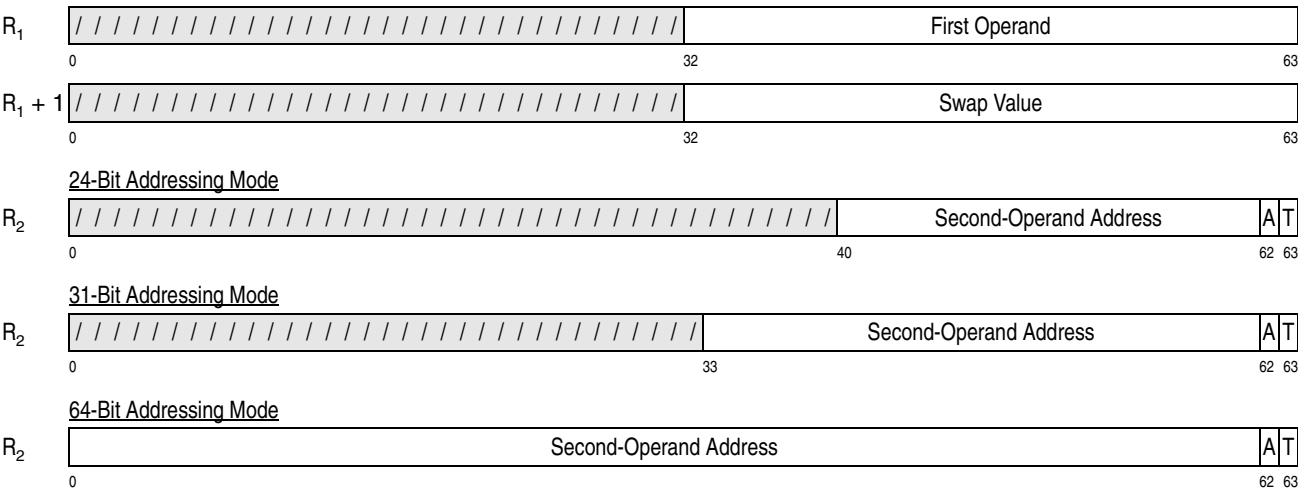


Figure 10-5. Register Contents for COMPARE AND SWAP AND PURGE (CSP)

When the result of the comparison is unequal, the second-operand is loaded at the first-operand location, bits 0-31 of general register R_1 remain unchanged for CSP only, and the second-operand location remains unchanged. However, on some models, the second operand may be fetched and subsequently stored back unchanged at the second-operand location. This update appears to be a block-concurrent interlocked-update reference as observed by other CPUs.

A serialization function is performed before the operand is fetched and again after the operation is completed.

When an equal comparison occurs, this CPU clears entries from its ALB and TLB, as specified by bits 62 and 63 of general register R_2 , and signals all CPUs in the configuration to clear the same specified entries from their ALBs and TLBs. The ALB entries that are cleared are all ALB access-list designations, access-list entries, ASN-second-table entries, and authority-table entries. The TLB entries that are cleared are all

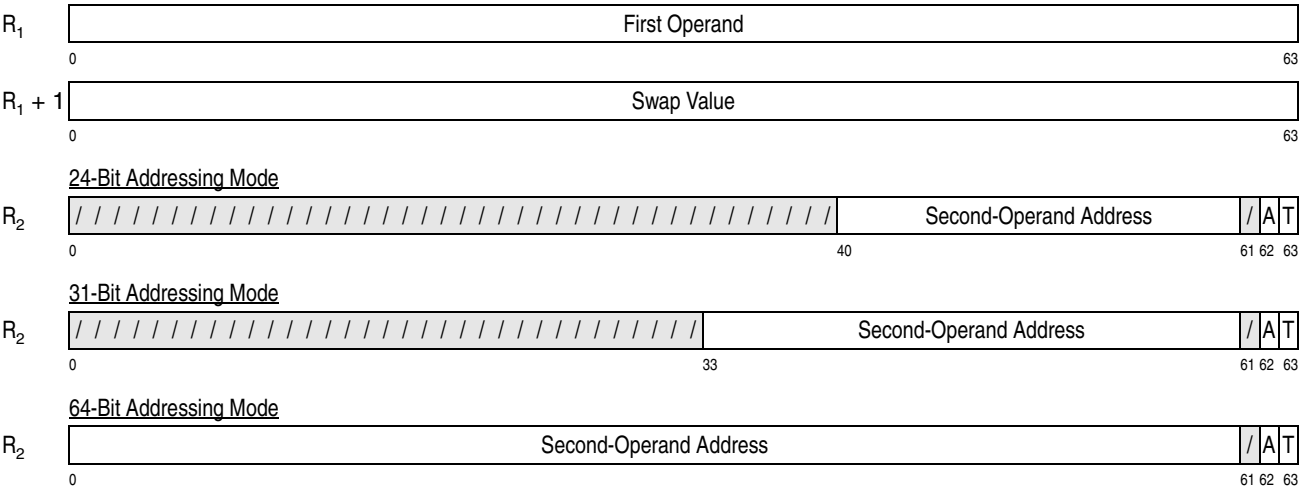


Figure 10-6. Register Contents for COMPARE AND SWAP AND PURGE (CSPG)

combined region-and-segment-table entries, page-table entries, and real-space entries.

The execution of COMPARE AND SWAP AND PURGE is not completed on the CPU which executes it until (1) all specified entries have been cleared from the ALB and TLB of this CPU and (2) all other CPUs in the configuration have completed any storage accesses, including the updating of the change and reference bits, by using the specified ALB and TLB entries.

Special Conditions

The R₁ field must designate an even register; otherwise, a specification exception is recognized.

Resulting Condition Code:

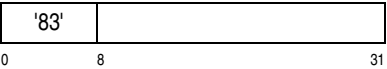
- 0 First and second operands equal, second operand replaced by contents of general register R₁ + 1
- 1 First and second operands unequal, first operand replaced by second operand
- 2 --
- 3 --

Program Exceptions:

- Access (fetch and store, operand 2)
- Operation (if DAT-enhancement facility is not installed, CSPG only)
- Privileged operation
- Specification

Programming Note: COMPARE AND SWAP AND PURGE provides a broadcast form of the PURGE ALB and PURGE TLB instructions, thus making it possible to avoid uses of SIGNAL PROCESSOR.

DIAGNOSE



The CPU performs built-in diagnostic functions, or other model-dependent functions. The purpose of the diagnostic functions is to verify proper functioning of equipment and to locate faulty components. Other model-dependent functions may include disabling of failing buffers, reconfiguration of CPUs, storage, and channel paths, and modification of control storage.

Bits 8-31 may be used as in the SI or RS formats, or in some other way, to specify the particular diagnostic function. The use depends on the model.

The execution of the instruction may affect the state of the CPU and the contents of a register or storage location, as well as the progress of an I/O operation. Some diagnostic functions may cause the test indicator to be turned on.

Resulting Condition Code: The code is unpredictable.

Program Exceptions:

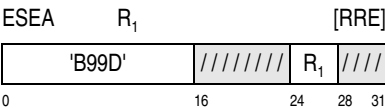
- Privileged operation

- Depending on the model, other exceptions may be recognized.

Programming Notes:

1. Since the instruction is not intended for problem-state-program or control-program use, DIAGNOSE has no mnemonic.
2. DIAGNOSE, unlike other instructions, does not follow the rule that programming errors are distinguished from equipment errors. Improper use of DIAGNOSE may result in false machine-check indications or may cause actual machine malfunctions to be ignored. It may also alter other aspects of system operation, including instruction execution and channel-program operation, to an extent that the operation does not comply with that specified in this publication. As a result of the improper use of DIAGNOSE, the system may be left in such a condition that the power-on reset or initial-microprogram-loading (IML) function must be performed. Since the function performed by DIAGNOSE may differ from model to model and between versions of a model, the program should avoid issuing DIAGNOSE unless the program recognizes both the model number and version code stored by STORE CPU ID.

EXTRACT AND SET EXTENDED AUTHORITY



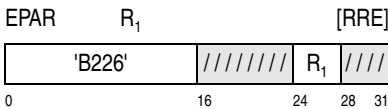
The extended authorization index (EAX), bits 32-47 of control register 8, is saved in bit positions 32-47 of the first operand, and then the EAX in control register 8 is replaced by the contents of bit positions 48-63 of the first operand. Bits 0-31 of the first operand are ignored, and bits 0-31 and 48-63 of the operand remain unchanged.

Condition Code: The code remains unchanged.

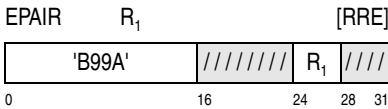
Program Exceptions:

- Privileged operation

EXTRACT PRIMARY ASN



EXTRACT PRIMARY ASN AND INSTANCE



The 16-bit PASN, bits 48-63 of control register 4, is placed in bit positions 48-63 of general register R₁. Bits 32-47 of the general register are set to zeros.

In the EXTRACT PRIMARY ASN AND INSTANCE operation, the PASTEIN, bits 0-31 of control register 4, is placed in bit positions 0-31 of general register R₁. In the EXTRACT PRIMARY ASN operation, bits 0-31 of general register R₁ remain unchanged.

Special Conditions

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized.

In the problem state, the extraction-authority control, bit 36 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-7.

Condition Code: The code remains unchanged.

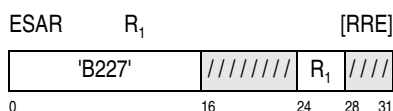
Program Exceptions:

- Operation (if the ASN-and-LX-reuse facility is not installed, EPAIR only)
- Privileged operation (extraction-authority control is zero in the problem state)
- Special operation

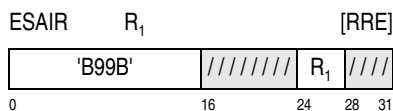
1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B.1	Operation exception if the ASN-and-LX-reuse facility is not installed (EPAIR only).
7.B.2	Special-operation exception due to DAT being off.
8.	Privileged-operation exception due to extraction-authority control, bit 36 of control register 0, being zero in problem state.

Figure 10-7. Priority of Execution: *EXTRACT PRIMARY ASN*

EXTRACT SECONDARY ASN



EXTRACT SECONDARY ASN AND INSTANCE



The 16-bit SASN, bits 48-63 of control register 3, is placed in bit positions 48-63 of general register R_1 . Bits 32-47 of the general register are set to zeros.

In the *EXTRACT SECONDARY ASN AND INSTANCE* operation, the SASTEIN, bits 0-31 of control register 3, is placed in bit positions 0-31 of general register R_1 . In the *EXTRACT SECONDARY ASN* operation, bits 0-31 of general register R_1 remain unchanged.

Special Conditions

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized.

In the problem state, the extraction-authority control, bit 36 of control register 0, must be one; otherwise, a

privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-8 on page 10-21.

Condition Code: The code remains unchanged.

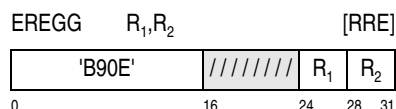
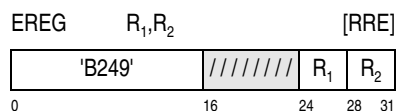
Program Exceptions:

- Operation (if the ASN-and-LX-reuse facility is not installed, ESAIR only)
- Privileged operation (extraction-authority control is zero in the problem state)
- Special operation

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B.1	Operation exception if the ASN-and-LX-reuse facility is not installed (ESAIR only).
7.B.2	Special-operation exception due to DAT being off.
8.	Privileged-operation exception due to extraction-authority control, bit 36 of control register 0, being zero in problem state.

Figure 10-8. Priority of Execution: *EXTRACT SECONDARY ASN*

EXTRACT STACKED REGISTERS



Contents of a set of general registers and a set of access registers that were saved in the last state entry in the linkage stack are restored to the registers. Each set of registers begins with register R_1 and ends with register R_2 .

For EXTRACT STACKED REGISTERS (EREG), the contents of bit positions 32-63 of the general registers are restored, and the contents of bit positions 0-31 of the registers remain unchanged. For EXTRACT STACKED REGISTERS (EREKG), the contents of bit positions 0-63 of the general registers are restored. In either case, the contents of bit positions 0-31 of the access registers are restored.

For each of the general registers and the access registers, the registers are loaded in ascending order of their register numbers, starting with register R₁ and continuing up to and including register R₂, with register 0 following register 15. The bit positions of each register are loaded from the position in the state entry where the contents of the bit positions were saved when the state entry was created. The contents of the state entry remain unchanged.

The last state entry is located as described in “Unstacking Process” on page 5-82. The state entry remains in the linkage stack, and the linkage-stack-entry address in control register 15 remains unchanged.

Key-controlled protection does not apply to references to the linkage stack.

Special Conditions

The CPU must be in the primary-space mode, access-register mode, or home-space mode; otherwise, a special-operation exception is recognized.

A stack-empty, stack-specification, or stack-type exception may be recognized during the unstacking process.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-9 on page 10-22.

Condition Code: The code remains unchanged.

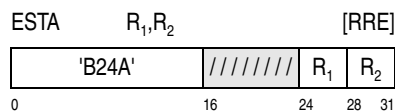
Program Exceptions:

- Access (fetch, except for protection, linkage-stack entry)
- Special operation
- Stack empty
- Stack specification
- Stack type

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off or the CPU being in secondary-space mode.
8.	Access exceptions (fetch) for entry descriptor of the current linkage-stack entry.
9.	Stack-type exception due to current entry not being a state entry or header entry. Note: Exceptions 10-14 can occur only if the current entry is a header entry.
10.	Access exceptions (fetch) for second word of the header entry.
11.	Stack-empty exception due to backward stack-entry validity bit in the header entry being zero.
12.	Access exceptions (fetch) for entry descriptor of preceding entry, which is the entry designated by the backward stack-entry address in the current (header) entry.
13.	Stack-specification exception due to preceding entry being a header entry.
14.	Stack-type exception due to preceding entry not being a state entry.
15.	Access exceptions (fetch) for the selected contents of the state entry.

Figure 10-9. Priority of Execution: EXTRACT STACKED REGISTERS

EXTRACT STACKED STATE



Sixty-four or 128 bits of status information in the last state entry in the linkage stack are placed in the pair of general registers designated by the R_1 field. The condition code is set to indicate whether the state entry is a branch state entry or a program-call state entry.

The R_1 field designates the even-numbered register of an even-odd pair of general registers.

Bits 56-63 of general register R_2 are an unsigned binary integer that is used as a code to select the state-entry byte positions, or byte and bit positions, from which information is to be extracted, as follows:

Code (Bits 56-63 of Gen. Reg. R_2)	State-Entry Byte Positions, or Byte and Bit Positions, Selected
0	128-135
1	136-139, 140.0, and 168-175.33-63 (see text)
2	144-151
3	152-159
4	136-143 and 168-175
5	176-183

For a code of 0, 2, or 3 in bit positions 56-63 of general register R_2 , the contents of the leftmost four bytes of the eight bytes of status information are placed in bit positions 32-63 of general register R_1 , and the contents of the rightmost four bytes of the status information are placed in bit positions 32-63 of general register $R_1 + 1$. The contents of bit positions 0-31 of general registers R_1 and $R_1 + 1$ remain unchanged.

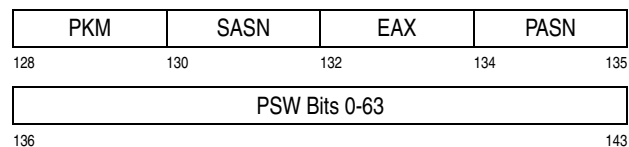
For a code of 1 in bit positions 56-63 of general register R_2 , the contents of bytes 136-139 of the state entry, which are bits 0-31 of the PSW in the state entry, are placed in bit positions 32-63 of general register R_1 ; the contents of bit position 0 of byte 140 of the entry, which is bit 32 of that PSW, are placed in bit position 32 of general register $R_1 + 1$; and the contents of bit positions 33-63 of bytes 168-175 of the entry, which are bits 97-127 of the PSW, are placed in bit positions 33-63 of general register

$R_1 + 1$. However, bit 44 of general register R_1 , which corresponds to bit 12 of the PSW in the state entry, is set to one, indicating the ESA/390 mode. Also, if bits 0-32 of bytes 168-175 of the state entry are not all zeros, bit 63 of general register $R_1 + 1$ is set to one; otherwise, bit 63 remains with the value loaded from bit position 63 of bytes 168-175 of the state entry. The contents of bit positions 0-31 of general registers R_1 and $R_1 + 1$ remain unchanged.

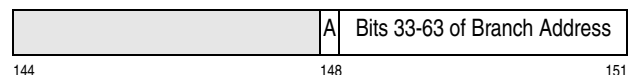
For a code of 4 in bit positions 56-63 of general register R_2 , the contents of bytes 136-143 of the state entry, which are bits 0-63 of the PSW in the state entry, are placed in bit positions 0-63 of general register R_1 , and the contents of bytes 168-175 of the state entry, which are bits 64-127 of that PSW, are placed in bit positions 0-63 of general register $R_1 + 1$.

Code 5 in bit positions 56-63 of general register R_2 is valid only if the ASN-and-LX-reuse facility is installed. For code 5, the contents of the leftmost four bytes of the eight bytes of status information are placed in bit positions 0-31 of general register R_1 , and the contents of the rightmost four bytes of the status information are placed in bit positions 0-31 of general register $R_1 + 1$. The contents of bit positions 32-63 of general registers R_1 and $R_1 + 1$ remain unchanged.

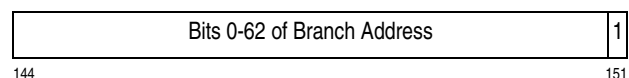
The format of byte positions 128-183 of the state entry is as follows:



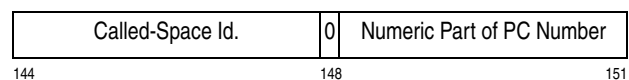
In a Branch State Entry Made in 24-Bit or 31-Bit Mode



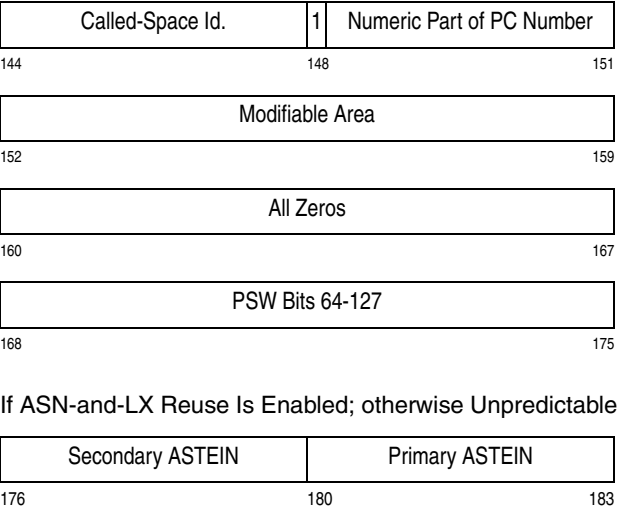
In a Branch State Entry Made in 64-Bit Mode



In a Program-Call State Entry Made When Resulting Mode Is 24 Bit or 31 Bit



In a Program-Call State Entry Made When Resulting Mode Is 64 Bit



The contents of the state entry remain unchanged.

The last state entry is located as described in “Unstacking Process” on page 5-82. The state entry remains in the linkage stack, and the linkage-stack-entry address in control register 15 remains unchanged.

When the entry-type code in the entry descriptor of the state entry is 0001100 binary, indicating a branch state entry, the condition code is set to 0. When the entry-type code is 0001101 binary, indicating a program-call state entry, the condition code is set to 1.

Key-controlled protection does not apply to references to the linkage stack.

Bits 0-55 of general register R₂ are ignored.

Special Conditions

A specification exception is recognized when R₁ is odd or the code in bit positions 56-63 of general register R₂ is greater than 4 when the ASN-and-LX-reuse facility is not installed or is greater than 5 when the facility is installed.

The CPU must be in the primary-space mode, access-register mode, or home-space mode; otherwise, a special-operation exception is recognized.

A stack-empty, stack-specification, or stack-type exception may be recognized during the unstacking process.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-10 on page 10-25.

Resulting Condition Code:

- 0 Branch state entry
- 1 Program-call state entry
- 2 --
- 3 --

Program Exceptions:

- Access (fetch, except for protection, linkage-stack entry)
- Special operation
- Specification
- Stack empty
- Stack specification
- Stack type

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off or the CPU being in secondary-space mode.
8.A	Specification exception due to R_1 being odd or bits 56-63 of general register R_2 having a value greater than 4 when the ASN-and-LX-reuse facility is not installed or greater than 5 when the facility is installed.
8.B.1	Access exceptions (fetch) for entry descriptor of the current linkage-stack entry.
8.B.2	Stack-type exception due to current entry not being a state entry or header entry. Note: Exceptions 8.B.3-8.B.7 can occur only if the current entry is a header entry.
8.B.3	Access exceptions (fetch) for second word of the header entry.
8.B.4	Stack-empty exception due to backward stack-entry validity bit in the header entry being zero.
8.B.5	Access exceptions (fetch) for entry descriptor of preceding entry, which is the entry designated by the backward stack-entry address in the current (header) entry.
8.B.6	Stack-specification exception due to preceding entry being a header entry.
8.B.7	Stack-type exception due to preceding entry not being a state entry.
8.B.8	Access exceptions (fetch) for the selected contents of the state entry.

Figure 10-10. Priority of Execution: EXTRACT STACKED STATE

Programming Note: The results for a code of 1 in bit positions 56-63 of general register R_2 are intended to provide compatibility with ESA/390. (It may be that only values of bits in bit positions 0-31 of the PSW are required.) Bit 63 of general register $R_1 + 1$ is set to one if the instruction address in the PSW in the state entry is larger than a 31-bit address.

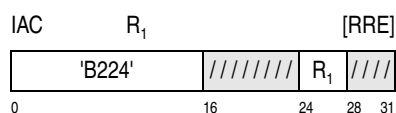
tions 54 and 55 of general register R_1 ; that is, bit 16 is placed in bit position 55, and bit 17 is placed in bit position 54. Bits 48-53 of the register are set to zeros, and bits 0-47 and 56-63 of the register remain unchanged. The address-space-control bits are also used to set the condition code.

Special Conditions

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized.

In the problem state, the extraction-authority control, bit 36 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

INSERT ADDRESS SPACE CONTROL



The address-space-control bits, bits 16 and 17 of the current PSW, are placed in reversed order in bit posi-

The priority of recognition of program exceptions for the instruction is shown in Figure 10-11 on page 10-26.

Resulting Condition Code:

- 0 PSW bits 16 and 17 zeros (indicating primary-space mode)
- 1 PSW bit 16 one and bit 17 zero (indicating secondary-space mode)
- 2 PSW bit 16 zero and bit 17 one (indicating access-register mode)
- 3 PSW bits 16 and 17 ones (indicating home-space mode)

Program Exceptions:

- Privileged operation (extraction-authority control is zero in the problem state)
- Special operation

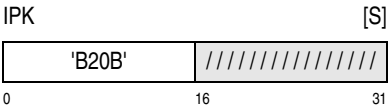
1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off.
8.	Privileged-operation exception due to extraction-authority control, bit 36 of control register 0, being zero in problem state.

Figure 10-11. Priority of Execution: INSERT ADDRESS SPACE CONTROL

Programming Notes:

- 1. Bits 48-53 of general register R₁ are reserved for expansion for use with possible future facilities. The program should not depend on these bits being set to zeros.
- 2. INSERT ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL are defined to operate on the seventh byte of a general register so that the address-space-control bits can be saved in the same general register as the PSW key, which is placed in the eighth byte of general register 2 by INSERT PSW KEY.

INSERT PSW KEY



The four-bit PSW-key, bits 8-11 of the current PSW, is inserted in bit positions 56-59 of general register 2, and bits 60-63 of that register are set to zeros. Bits 0-55 of general register 2 remain unchanged.

Special Conditions

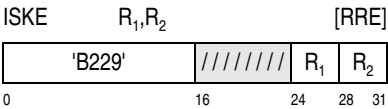
In the problem state, the extraction-authority control, bit 36 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

Condition Code: The code remains unchanged.

Program Exceptions:

- Privileged operation (extraction-authority control is zero in the problem state)

INSERT STORAGE KEY EXTENDED



The storage key for the block that is addressed by the contents of general register R₂ is inserted in general register R₁.

In the 24-bit addressing mode, bits 40-51 of general register R₂ designate a 4K-byte block in real storage, and bits 0-39 and 52-63 of the register are ignored. In the 31-bit addressing mode, bits 33-51 of general register R₂ designate a 4K-byte block in real storage, and bits 0-32 and 52-63 of the register are ignored. In the 64-bit addressing mode, bits 0-51 of general register R₂ designate a 4K-byte block in real storage, and bits 52-63 of the register are ignored.

The address designating the storage block, being a real address, is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

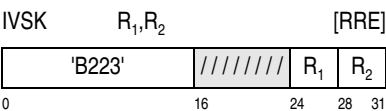
The seven-bit storage key is inserted in bit positions 56-62 of general register R_1 , and bit 63 is set to zero. The contents of bit positions 0-55 of the register remain unchanged.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (address specified by general register R_2)
- Privileged operation

INSERT VIRTUAL STORAGE KEY



The storage key for the location designated by the virtual address in general register R_2 is inserted in general register R_1 .

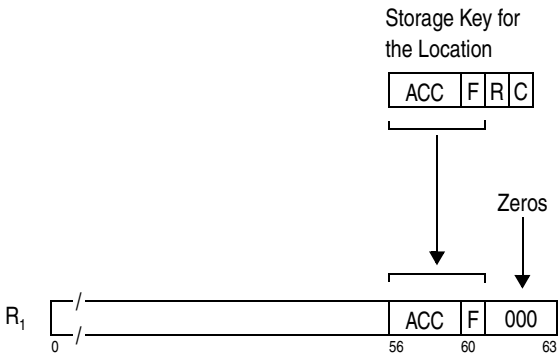
Selected bits of general register R_2 are used as a virtual address. In the 24-bit addressing mode, the address is specified by bits 40-63 of the register, and bits 0-39 are ignored. In the 31-bit addressing mode, the address is specified by bits 33-63, and bits 0-32 is ignored. In the 64-bit addressing mode, the address is specified by bits 0-63 of the register.

The address is a virtual address and is subject to the address-space-control bits, bits 16 and 17 of the current PSW. The address is treated as a primary virtual address in the primary-space mode, as a secondary virtual address in the secondary-space mode, as an AR-specified virtual address in the access-register mode, or as a home virtual address in the home-space mode. The reference to the storage key is not subject to a protection exception.

Bits 0-4 of the storage key, which are the access-control bits and the fetch-protection bit, are placed in bit positions 56-60 of general register R_1 , with bits 61-63 set to zeros. The contents of bit positions 0-55 of the

register remain unchanged. The change and reference bits in the storage key are not inspected. The change bit is not affected by the operation. The reference bit, depending on the model, may or may not be set to one as a result of the operation.

The following diagram shows the storage key and the register positions just described.



Special Conditions

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized.

In the problem state, the extraction-authority control, bit 36 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-12 on page 10-28.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (except for protection, address specified by general register R_2)
- Privileged operation (extraction-authority control is zero in the problem state)
- Special operation

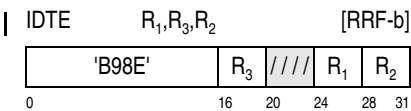
1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off.
8.	Privileged-operation exception due to extraction-authority control, bit 36 of control register 0, being zero.
9.	Access exceptions (except for protection) for address specified by general register R ₂ .

Figure 10-12. Priority of Execution: INSERT VIRTUAL STORAGE KEY

Programming Notes:

- 1. Since all bytes in a 4K-byte block are associated with the same page and the same storage key, bits 52-63 of general register R₂ essentially are ignored. Similarly, since all bytes in a 1M-byte block are associated with the same segment, bits 44-63 of general register R₂ may be ignored when enhanced DAT applies and the STE-format and ACCF-validity controls are both one.
- 2. In the access-register mode, access register 0 designates the primary address space regardless of the contents of access register 0.

INVALIDATE DAT TABLE ENTRY



When the clearing-by-ASCE-option bit, bit 52 of general register R₂, is zero, an operation called the invalidation-and-clearing operation is performed, as follows. The designated region-table entry or segment-table entry in storage, or a range of entries beginning with the designated entry, is invalidated, and the translation-lookaside buffers (TLBs) in all CPUs in the configuration are cleared of (1) all combined region-and-segment-table entries (CRSTEs) and page-table entries (PTEs) formed through use of the invalidated entry or entries in storage and (2) all PTEs formed through use of a CRSTE formed

through use of the invalidated entry or entries in storage. The TLB entries cleared may optionally be limited to entries formed to translate addresses in a specified address space.

When the clearing-by-ASCE-option bit is one, an operation called the clearing-by-ASCE operation is performed, as follows. The operation does not perform any invalidation of DAT-table entries in storage, but it does clear, from the TLBs in all CPUs in the configuration, all combined region-and-segment-table entries and page-table entries formed to translate addresses in a specified address space.

The two operations are described separately below, before the section "Common Operation."

Invalidation-and-Clearing Operation

When bit 52 of general register R₂, the clearing-by-ASCE-option bit, is zero, the invalidation-and-clearing operation is specified.

The contents of general register R₁ have the format of an address-space-control element with only the table origin, bits 0-51, and designation-type control (DT), bits 60 and 61, used. The table origin designates the DAT table in which one or more entries are to be invalidated, and DT specifies the type of that table.

Bits 52-59, 62, and 63 of general register R₁ are ignored.

Bits 0-43 of general register R₂ have the format of the region index and segment index of a virtual address. The part of bits 0-43 normally used by DAT to select an entry in the type of table designated by general register R₁ is called the effective invalidation index.

Bits 60 and 61 of general register R₁ specify a table type and an effective invalidation index as follows:

Bits 60 and 61 of Reg. R ₁		Effective Invalidation Index in Reg. R ₂
Table Type		
11	Region first table	Region first index (bits 0-10)
10	Region second table	Region second index (bits 11-21)

Bits 60 and 61 of Reg. R_1	Table Type	Effective Invalidation Index in Reg. R_2
01	Region third table	Region third index (bits 22-32)
00	Segment table	Segment index (bits 33-43)

The part of bits 0-43 of general register R_2 to the right of the effective invalidation index is ignored.

The table origin in general register R_1 and the effective invalidation index designate a DAT-table entry to be invalidated. Bits 53-63 of general register R_2 are an unsigned binary integer specifying the number of additional table entries to be invalidated. Therefore,

the number of entries to be invalidated is 1-2,048, corresponding to a value of bits 53-63 of 0-2,047.

Bits 44-51 of general register R_2 must be zeros; otherwise, a specification exception is recognized.

If R_3 is nonzero, the contents of general register R_3 have the format of an address-space-control element with only the table origin, bits 0-51, and designation-type control (DT), bits 60 and 61, used. These contents are used to select TLB entries to be cleared. Bits 52-59, 62, and 63 of general register R_3 are ignored. If R_3 is zero, the entire contents of general register 0 are ignored.

The contents of the general registers just described are shown in Figure 10-13 on page 10-29.

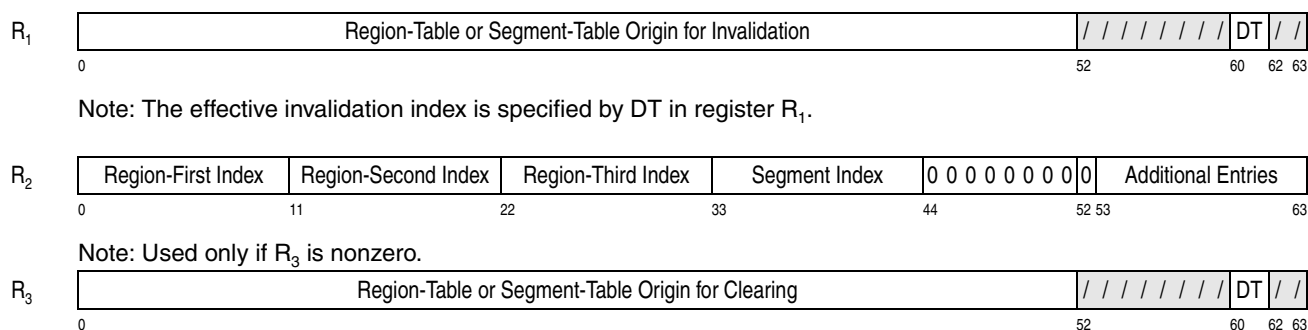


Figure 10-13. Register Contents for INVALIDATE DAT TABLE ENTRY Invalidation-and-Clearing Operation (Bit 52 of GR R_2 Is Zero)

The table origin in general register R_1 and effective invalidation index in general register R_2 designate a table entry in accordance with the rules in “Lookup in a Table Designated by an Address-Space-Control Element” on page 3-48, except that a carry from bit position 0 of the resulting address is always ignored, and the index is not checked against a table-length field. The table origin is treated as a 64-bit address, and the addition is performed by using the rules for 64-bit address arithmetic, regardless of the current addressing mode specified by bits 31 and 32 of the current PSW. The address formed from these two components is a real or absolute address. The invalid bit, bit 58, of this region-table or segment-table entry is set to one. During this procedure, the entry is not checked for a format error or for whether the origin, in the entry, of the next-lower-level table would cause an addressing exception. The table-type field in the entry is ignored. If the entry is a segment-table entry, the common-segment bit in the entry is ignored.

The entire table entry is fetched concurrently from storage. Subsequently, the byte containing the invalid bit is stored. The fetch access to the entry is subject to key-controlled protection, and the store access is subject to key-controlled protection and low-address protection.

If bits 53-63 of general register R_2 are not all zeros, the setting of the invalid bit to one in a region-table or segment-table entry is repeated by adding one to the previously used value of the effective invalidation index, and this is done as many times as are specified by bits 53-63. A carry out of the leftmost bit position of the effective invalidation index is ignored, and wraparound in the table occurs in this case. The contents of general register R_2 remain unchanged.

A serialization function is performed before the operation begins and again after the operation is completed. As is the case for all serialization operations,

this serialization applies only to this CPU; other CPUs are not necessarily serialized.

After it has set an invalid bit to one, this CPU clears selected entries from its TLB and signals all other CPUs in the configuration to clear selected entries from their TLBs. Each TLB is cleared of at least those combined region-and-segment-table entries (CRSTEs) for which all of the following conditions are met:

- The effective invalidation index and all bits to the left of it in general register R_2 match the same part of the region-and-segment-index field in the CRSTE. Note that when multiple table entries are invalidated due to bits 53-63 of general register R_2 , then the effective invalidation index is incremented, a carry out of the leftmost bit position of the index is lost, and CRSTEs are cleared for each value of the index so obtained.
- Either R_3 is zero or the table-origin and designation-type fields in general register R_3 match the table-origin and designation-type fields in the address-space-control element (ASCE) used to form the CRSTE. This ASCE is the one that attached the translation path from which the CRSTE was formed. See “Formation of TLB Entries” on page 3-53 for the meaning of the terminology used here.
- If the entry invalidated in storage is a segment-table entry, the page-table-origin field in the invalidated entry matches the page-table-origin field in the CRSTE.

Each TLB is also cleared of at least those page-table entries (PTEs) for which all of the following conditions are met:

- The TLB PTE was formed through use of an entry invalidated in storage or through use of a CRSTE formed through use of an entry invalidated in storage.
- Either R_3 is zero or the table-origin and designation-type fields in general register R_3 match the table-origin and designation-type fields in the address-space-control element (ASCE) used to form the TLB PTE. This ASCE may be one that attached a translation path containing a segment-table entry that attached the PTE in storage from which the TLB PTE was formed, or it may be one that made usable a CRSTE that attached the PTE in storage from which the TLB

PTE was formed. See “Formation of TLB Entries” on page 3-53 for the meaning of the terminology used here.

- If the entry invalidated in storage is a segment-table entry, the page-table-origin field in the invalidated entry matches the page-table-origin field in the TLB PTE.

Storing in the region- or segment-table entry and the clearing of TLB entries may or may not occur if the invalid bit is already one in the region- or segment-table entry.

When multiple entries are invalidated, clearing of TLB entries may be delayed until all entries have been invalidated.

Clearing-by-ASCE Operation

When bit 52 of general register R_2 , the clearing-by-ASCE-option bit, is one, the clearing-by-ASCE operation is specified.

The contents of general register R_3 have the format of an address-space-control element with only the table origin, bits 0-51, and designation-type control (DT), bits 60 and 61, used. These contents are used to select TLB entries to be cleared. Bits 52-59, 62, and 63 of general register R_3 are ignored. R_3 may be zero or nonzero, that is, any general register, including register 0, may be designated.

Bits 44-51 of general register R_2 must be zeros; otherwise, a specification exception is recognized.

The contents of general register R_1 and of bit positions 0-43 and 53-63 of general register R_2 are ignored.

The contents of the general registers just described are shown in Figure 10-14 on page 10-31.

The TLBs of all CPUs in the configuration are cleared of at least those combined region-and-segment-table entries (CRSTEs) for which the table-origin and designation-type fields in general register R_3 match the table-origin and designation-type fields in the address-space-control element (ASCE) used to form the CRSTE. This ASCE is the one that attached the translation path from which the CRSTE was formed. See “Formation of TLB Entries” on page 3-53 for the meaning of the terminology used here.

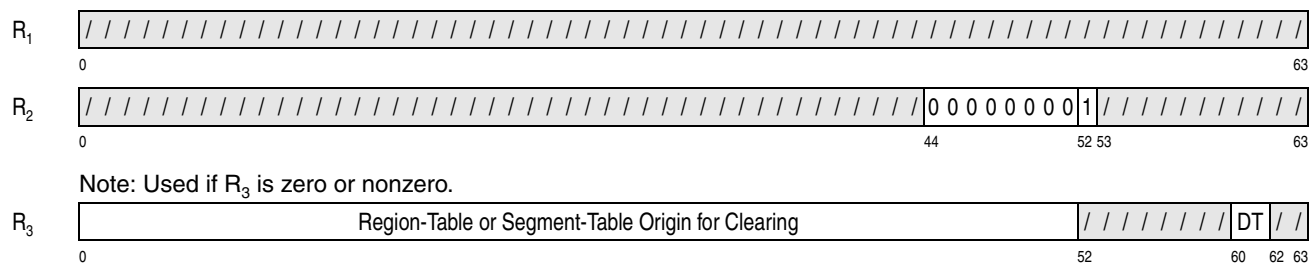


Figure 10-14. Register Contents for INVALIDATE DAT TABLE ENTRY Clearing-by-ASCE Operation (Bit 52 of GR R2 Is One)

Each TLB is also cleared of at least those page-table entries (PTEs) for which the table-origin and designation-type fields in general register R₃ match the table-origin and designation-type fields in the address-space-control element (ASCE) used to form the TLB PTE. This ASCE may be one that attached a translation path containing a segment-table entry that attached the PTE in storage from which the TLB PTE was formed, or it may be one that made usable a CRSTE that attached the PTE in storage from which the TLB PTE was formed. See “Formation of TLB Entries” on page 3-53 for the meaning of the terminology used here.

Common Operation

When the clearing-by-ASCE-option bit, bit 52 of general register R₂, is either zero or one, the execution of INVALIDATE DAT TABLE ENTRY is not completed on the CPU which executes it until (1) all entries corresponding to the specified parameters have been cleared from the TLB of this CPU and (2) all other CPUs in the configuration have completed any storage accesses, including the updating of the change and reference bits, by using TLB entries corresponding to the specified parameters.

The operations do not necessarily have any effect on TLB real-space entries.

Special Conditions

Bits 44-51 of general register R₂ must be zeros; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions (invalidation-and-clearing operation only).

Resulting Condition Code: The code is unpredictable.

Program Exceptions:

- Addressing (invalidated region- or segment-table entry, invalidation-and-clearing operation only).
- Privileged operation
- Operation (if the DAT-enhancement facility is not installed)
- Protection (fetch and store, region- or segment-table entry, key-controlled protection and low-address protection; invalidation-and-clearing operation only).
- Specification

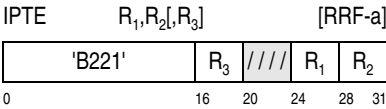
Programming Notes:

1. The selective clearing of TLB entries may be implemented in different ways, depending on the model, and, in general, more entries may be cleared than the minimum number required. When the invalidation-and-clearing operation is performed, some models may clear all TLB entries when the effective invalidation index is not a segment index or may clear an entry regardless of the page-table origin in the entry. When that operation or the clearing-by-ASCE operation is performed, some models may clear a TLB entry regardless of the designation-type field in general register R₃. When either operation is performed, other models may clear precisely the minimum number of entries required. Therefore, in order for a program to operate on all models, the program should not take advantage of any properties obtained by a less selective clearing on a particular model.
2. When using the clearing-by-ASCE operation to clear TLB entries associated with common segments, note that these entries may have been formed through use of address-space-control elements containing many different table origins.

The following notes apply when the invalidation-and-clearing operation is specified.

- 3. The clearing of TLB entries may make use of the page-table origin in a segment-table entry. Therefore, if the segment-table entry, when in the attached state, ever contained a page-table origin that is different from the current value, copies of entries containing the previous values may remain in the TLB.
- 4. INVALIDATE DAT TABLE ENTRY cannot be safely used to update a shared location in main storage if the possibility exists that another CPU or a channel program may also be updating the location.
- 5. The address of the DAT-table entry for INVALIDATE DAT TABLE ENTRY is a 64-bit address, and the address arithmetic is performed by following the normal rules for 64-bit address arithmetic, with wraparound at $2^{64} - 1$. Also, offset and length fields are not used. Contrast this with implicit translation and the translations for LOAD REAL ADDRESS and STORE REAL ADDRESS, all of which may result either in wraparound or in an addressing exception when a carry occurs out of bit position 0 and which indicate an exception condition when the designated entry does not lie within its table. Accordingly, the DAT tables should not be specified to wrap from maximum storage locations to location 0, and the first designated entry and all additional entries specified by bits 53-63 of general register 3 should lie within the designated table.

INVALIDATE PAGE TABLE ENTRY



The designated page-table entries are invalidated, and the translation-lookaside buffers (TLBs) in all CPUs in the configuration are cleared of the associated entries.

The contents of general register R_1 have the format of a segment-table entry, with only the page-table origin used. The contents of general register R_2 have the format of a virtual address, with only the page index used. The contents of fields that are not part of the page-table origin or page index are ignored.

When the IPTE-range facility is not installed, or when the R_3 field is zero, the single page-table entry designated by the first and second operands is invalidated.

When the IPTE-range facility is installed and the R_3 field is nonzero, bits 56-63 of general register R_3 contain an unsigned binary integer specifying the count of additional page-table entries to be invalidated. Therefore, the number of page-table entries to be invalidated is 1-256, corresponding to a value of 0-255 in bits 56-63 of the register.

The contents of the general registers just described are shown in Figure 10-15 on page 10-32.

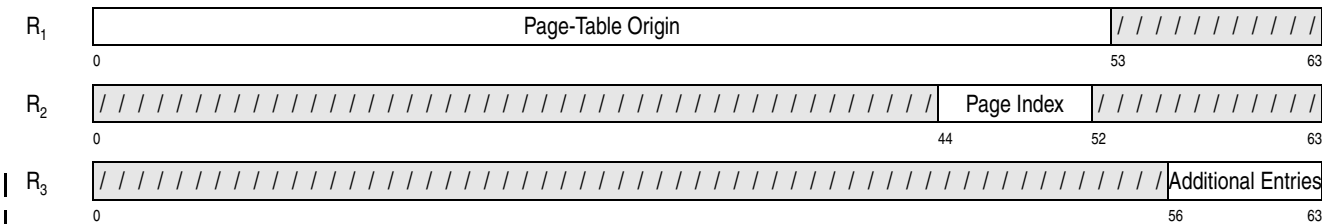


Figure 10-15. Register Contents for INVALIDATE PAGE TABLE ENTRY

When the IPTE-range facility is not installed, the R_3 field is ignored but should contain zeros; otherwise, the program may not operate compatibly in the future.

The page-table origin and the page index designate a page-table entry, following the dynamic-address-translation rules for page-table lookup. The page-table origin is treated as a 64-bit address, and the

addition is performed by using the rules for 64-bit address arithmetic, regardless of the current addressing mode, which is specified by bits 31 and 32 of the current PSW. A carry out of bit position 0 as a result of the addition of the page index and page-table origin cannot occur. The address formed from these two components is a real or absolute address. The page-invalid bit of this page-table entry is set to one. During this procedure, the page-table entry is

not inspected for whether the page-invalid bit is already one or for format errors. Additionally, the page-frame real address contained in the entry is not checked for an addressing exception.

When the IPTE-range facility is installed and the R_3 field is nonzero, the instruction is interruptible, and processing is as follows:

1. The invalidation process described above is repeated for each subsequent entry in the page table until either the number of additional entries specified in bits 56-63 of general register R_3 have been invalidated or an interruption occurs.
2. The page-index in bits 44-51 of general register R_2 is incremented by the number of page-table entries that were invalidated; a carry out of bit position 44 of general register R_2 is ignored.
3. The additional-entry count in bits 56-63 of general register R_3 is decremented by the number of page-table entries that were invalidated.

Therefore, when the IPTE-range facility is installed, the R_3 field is nonzero, and an interruption occurs (other than one that causes termination), general registers R_2 and R_3 have been updated, so that the instruction, when reexecuted, resumes at the point of interruption.

When the IPTE-range facility is not installed, or when the R_3 field is zero, the contents of registers R_2 , and R_3 remain unchanged.

For each page-table entry that is invalidated, the entire page-table entry appears to be fetched concurrently from storage as observed by other CPUs. Subsequently, the byte containing the page-invalid bit is stored. The fetch access to each page-table entry is subject to key-controlled protection, and the store access is subject to key-controlled protection and low-address protection.

A serialization function is performed before the operation begins and again after the operation is completed. As is the case for all serialization operations, this serialization applies only to this CPU; other CPUs are not necessarily serialized.

If no exceptions are recognized, this CPU clears selected entries from its TLB and signals all CPUs in the configuration to clear selected entries from their TLBs. For each page-table entry invalidated, each

TLB is cleared of at least those entries that have been formed using all of the following:

- The page-table origin specified by general register R_1
- The page index specified by general register R_2
- The page-frame real address contained in the designated page-table entry

The execution of INVALIDATE PAGE TABLE ENTRY is not completed on the CPU which executes it until (1) all entries corresponding to the specified parameters have been cleared from the TLB of this CPU and (2) all other CPUs in the configuration have completed any storage accesses, including the updating of the change and reference bits, by using TLB entries corresponding to the specified parameters.

Special Conditions

When the IPTE-range facility is installed, the R_3 field is nonzero, and the page index in general register R_2 plus the additional-entry count in general register R_3 is greater than 255, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (page-table entry)
- Privileged operation
- Protection (fetch and store, page-table entry, key-controlled protection, and low-address protection)
- Specification

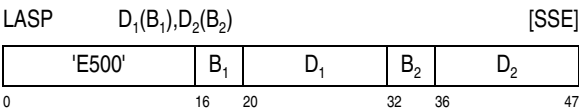
Programming Notes:

1. The selective clearing of entries may be implemented in different ways, depending on the model, and, in general, more entries may be cleared than the minimum number required. Some models may clear all entries which contain the page-frame real addresses obtained from the page-table entries in storage. Others may clear all entries which contain the designated page index (or indices), and some implementations may clear precisely the minimum number of

entries required. Therefore, in order for a program to operate on all models, the program should not take advantage of any properties obtained by a less selective clearing on a particular model.

- 2. The clearing of TLB entries may make use of the page-frame real addresses in the page-table entries. Therefore, if the page-table entries, when in the attached state, ever contained page-frame real addresses that are different from the current values, copies of entries containing the previous values may remain in the TLB.
- 3. INVALIDATE PAGE TABLE ENTRY cannot be safely used to update a shared location in main storage if the possibility exists that another CPU or a channel program may also be updating the location.
- 4. When the IPTE-range facility is installed and the R_3 field is nonzero, the following applies:
 - a. All of the page-table entries to be invalidated must reside in the same page table. A specification exception is recognized if the page index in general register R_1 plus the additional-entry count in general register R_3 is greater than the maximum page index of 255.
 - b. The number of page-table entries that are invalidated by INVALIDATE PAGE TABLE ENTRY may vary from one execution to another.
 - c. The instruction cannot be used for situations where the program must rely on uninterrupted execution of the instruction. Similarly, the program should normally not use INVALIDATE PAGE TABLE ENTRY to invalidate a page-table entry, the page-frame-real address of which designates the 4K-byte block containing the instruction or of an execute-type instruction that executes the IPTE.
 - d. Further programming notes concerning interruptible instructions are included in "Interruptible Instructions" on page 5-20.

LOAD ADDRESS SPACE
PARAMETERS



The first operand contains values to be loaded into control registers 3 and 4, including a secondary ASN (SASN) and a primary ASN (PASN), and, possibly, a secondary ASTE instance number (SASTEIN) and a primary ASTE instance number (PASTEIN). Execution of the instruction consists in performing four major steps: PASN translation, SASN translation, SASN authorization, and control-register loading. Each of these steps may or may not be performed, depending on the outcome of certain tests and on the setting of bits 61-63 of the second-operand address. The first three of these steps, when performed and successful, obtain additional values, that are loaded into control registers 1, 5, and 7. When the first three steps are not successful when performed, no control registers are changed, and the reason is indicated in the condition code.

When the ASN-and-LX-reuse facility is not installed, or is installed but is not enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, the first operand is a doubleword containing a PSW-key mask (PKM), a SASN, an authorization index (AX), and a PASN. When the ASN-and-LX-reuse facility is installed and enabled, the first operand is two consecutive doublewords containing those four values and also a SASTEIN and a PASTEIN.

The primary ASN may be translated by means of the ASN-translation tables to obtain a primary-ASTE (PASTE) origin (PASTEO) and, from the PASTE, a primary ASCE (PASCE). The secondary ASN may be translated by means of the ASN-translation tables to obtain a secondary ASCE (SASCE). If the first operand is two doublewords and PASN or SASN translation occurs, the ASTEIN in the located ASTE is checked for being equal to the PASTEIN or SASTEIN, respectively, in the first operand. An authority check for ensuring that the new AX is authorized to establish the new SASN can be prevented when SASN translation occurs, or it can be required and cause SASN translation even when SASN translation would not otherwise occur. The new AX, which is placed in a control register and is used in SASN authorization if that occurs, is the old AX if PASN

translation has not occurred or is the AX in the located PASTE otherwise. However, in either of these cases, the new AX may be the AX specified in the first operand.

A doubleword first operand has the following format:

First Operand if ASN-and-LX Reuse Is Not Enabled

PKM-d	SASN-d	AX-d	PASN-d	
0	16	32	48	63

A two-doubleword first operand has the following format:

First Operand if ASN-and-LX Reuse Is Enabled

SASTEIN-d		PKM-d	SASN-d
0	32	48	63
PASTEIN-d		AX-d	PASN-d
64	96	112	127

The “d” stands for designated value and is used to distinguish these fields from other fields with similar names which are referred to in the definition. The current contents of the corresponding fields in the control registers are referred to as SASTEIN-old, PKM-old, SASN-old, etc. The updated contents of the control registers are referred to as SASTEIN-new, PKM-new, SASN-new, etc.

The second-operand address is not used to address data; instead, the rightmost three bits are used to control portions of the operation. The remainder of

the second-operand address is ignored. Bits 61-63 of the second-operand address are used as follows:

Function Specified in Second-Operand Address		
Bit	When Bit Is Zero	When Bit Is One
61	ASN translation performed only when new ASN and old ASN are different.	ASN translation performed.*
62	AX associated with new PASN used.	AX in first operand used.
63	SASN authorization performed.*	SASN authorization not performed.
Explanation: * SASN translation and SASN authorization are performed only when SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d, the SASCE is set equal to the PASCE, the SASTEIN is set equal to the PASTEIN (if ASN-and-LX reuse is enabled), and no authorization is performed.		

The operation of LOAD ADDRESS SPACE PARAMETERS is depicted in Figure 10-19 on page 10-43.

Note: In the following sections, the actions involving the PASTEIN and SASTEIN occur only if the ASN-and-LX-reuse facility is installed and is enabled by the ASN-and-LX-reuse control in control register 0.

PASN Translation and Related Processing

In the PASN-translation process, the PASN-d is translated by means of the ASN first table and the ASN second table. The PASTEO resulting from PASN translation replaces the PASTEO in control register 5. The ASCE in the located ASTE replaces the PASCE in control register 1. When bit 62 of the second-operand address is zero, the AX in the ASTE replaces the AX in control register 4. When bit 62 is one, AX-d replaces the AX in the control register. When ASN-and-LX reuse is enabled, PASTEIN-d is compared to the ASTEIN in the ASTE, and, when equal, replaces the PASTEIN in control register 4.

When bit 61 of the second-operand address is one, PASN translation is always performed. When bit 61 is zero, PASN translation is performed only if PASN-d is not equal to PASN-old. When bit 61 is zero and PASN-d is equal to PASN-old, the PASCE-old, PASTEO-old, and PASTEIN-old are left unchanged in the control registers and become the PASCE-new, PAS-

TEO-new, and PASTEIN-new, respectively. In this case, if bit 62 is zero, then the AX-old is left unchanged in the control register and becomes the AX-new, or, if bit 62 is one, AX-new is set equal to AX-d.

The PASN translation follows the normal rules for ASN translation, except that the invalid bits, bit 0 in the ASN-first-table entry and bit 0 in the ASTE, when ones, do not result in an ASN-translation exception. When either of the invalid bits is one, condition code 1 is set. Condition code 1 is also set if PASN translation occurs, the ASN-and-LX-reuse facility is enabled, and PASTEIN-d is not equal to the ASTEIN in the ASTE. When a reason for setting condition code 1 does not exist and either the current primary space-switch-event-control bit in control register 1 is one or the space-switch-event-control bit in the ASTE is one, a space-switch event does not occur; instead, condition code 3 is set. When condition code 1 or 3 is set, the control registers remain unchanged.

The contents of the AX, ASCE, and ASTEIN fields in the ASTE which is accessed as a result of the PASN translation are referred to as AX-p, ASCE-p, and ASTEIN-p, respectively. The origin of the ASTE is referred to as PASTEO-p.

The description in this paragraph applies to use of the subspace-group facility. After ASCE-p has been obtained, if (1) the subspace-group-control bit, bit 54 in ASCE-p, is one, (2) the dispatchable unit is subspace active, and (3) PASTEO-p designates the ASTE for the base space of the dispatchable unit, then a copy of ASCE-p, called ASCE-rp, is made, and bits 0-55 and 58-63 of ASCE-rp are replaced by the same bits of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. Further details are in "Subspace-Replacement Operations" on page 5-65. If bit 0 in the subspace ASTE is one, or if the ASTE sequence number (ASTESN) in the subspace ASTE does not equal the subspace ASTESN in the dispatchable-unit control table, an exception is not recognized; instead, condition code 1 is set, and the control registers remain unchanged.

SASN Translation and Related Processing

In the SASN-translation process, the SASN-d is translated by means of the ASN first table and the ASN second table. The ASCE field obtained from the ASTE subsequently replaces the secondary ASCE (SASCE) in control register 7. When ASN-and-LX reuse is enabled, SASTEIN-d is compared to the

ASTEIN in the ASTE, and, when equal, replaces the SASTEIN in control register 3.

SASN translation is performed only when, but not necessarily when, SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d, SASCE-new and SASTEIN-new are set equal to PASCE-new and PASTEIN-new, respectively. In this case, there is not a test of whether SASTEIN-d is equal to PASTEIN-d; SASTEIN-d is ignored. When SASN-d is not equal to PASN-d and is equal to SASN-old, bit 61 (force ASN translation) is zero, and bit 63 (skip SASN authorization) is one, SASN translation is not performed, and SASCE-old and SASTEIN-old become SASCE-new and SASTEIN-new, respectively. In this case, there is not a test of whether SASTEIN-d is equal to SASTEIN-old; SASTEIN-d is ignored.

SASN translation is performed in each of the following cases:

- SASN-d is not equal to PASN-d or SASN-old.
- SASN-d is not equal to PASN-d but is equal to SASN-old, and either bit 61 (force ASN translation) of the second-operand address is one or bit 63 (skip secondary authority test) of that address is zero. (The translation must be performed when bit 63 is zero in order to obtain the ATO and ATL from the SASTE.)

The SASN translation follows the normal rules for ASN translation, except that the invalid bits, bit 0 in the ASN-first-table entry and bit 0 in the ASTE, when ones, do not result in an ASN-translation exception. When either of the invalid bits is one, condition code 2 is set. Condition code 2 is also set if SASN translation occurs, the ASN-and-LX-reuse facility is enabled, and SASTEIN-d is not equal to the ASTEIN in the ASTE. When condition code 2 is set, the control registers remain unchanged.

The contents of the ASCE, ATO, ATL and ASTEIN fields in the ASTE which is accessed as a result of the SASN translation are referred to as ASCE-s, ATO-s, ATL-s, and ASTEIN-s, respectively. The origin of the ASTE is referred to as SASTEO-s.

The description in this paragraph applies to use of the subspace-group facility. After ASCE-s has been obtained, if (1) the subspace-group-control bit, bit 54 in ASCE-s, is one, (2) the dispatchable unit is subspace active, and (3) SASTEO-s designates the ASTE for the base space of the dispatchable unit,

then a copy of ASCE-s, called ASCE-rs, is made, and bits 0-55 and 58-63 of ASCE-rs are replaced by the same bits of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. Further details are in “Subspace-Replacement Operations” on page 5-65. If bit 0 in the subspace ASTE is one, or if the ASTE sequence number (ASTESN) in the subspace ASTE does not equal the subspace ASTESN in the dispatchable-unit control table, an exception is not recognized; instead, condition code 2 is set, and the control registers remain unchanged.

SASN Authorization

SASN authorization is performed when bit 63 of the second-operand address is zero and SASN-d is not equal to PASN-d; it is performed in this case regardless of whether SASN-d is equal to SASN-old. When SASN-d is equal to PASN-d or when bit 63 of the second-operand address is one, SASN authorization is not performed.

SASN authorization is performed by using ATO-s, ATL-s, and the intended value for AX-new. When bit 62 of the second-operand address is zero and PASN translation was performed, the intended value for AX-new is AX-p. When bit 62 of that address is zero and PASN translation was not performed, the AX is not changed, and AX-new is the same as AX-old. When bit 62 of that address is one, the intended value for AX-new is AX-d. SASN authorization follows the rules for secondary authorization as described in “ASN-Authorization Process” on page 3-32. If the SASN is not authorized (that is, the authority-table length is exceeded, or the selected bit is zero), condition code 2 is set, and none of the control registers is updated.

Control-Register Loading

When the PASN-translation and SASN-translation functions and related functions, and the SASN-authorization functions and subspace-replacement operations, if called for in the instruction execution, are performed without encountering any exceptions or exception conditions, the execution is completed by replacing the contents of control registers 1, 3, 4, 5, and 7 with the new values, and condition code 0 is set. The control registers are loaded as follows.

The PSW-key-mask, bits 32-47, and SASN, bits 48-63, in control register 3 are replaced by the contents of the PKM-d and SASN-d fields of the first operand.

If ASN-and-LX reuse is enabled, the SASTEIN, bits 0-31, in control register 3 is replaced as follows:

- When SASN translation is performed, by SASTEIN-d.
- When SASN translation is not performed because SASN-d is equal to PASN-d, by PASTEIN-new.
- When SASN translation is not performed because (1) SASN-d is not equal to PASN-d but is equal to SASN-old, (2) bit 61 (force ASN translation) of the second-operand address is zero, and (3) bit 63 (skip secondary authority test) of the second-operand address is one, the SASTEIN remains unchanged.

When ASN-and-LX reuse is not enabled, bits 0-31 of control register 3 always remain unchanged.

The PASN, bits 48-63 of control register 4, is replaced by the PASN-d of the first operand. If ASN-and-LX reuse is enabled, the PASTEIN, bits 0-31 of control register 4, is replaced by the PASTEIN-d of the first operand; otherwise, bits 0-31 of the register remain unchanged.

The authorization index, bits 32-47 of control register 4, is replaced as follows:

- When bit 62 of the second-operand address is one, by AX-d.
- When bit 62 of the second-operand address is zero and PASN translation is performed, by AX-p.
- When bit 62 of the second-operand address is zero and PASN translation is not performed, the authorization index remains unchanged.

The primary address-space-control element (PASCE) in control register 1 and the primary-ASN-second-table-entry origin (PASTE0) in control register 5 are replaced as follows:

- When PASN translation is performed, the PASCE in control register 1 is replaced by the ASCE-p obtained as a result of PASN translation, except that it is replaced by ASCE-rp if a subspace-replacement operation was performed on ASCE-p. Also, the PASTE0 in control register 5 is replaced by PASTE0-p.

The PASTEO-p is placed in bit positions 33-57 of control register 5, and zeros are placed in bit positions 32 and 58-63. Bits 0-31 of the register remain unchanged.

- When PASN translation is not performed, the contents of control registers 1 and 5 remain unchanged.

The secondary address-space-control element (SASCE) in control register 7 is replaced as follows:

- When SASN-d equals PASN-d, by the new contents of control register 1, the PASCE. The new contents may be PASCE-old, ASCE-p, or ASCE-rp.
- When SASN translation is performed, by ASCE-s, or by ASCE-rs if a subspace-replacement operation was performed on ASCE-s.

When SASN-d does not equal PASN-d and SASN translation is not performed, the SASCE remains unchanged.

Other Condition-Code Settings

When PASN translation is called for and cannot be completed because bit 0 is one in either the ASN-first-table entry or the ASTE, or if it can be completed but (1) ASN-and-LX reuse is enabled and PASTEIN-d does not equal the ASTEIN in the ASTE or (2) a subspace-replacement-exception condition exists due to bit 0 or the ASTE sequence number in the subspace ASTE during a subspace-replacement operation on the ASCE-p, condition code 1 is set, and the control registers are not changed.

When PASN translation is called for and completed and any required PASTEIN-d comparison and subspace-replacement operations on the ASCE-p are also completed, and then either (1) the current primary space-switch-event-control bit, bit 57 of control register 1, is one or (2) the space-switch-event-control bit in the ASTE designated by PASTEO-p is one, condition code 3 is set, and the control registers are not changed.

When SASN translation is called for and the translation cannot be completed because bit 0 is one in either the ASN-first-table entry or the ASTE, or if it can be completed but (1) ASN-and-LX reuse is enabled and SASTEIN-d does not equal the ASTEIN in the ASTE, (2) a subspace-replacement-exception

condition exists due to bit 0 or the ASTE sequence number in the subspace ASTE during a subspace-replacement operation on the ASCE-s, or (3) SASN authorization is called for and the SASN is not authorized, or condition code 2 is set, and the control registers are not changed.

Special Conditions

The instruction can be executed only when the ASN-translation control, bit 44 of control register 14, is one. If the ASN-translation-control bit is zero, a special-operation exception is recognized.

The first operand must be designated on a double-word boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

Figure 10-18 on page 10-42 and Figure 10-16 on page 10-40 summarize the functions of the instruction and the priority of recognition of exceptions and condition codes.

Resulting Condition Code:

- 0 Translation and authorization complete; parameters loaded
- 1 Primary ASN or subspace not available; parameters not loaded
- 2 Secondary ASN not available or not authorized, or secondary subspace not available; parameters not loaded
- 3 Space-switch event specified; parameters not loaded

Program Exceptions:

- Access (fetch, operand 1)
- Addressing (ASN-first-table entry, ASN-second-table entry, authority-table entry, dispatchable-unit control table)
- Privileged operation
- Special operation
- Specification

Programming Notes:

1. Bits 61 and 63 in the second-operand address are intended primarily to provide improved performance for those cases where the associated action is unnecessary.

When bit 61 is set to zero, the action of the instruction is based on the assumption that the current values for PASCE-old, PASTEO-old, and AX-old are consistent with PASN-old and that SASCE-old is consistent with SASN-old. When this is not the case, bit 61 should be set to one.

Bit 63, when one, eliminates the SASN-authorization test. The program may be able to determine in certain cases that the SASN is authorized, either because of prior use or because the AX being loaded is authorized to access all address spaces.

2. The SASN-translation and SASN-authorization steps are not performed when SASN-d is equal to PASN-d. This is consistent with the action in SET SECONDARY ASN to current primary (SSAR-cp), which does not perform the translation or ASN authorization.
3. The storage-operand references for LOAD ADDRESS SPACE PARAMETERS may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)
4. See Figure 10-17 on page 10-41 for a listing of abbreviations used in this instruction description.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second and third instruction halfwords.
7.B.1	Privileged-operation exception.
7.B.2	Special-operation exception due to the ASN-translation control, bit 44 of control register 14, being zero.
8.	Specification exception.
9.	Access exceptions for the first operand.
10.	PASN translation and related processing (when performed).
10.1	Addressing exception for access to ASN-first-table entry.
10.2	Condition code 1 due to I bit (bit 0) in ASN-first-table entry being one.
10.3	Addressing exception for access to ASN-second-table entry.
10.4	Condition code 1 due to (1) I bit (bit 0) in ASN-second-table entry (ASTE) being one or (2) ASN-and-LX reuse enabled and primary ASTE instance number (PASTEIN) in first operand not being equal to ASTEIN in ASTE.
10.5	Addressing exception for access to dispatchable-unit control table.
10.6	Addressing exception for access to subspace ASN-second-table entry.
10.7	Condition code 1 due to I bit (bit 0) in subspace ASN-second-table entry being one.
10.8	Condition code 1 due to subspace ASN-second-table-entry sequence number (SSASTESN) in dispatchable-unit control table not being equal to ASTESN in subspace ASN-second-table entry.
10.9	Condition code 3 due to either the old or new space-switch-event-control bit being one.
11.	SASN translation and related processing (when performed).
11.1	Addressing exception for access to ASN-first-table entry.
11.2	Condition code 2 due to I bit (bit 0) in ASN-first-table entry being one.
11.3	Addressing exception for access to ASN-second-table entry.
11.4	Condition code 2 due to (1) I bit (bit 0) in ASN-second-table entry (ASTE) being one or (2) ASN-and-LX reuse enabled and secondary ASTE instance number (SASTEIN) in first operand not being equal to ASTEIN in ASTE.
12.A	Execution of secondary authorization (when performed).
12.A.1	Condition code 2 due to authority-table entry being outside table.
12.A.2	Addressing exception for access to authority-table entry.
12.A.3	Condition code 2 due to S bit in authority-table entry being zero.
12.B.1	Addressing exception for access to dispatchable-unit control table.
12.B.2	Addressing exception for access to subspace ASN-second-table entry.
12.B.3	Condition code 2 due to I bit (bit 0) in subspace ASN-second-table entry being one.
12.B.4	Condition code 2 due to subspace ASN-second-table-entry sequence number (SSASTESN) in dispatchable-unit control table not being equal to ASTESN in subspace ASN-second-table entry.

Figure 10-16. Priority of Execution: LOAD ADDRESS SPACE PARAMETERS

First-Operand Bit Positions when ASN-and-LX Reuse Not Enabled	Abbreviation
0-15	PKM-d
16-31	SASN-d
32-47	AX-d
48-63	PASN-d

First-Operand Bit Positions when ASN-and-LX Reuse Enabled	Abbreviation
0-31	SASTEIN-d
32-47	PKM-d
48-63	SASN-d
64-95	PASTEIN-d
96-111	AX-d
112-127	PASN-d

Control-Register Number.Bit	Abbreviation for	
	Previous Contents	Subsequent Contents
1.0-63	PASCE-old	PASCE-new
3.0-31	SASTEIN-old	SASTEIN-new
3.32-47	PKM-old	PKM-new
3.48-63	SASN-old	SASN-new
4.0-31	PASTEIN-old	PASTEIN-new
4.32-47	AX-old	AX-new
4.48-63	PASN-old	PASN-new
5.33-57	PASTEO-old	PASTEO-new
7.0-63	SASCE-old	SASCE-new

Field in ASN-Second-Table Entry	Abbreviation Used for the Field When Accessed as Part of	
	PASN Translation	SASN Translation
1-29	-	ATO-s
32-47	AX-p	-
48-59	-	ATL-s
64-127	ASCE-p ¹	ASCE-s ¹
352-383	ASTEIN-p	ASTEIN-s
Explanation: - The field is not used in this case. ¹ ASCE-rp is formed from ASCE-p, and ASCE-rs is formed from ASCE-s, by a subspace-replacement operation.		

Figure 10-17. Summary of Abbreviations for LOAD ADDRESS SPACE PARAMETERS

PASN-d Equals PASN- old	Second-Operand- Address Bits ¹		PASN Translation Performed	Result Field						
	61	62		PASCE- new	AX- new	PASTEO- new	PKM- new	SASN- new	PASN- new	PASTEIN- new
Yes	0	0	No	PASCE-old	AX-old	PASTEO-old	PKM-d	SASN-d	PASN-d	PASTEIN-old
Yes	0	1	No	PASCE-old	AX-d	PASTEO-old	PKM-d	SASN-d	PASN-d	PASTEIN-old
Yes	1	0	Yes	ASCE-p ²	AX-p	PASTEO-p	PKM-d	SASN-d	PASN-d	PASTEIN-d
Yes	1	1	Yes	ASCE-p ²	AX-d	PASTEO-p	PKM-d	SASN-d	PASN-d	PASTEIN-d
No	-	0	Yes	ASCE-p ²	AX-p	PASTEO-p	PKM-d	SASN-d	PASN-d	PASTEIN-d
No	-	1	Yes	ASCE-p ²	AX-d	PASTEO-p	PKM-d	SASN-d	PASN-d	PASTEIN-d

Figure 10-18. Summary of Actions: LOAD ADDRESS SPACE PARAMETERS (Part 1 of 2).

SASN-d Equals PASN-d	SASN-d Equals SASN-old	Second-Operand- Address Bits ¹		SASN Translation Performed	SASN Authorization Performed ³	Result Field	
		61	63			SASCE-new	SASTEIN- new
Yes	-	-	-	No	No	PASCE-new	PASTEIN-new
No	Yes	0	1	No	No	SASCE-old	SASTEIN-old
No	Yes	1	1	Yes	No	ASCE-s ⁴	SASTEIN-d
No	Yes	-	0	Yes	Yes	ASCE-s ⁴	SASTEIN-d
No	No	-	1	Yes	No	ASCE-s ⁴	SASTEIN-d
No	No	-	0	Yes	Yes	ASCE-s ⁴	SASTEIN-d

Explanation:

- Action in this case is the same regardless of the outcome of this comparison or of the setting of this bit.
- ¹ Second-operand-address bits:
61 Force ASN translation.
62 Use AX from first operand.
63 Skip secondary authority test.
- ² PASCE-new is ASCE-rp (a copy of ASCE-p except with bits 0-55 and 58-63 replaced from the ASCE in the subspace ASTE), if subspace replacement is performed.
- ³ SASN authorization is performed using ATO-s, ATL-s, and AX-new.
- ⁴ SASCE-new is ASCE-rs (a copy of ASCE-s except with bits 0-55 and 58-63 replaced from the ASCE in the subspace ASTE), if subspace replacement is performed.

Figure 10-18. Summary of Actions: LOAD ADDRESS SPACE PARAMETERS (Part 2 of 2).

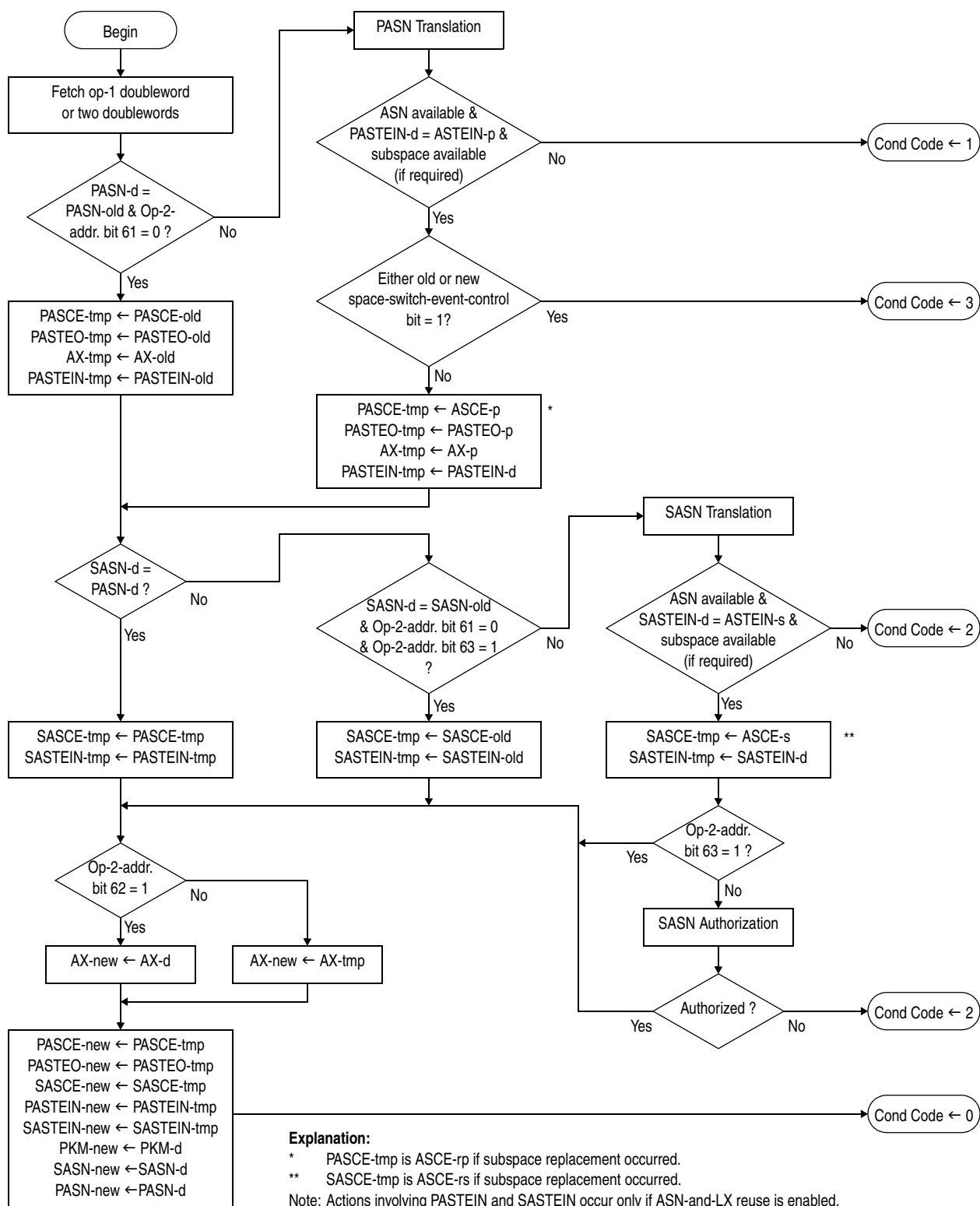
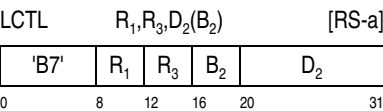


Figure 10-19. Execution of LOAD ADDRESS SPACE PARAMETERS

LOAD CONTROL



Bit positions of the set of control registers starting with control register R_1 and ending with control register R_3 are loaded from storage beginning at the location designated by the second-operand address and continuing through as many locations as needed.

For LOAD CONTROL (LCTL), bit positions 32-63 of the control registers are loaded from successive words beginning at the second-operand address, and bits 0-31 of the registers remain unchanged. For LOAD CONTROL (LCTLG), bit positions 0-63 of the control registers are loaded from successive doublewords beginning at the second-operand address. The control registers are loaded in ascending order of their register numbers, starting with control register R_1 and continuing up to and including control register R_3 , with control register 0 following control register 15.

The information loaded into the control registers becomes active when instruction execution has ended.

The displacement for LCTL is treated as a 12-bit unsigned binary integer. The displacement for LCTLG is treated as a 20-bit signed binary integer.

Special Conditions

The second operand must be designated on a word boundary for LCTL or on a doubleword boundary for LCTLG; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Privileged operation
- Specification

Programming Notes:

1. To ensure that existing programs operate correctly if and when new facilities using additional control-register positions are defined, only zeros should be loaded in unassigned control-register positions.
2. Loading of control registers on some models may require a significant amount of time. This is particularly true for changes in significant parameters.

For example, the TLB may be cleared of entries as a result of changing or enabling the program-event-recording parameters in control registers 9-11. Where possible, the program should avoid unnecessary loading of control registers. In loading control registers 9-11, most models attempt to optimize for the case when the bits of control register 9 are zeros.

LOAD PAGE TABLE ENTRY ADDRESS



General register R_2 contains a virtual address that is processed by means of dynamic address translation to locate a page-table entry, the 64-bit real or absolute address of which is returned in general register R_1 . The M_4 field contains a value designating the effective address-space-control element (ASCE) that is used by DAT, as shown in the following table:

M_4 Field (binary)	Effective Address-Space-Control Element Used by DAT
0000	Contents of control register 1
0001	The address-space-control element obtained by applying the access-register-translation (ART) process to the access register designated by the R_2 field.
0010	Contents of control register 7
0011	Contents of control register 13
0100	The address-space-control element corresponding to the current address-space-control bits, bits 16 and 17 of the PSW.

When the M_4 field contains 0001 binary, or when the M_4 field contains 0100 binary and bits 16-17 of the PSW contain 01 binary, then access-register translation precedes, as is normal, the dynamic address translation.

The R_3 field is ignored but should be zero to permit possible future extensions.

The virtual address specified by the R_2 field is translated by means of dynamic-address translation, regardless of whether DAT is on or off. ART and DAT may be performed with the use of the ART-lookaside buffer (ALB) and translation-lookaside buffer (TLB), respectively. The DAT process completes upon locating the page-table entry. The contents of the page-table entry are not examined for format errors, the I bit is not examined to determine whether the page is invalid, and the page-frame real address is not fetched.

Virtual address computation is performed according to the addressing mode specified by bits 31 and 32 of the current PSW.

The addresses of the region-table entry or entries, if used, and of the segment-table entry and page-table entry are treated as 64-bit addresses, regardless of the current addressing mode. It is unpredictable whether the addresses of these entries are treated as real or absolute addresses.

If the DAT process successfully locates the page-table entry, the 64-bit real or absolute address of the entry is placed in general register R_1 , and the condition code is set as follows:

- When enhanced DAT does not apply, the condition code is set based on the DAT-protection bit (P) of the segment-table entry used in the translation. Condition code 0 is set when the P bit is zero; condition code 1 is set when the P bit is one.
- When enhanced DAT applies, the condition code is set based on all of the DAT-protection bits encountered in the translation. If the DAT-protec-

tion bit is zero in the segment-table entry and the DAT-protection bits are zero in any and all region-table entries used in the translation, condition code 0 is set. If the DAT-protection bit is one in any region- or segment-table entry used in the translation, condition code 1 is set.

When enhanced DAT applies and the DAT process successfully locates a valid segment-table entry in which the STE-format control is one, bits 0-60 of the segment-table entry address are placed in bit positions 0-60 of general register R_1 , bit 61 of the register is set to the logical OR of the DAT-protection bit in the segment-table entry and any region-table entries used in the translation, bits 62-63 of the register are set to zeros, and the instruction completes by setting condition code 2.

When a condition exists that would normally cause one of the exceptions shown in Figure 10-20, processing is as follows:

1. General register R_1 is updated as follows:
 - a. Bits 0-60 of the table-entry address are placed in bit positions 0-60 of the register.
 - b. When enhanced DAT does not apply, bit 61 of the register is set to zero. When enhanced DAT applies, bit 61 is set to the logical OR of the DAT-protection bits in all valid region-table entries used in the translation; if the DAT process does not require region-table entries, or if no valid region-table entries are encountered, bit 61 is set to zero.
 - c. The expected table-type bits are placed in bit positions 62-63 of the register. When the region table or segment table containing the invalid entry is directly designated by the address-space-control element, then the expected table-type bits are those contained in the designation-type control (DT) in the ASCE. Otherwise, the expected table-type bits are one less than the value of those bits in the next-higher-level table.

2. The instruction is completed by setting condition code 2 .

Exception Name	Cause	Code (hex)	Expected TT Bits
Region first translation	Invalid bit is one in the region-first-table entry selected by the RFX portion of the virtual address.	0039	11
Region second translation	Invalid bit is one in the region-second-table entry selected by the RSX portion of the virtual address.	003A	10
Region third translation	Invalid bit is one in the region-third-table entry selected by the RTX portion of the virtual address.	003B	01
Segment translation	Invalid bit is one in the segment-table entry selected by the SX portion of the virtual address.	0010	00

Figure 10-20. LPTEA Exception Conditions Causing CC2

When a condition exists that would normally cause one of the exceptions shown in Figure 10-21, (1) the interruption code assigned to the exception is placed in bit positions 48-63 of general register R₁, and bits 0-47 of the register are set to zeros; and (2) the instruction is completed by setting condition code 3.

Exception Name	Cause	Code (hex)
ALET specification	ALET bits 0-6 not all zeros.	0028
ALEN translation	ALE outside list or I bit is one.	0029
ALE sequence	ALESN in ALET not equal to ALESN in ALE.	002A
ASTE validity	ASTE I bit is one.	002B
ASTE sequence	ASTESN in ALE not equal to ASTESN in ASTE.	002C
Extended authority	ALE P bit not zero, ALEAX not equal to EAX, and secondary bit selected by EAX either outside authority table or zero.	002D

Figure 10-21. LPTEA Exception Conditions Causing CC3

Exception Name	Cause	Code (hex)
ASCE type	ASCE is a region-second-table designation, and bits 0-10 of virtual address not all zeros; ASCE is a region-third-table designation, and bits 0-21 of virtual address not all zeros; or ASCE is a segment-table designation, and bits 0-32 of virtual address not all zeros.	0038
Region first translation	Region-first-table entry selected by RFX portion of virtual address outside table.	0039
Region second translation	Region-second-table entry selected by RSX portion of virtual address outside table.	003A
Region third translation	Region-third-table entry selected by RTX portion of virtual address outside table.	003B
Segment translation	Segment-table entry selected by SX portion of virtual address outside table.	0010

Figure 10-21. LPTEA Exception Conditions Causing CC3

Special Conditions

If the M₄ field contains any value other than 0000-0100 binary, a specification exception is recognized.

The address-space-control element used in the translation must not be a real-space designation; otherwise, a special-operation exception is recognized. The exception due to the address-space-control element has priority 9 in Figure 6-6 on page 6-45.

An addressing exception is recognized when the address used by ART to fetch the effective access-list designation, the access-list entry, the address-space-second-table entry, or the authority-table entry designates a location which is not available in the configuration. An addressing exception is also recognized when the address used by DAT to fetch a region-table entry or the segment-table entry designates a location which is not available in the configuration.

A carry out of bit position 0 as a result of the addition done to compute the address of a region-table entry or the segment-table entry may be ignored or may result in an addressing exception.

A translation-specification exception is recognized if an accessed region-table entry or the segment-table entry has a zero I bit and a format error.

The operation is suppressed on all addressing exceptions.

Resulting Condition Code:

- 0 PTE address returned; STE P bit is 0
- 1 PTE address returned; STE P bit is 1
- 2 Invalid bit is one in the region- or segment-table entry, or enhanced DAT applies, a valid STE was located, and STE FC is 1
- 3 Exception condition exists

Program Exceptions:

- Addressing (effective access-list designation, access-list entry, ASN-second-table entry, authority-table entry, region-table entry, or segment-table entry)
- Operation (DAT-enhancement facility 2 not installed)
- Privileged operation
- Special operation
- Specification
- Translation specification

Programming Notes:

1. An addressing exception is not recognized if the page-table-entry address returned in general register R_1 is not available in the configuration.
2. When enhanced DAT applies, the STE-format control is one, and no other exception conditions apply, the address of the segment-table entry is placed in general register R_1 , and condition code 2 is set. This effectively performs a *load-segment-table-entry-address* function, although no such instruction is defined.

However, because condition code 2 is also used to indicate an invalid region-table entry or segment-table entry, all of the following conditions must exist in order to assume that the located table entry is a valid STE:

- a. Bit positions 62-63 of general register R_1 (the expected table-type) must contain 00 binary.
- b. Bit position 58 of the table entry designated by general register R_1 must contain zero (indicating a valid table entry).

- c. The table-type field in bit positions 60-61 of the table entry designated by general register R_1 must contain 00 binary (designating a segment-table entry).

The following program fragment illustrates this determination. If no branch is taken, then the address in general register 1 is that of a segment-table entry in which the STE-format control is one:

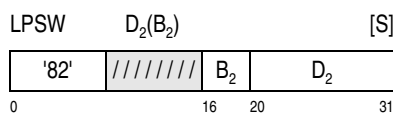
```

LPTEA 1,0,2,4
BRC 8,PTE_WITH_NEITHER_STE_NOR_RTE_PROT
BRC 4,PTE_WITH_PROTECTED_STE_OR_RTE
BRC 1,EXCEPTION_EXISTS
TMLL 1,X'0003'
BRNZ NOT_STE
NILL 1,X'FFF8'
LURAG 10,1
TMLL 10,X'002C'
BRNZ INVALID_OR_MALFORMED_STE

```

3. When condition code 2 is set as a result of locating a valid segment-table entry in which the STE-format control is one, bit 61 of general register R_1 indicates whether DAT protection applies to the segment.

LOAD PSW



The current PSW is replaced by a 16-byte PSW formed from the contents of the doubleword at the location designated by the second-operand address.

Bit 12 of the doubleword must be one; otherwise, a specification exception may be recognized, depending on the model.

Bits 0-32 of the doubleword, except with bit 12 inverted, are placed in bit positions 0-32 of the current PSW. Bits 33-63 of the doubleword are placed in bit positions 97-127 of the current PSW. Bits 33-96 of the current PSW are set to zeros.

A serialization and checkpoint-synchronization function is performed before or after the operand is fetched and again after the operation is completed.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized. A specification exception may be recognized if bit 12 of the operand is zero, depending on the model.

The PSW fields which are to be loaded by the instruction are not checked for validity before they are loaded, except for the optional checking of bit 12. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, when any of the following is true for the newly loaded PSW:

- Any of bits 0, 2-4, 12, or 24-30 is a one.
- Bits 31 and 32 are both zero, and bits 97-103 are not all zeros.
- Bits 31 and 32 are one and zero, respectively.

In these cases, the operation is completed, and the resulting instruction-length code is 0.

The test for a specification exception after the PSW is loaded is described in “Early Exception Recognition” on page 6-8. It may be considered as occurring early in the process of preparing to execute the subsequent instruction.

The operation is suppressed on all addressing and protection exceptions.

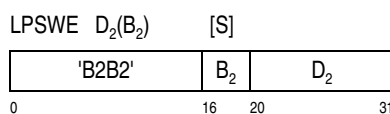
Resulting Condition Code: The code is set as specified in the new PSW loaded.

Program Exceptions:

- Access (fetch, operand 2)
- Privileged operation
- Specification

Programming Note: The second operand should have the format of an ESA/390 PSW. A specification exception will be recognized during or after the execution of LOAD PSW if bit 12 of the operand is zero.

LOAD PSW EXTENDED



The current PSW is replaced by the contents of the 16-byte second operand.

A serialization and checkpoint-synchronization function is performed before or after the operand is fetched and again after the operation is completed.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The value which is to be loaded by the instruction is not checked for validity before it is loaded. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, when any of the following is true for the newly loaded PSW:

- Any of the unassigned bits (0, 2-4, 24-30, or 33-63) is a one.
- Bit 12 is a one.
- Bits 31 and 32 are zero and one, respectively, and bits 64-96 are not all zeros.
- Bits 31 and 32 are both zero, and bits 64-103 are not all zeros.
- Bits 31 and 32 are one and zero, respectively.

In these cases, the operation is completed, and the resulting instruction-length code is zero.

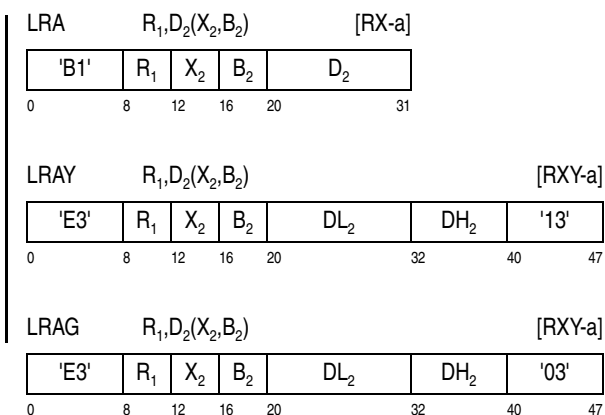
The test for a specification exception after the PSW is loaded is described in “Early Exception Recognition” on page 6-8. It may be considered as occurring early in the process of preparing to execute the subsequent instruction.

The operation is suppressed on all addressing and protection exceptions.

Resulting Condition Code: The code is set as specified in the new PSW loaded.

Program Exceptions:

- Access (fetch, operand 2)
- Privileged operation
- Specification

LOAD REAL ADDRESS

For **LOAD REAL ADDRESS (LRA, LRAY)** in the 24-bit or 31-bit addressing mode, if bits 0-32 of the 64-bit real or absolute address corresponding to the second-operand virtual address are all zeros, bits 32-63 of the real or absolute address are placed in bit positions 32-63 of general register R_1 , and bits 0-31 of the register remain unchanged. If bits 0-32 of the real address or absolute are not all zeros, a special-operation exception is recognized.

For **LRA** or **LRAY** in the 64-bit addressing mode, and for **LOAD REAL ADDRESS (LRAg)** in any addressing mode, the 64-bit real or absolute address corresponding to the second-operand virtual address is placed in general register R_1 .

When the enhanced-DAT facility is not installed, or when the facility is installed but the second operand is translated by means of a segment-table entry in which the STE-format control is zero, the address placed in general register R_1 is real. When the enhanced-DAT facility is installed and the second operand is translated by means of a segment-table entry in which the STE-format control is one, the address placed in general register R_1 is absolute.

The virtual address specified by the X_2 , B_2 , and D_2 fields is translated by means of the dynamic-address-translation facility, regardless of whether DAT is on or off.

The displacement for **LRA** is treated as a 12-bit unsigned binary integer. The displacement for **LRAY** and **LRAg** is treated as a 20-bit signed binary integer.

DAT is performed by using an address-space-control element that depends on the current value of the address-space-control bits, bits 16 and 17 of the PSW, as shown in the following table:

PSW Bits 16 and 17 Address-Space-Control Element Used by DAT

00	Contents of control register 1
10	Contents of control register 7
01	The address-space-control element obtained by applying the access-register-translation (ART) process to the access register designated by the B_2 field
11	Contents of control register 13

ART and DAT may be performed with the use of the ART-lookaside buffer (ALB) and translation-lookaside buffer (TLB), respectively.

The virtual-address computation is performed according to the current addressing mode, specified by bits 31 and 32 of the current PSW.

The addresses of the region-table entry or entries, if used, and of the segment-table entry and page-table entry, if used, are treated as 64-bit addresses regardless of the current addressing mode. It is unpredictable whether the addresses of these entries are treated as real or absolute addresses.

Condition code 0 is set when both ART, if applicable, and DAT can be completed and a special-operation exception is not recognized, that is, when an address-space-control element can be obtained, the entry in each DAT table lies within the table and has a zero I bit, and, for **LRA** or **LRAY** in the 24-bit or 31-bit addressing mode, bits 0-32 of the resulting real or absolute address are zeros. The translated address is not inspected for boundary alignment or for addressing or protection exceptions.

When PSW bits 16 and 17 are 01 binary and an address-space-control element cannot be obtained because of a condition that would normally cause one of the exceptions shown in the following table, (1) the interruption code assigned to the exception is placed in bit positions 48-63 of general register R_1 , bit 32 of this register is set to one, bits 33-47 are set

to zeros, and bits 0-31 remain unchanged, and (2) the instruction is completed by setting condition code 3.

Exception Name	Cause	Code (Hex)
ALET specification	Access-list-entry-token (ALET) bits 0-6 not all zeros	0028
ALEN translation	Access-list entry (ALE) outside list or invalid (bit 0 is one)	0029
ALE sequence	ALE sequence number (ALESN) in ALET not equal to ALESN in ALE	002A
ASTE validity	ASN-second-table entry (ASTE) invalid (bit 0 is one)	002B
ASTE sequence	ASTE sequence number (ASTESN) in ALE not equal to ASTESN in ASTE	002C
Extended authority	ALE private bit not zero, ALE authorization index (ALEAX) not equal to extended authorization index (EAX), and secondary bit selected by EAX either outside authority table or zero	002D

When ART is completed normally, the operation is continued through the performance of DAT.

When the segment-table entry is outside the table and bits 0-32 of the real or absolute address of the entry are all zeros, condition code 3 is set, bits 32-63 of the entry address are placed in bit positions 32-63 of general register R_1 , and bits 0-31 of the register remain unchanged. If bits 0-32 of the address are not all zeros, the result is as shown in the next table below.

For LRA or LRAY in the 64-bit addressing mode or LRA in any addressing mode, when the I bit in the segment-table entry is one, condition code 1 is set, and the 64-bit real or absolute address of the segment-table entry is placed in general register R_1 . In this case except that LRA or LRAY is in the 24-bit or 31-bit addressing mode, if bits 0-32 of the address of the segment-table entry are all zeros, the result is the same except that bits 0-31 of general register R_1 remain unchanged. If bits 0-32 of the address are not all zeros, the result is as shown in the next table below.

For LRA or LRAY in the 64-bit addressing mode or LRA in any addressing mode, when the I bit in the page-table entry is one, condition code 2 is set, and the 64-bit real or absolute address of the page-table

entry is placed in general register R_1 . In this case except that LRA or LRAY is in the 24-bit or 31-bit addressing mode, if bits 0-32 of the address of the page-table entry are all zeros, the result is the same except that bits 0-31 of general register R_1 remain unchanged. If bits 0-32 of the address are not all zeros, the result is as shown in the next table below.

A segment-table-entry or page-table-entry address placed in general register R_1 is real or absolute in accordance with the type of address that was used during the attempted translation.

If a condition exists that would normally cause one of the exceptions shown in the following table, (1) the interruption code assigned to the exception is placed in bit positions 48-63 of general register R_1 , bit 32 of this register is set to one, bits 33-47 are set to zeros, and bits 0-31 remain unchanged, and (2) the instruction is completed by setting condition code 3.

Exception Name	Cause	Code (Hex)
ASCE type	Address-space-control element (ASCE) being used is a region-second-table designation, and bits 0-10 of virtual address not all zeros; ASCE is a region-third-table designation, and bits 0-21 of virtual address not all zeros; or ASCE is a segment-table designation, and bits 0-32 of virtual address not all zeros.	0038
Region first translation	Region-first-table entry selected by region-first-index portion of virtual address outside table or invalid.	0039
Region second translation	Region-second-table entry selected by region-second-index portion of virtual address outside table or invalid.	003A
Region third translation	Region-third-table entry selected by region-third-index portion of virtual address outside table or invalid.	003B
Segment translation	Segment-table entry selected by segment-index portion of virtual address outside table (only when bits 0-32 of entry address not all zeros); or segment-table entry invalid (LRA and LRAY only, and only in 24-bit or 31-bit addressing mode when bits 0-32 of entry address not all zeros).	0010

Exception Name	Cause	Code (Hex)
Page translation	Page-table entry selected by page-index portion of virtual address invalid (LRA and LRAY only, and only in 24-bit or 31-bit addressing mode when bits 0-32 of entry address not all zeros).	0011

Special Conditions

A special-operation exception is recognized when, for LRA or LRAY in the 24-bit or 31-bit addressing mode, bits 0-32 of the resultant 64-bit real address are not all zeros.

An addressing exception is recognized when the address used by ART to fetch the effective access-list designation or the ALE, ASTE, or authority-table entry designates a location which is not available in the configuration or when the address used to fetch the region-table entry or entries, if any, segment-table entry, or page-table entry designates a location which is not available in the configuration.

A translation-specification exception is recognized when an accessed region-table entry or the segment-table entry or page-table entry has a zero I bit and a format error, that is, when any of the reasons listed in "Translation-Specification Exception" on page 6-40 applies.

A carry out of bit position 0 as a result of the addition done to compute the address of a region-table entry or the segment-table entry may be ignored or may result in an addressing exception.

The operation is suppressed on all addressing exceptions.

Resulting Condition Code:

- 0 Translation available
- 1 Segment-table entry invalid (I bit one)
- 2 Page-table entry invalid (I bit one)
- 3 Address-space-control element not available, region-table entry outside table or invalid (I bit one), segment-table entry outside table, or, for LRA and LRAY only, and only in 24-bit or 31-bit addressing mode when bits 0-32 of entry

address not all zeros, segment-table entry or page-table entry invalid (I bit one)

Program Exceptions:

- Addressing (effective access-list designation, access-list entry, ASN-second-table entry, authority-table entry, region-table entry, segment-table entry, or page-table entry)
- Operation (LRAY, if the long-displacement facility is not installed)
- Privileged operation
- Special operation (LRA, LRAY only)
- Translation specification

Programming Notes:

1. Caution must be exercised in the use of LOAD REAL ADDRESS in a multiprocessing configuration. Since INVALIDATE DAT TABLE ENTRY or INVALIDATE PAGE TABLE ENTRY may set I bits in storage to one before causing the corresponding entries in TLBs of other CPUs to be cleared, the simultaneous execution of LOAD REAL ADDRESS on this CPU and either INVALIDATE DAT TABLE ENTRY or INVALIDATE PAGE TABLE ENTRY on another CPU may produce inconsistent results. Because LOAD REAL ADDRESS may access the tables in storage, the segment-table entry or page-table entry may appear to be invalid (condition code 1 or 2, respectively) even though the corresponding TLB entry has not yet been cleared, and the TLB entry may remain in the TLB until the completion of INVALIDATE DAT TABLE ENTRY or INVALIDATE PAGE TABLE ENTRY on the other CPU. There is no guaranteed limit to the number of instructions which may be executed between the completion of LOAD REAL ADDRESS and the TLB being cleared of the entry.

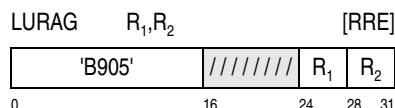
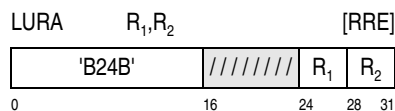
The above cautions for using LOAD REAL ADDRESS also apply when COMPARE AND SWAP AND PURGE is used to explicitly set the invalid bit in a DAT-table entry.

2. Figure 10-22 on page 10-52 summarizes the resulting contents of general register R₁ and the condition code.

Exception/Cause/ Entry-Address Size or Resultant-Real- Address Size	General Register R ₁ Contents and Condition Code									
	LRA or LRAY in 24-Bit or 31-Bit Addressing Mode					LRA or LRAY in 64-Bit Addr. Mode or LRAg in Any Addressing Mode				
	0-31	32	33-47	48-63	CC	0-31	32	33-47	48-63	CC
ALET specification	U	1	0s	0028	3	U	1	0s	0028	3
ALEN translation	U	1	0s	0029	3	U	1	0s	0029	3
ALE sequence	U	1	0s	002A	3	U	1	0s	002A	3
ASTE validity	U	1	0s	002B	3	U	1	0s	002B	3
ASTE sequence	U	1	0s	002C	3	U	1	0s	002C	3
Extended authority	U	1	0s	002D	3	U	1	0s	002D	3
ASCE type	U	1	0s	0038	3	U	1	0s	0038	3
Region first trans.	U	1	0s	0039	3	U	1	0s	0039	3
Region second trans.	U	1	0s	003A	3	U	1	0s	003A	3
Region third trans.	U	1	0s	003B	3	U	1	0s	003B	3
Segment translation/ entry outside table/ entry address < 2GB	U	0	EA3	EA4	3	U	0	EA3	EA4	3
Segment translation/ entry outside table/ entry address >= 2GB	U	1	0s	0010	3	U	1	0s	0010	3
Segment translation/ I bit one/ entry address < 2GB	U	0	EA3	EA4	1	EA1	EA2	EA3	EA4	1
Segment translation/ I bit one/ entry address >= 2GB	U	1	0s	0010	3	EA1	EA2	EA3	EA4	1
Page translation/ I bit one/ entry address < 2GB	U	0	EA3	EA4	2	EA1	EA2	EA3	EA4	2
Page translation/ I bit one/ entry address >= 2GB	U	1	0s	0011	3	EA1	EA2	EA3	EA4	2
Real Address < 2GB	U	0	RA3	RA4	0	RA1	RA2	RA3	RA4	0
Real Address >= 2GB	Special-Operation Exception					RA1	RA2	RA3	RA4	0
Explanation:										
EA1	Bits 0-31 of the entry address.									
EA2	Bit 32 of the entry address.									
EA3	Bits 33-47 of the entry address.									
EA4	Bits 48-63 of the entry address.									
RA1	Bits 0-31 of the resultant real address.									
RA2	Bit 32 of the resultant real address.									
RA3	Bits 33-47 of the resultant real address.									
RA4	Bits 48-63 of the resultant real address.									
U	Unchanged.									

Figure 10-22. Summary of Results: LOAD REAL ADDRESS

LOAD USING REAL ADDRESS



For LOAD USING REAL ADDRESS (LURA), the word at the real-storage location addressed by the contents of general register R_2 is placed in bit positions 32-63 of general register R_1 , and the contents of bit positions 0-31 remain unchanged. For LOAD USING REAL ADDRESS (LURAG), the doubleword at that real-storage location is placed in bit positions 0-63 of general register R_1 .

In the 24-bit addressing mode, bits 40-63 of general register R_2 designate the real-storage location, and bits 0-39 of the register are ignored. In the 31-bit addressing mode, bits 33-63 of general register R_2 designate the real-storage location, and bits 0-32 of the register are ignored. In the 64-bit addressing mode, bits 0-63 of general register R_2 designate the real-storage location.

Because it is a real address, the address designating the storage word or doubleword is not subject to dynamic address translation.

Special Conditions

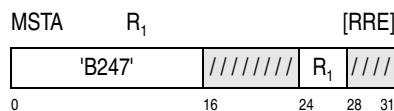
The contents of general register R_2 must designate a location on a word boundary for LURA or on a doubleword boundary for LURAG; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (address specified by general register R_2)
- Privileged operation
- Protection (fetch, operand 2, key-controlled protection)
- Specification

MODIFY STACKED STATE



The contents of bit positions 32-63 of the pair of general registers designated by the R_1 field are placed in the modifiable area, byte positions 152-159, of the last state entry in the linkage stack.

The R_1 field designates the even-numbered register of an even-odd pair of general registers.

The last state entry is located as described in “Unstacking Process” on page 5-82. The state entry remains in the linkage stack, and the linkage-stack-entry address in control register 15 remains unchanged.

Key-controlled protection does not apply to the references to the linkage stack, but low-address and DAT protection do apply.

Special Conditions

A specification exception is recognized when R_1 is odd.

The CPU must be in the primary-space mode, access-register mode, or home-space mode; otherwise, a special-operation exception is recognized.

A stack-empty, stack-specification, or stack-type exception may be recognized during the unstacking process.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-23 on page 10-54.

Condition Code: The code remains unchanged.

Program Exceptions:

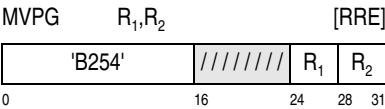
- Access (fetch and store, except for key-controlled protection, linkage-stack entry)
- Special operation
- Specification
- Stack empty

- Stack specification
- Stack type

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off or the CPU being in secondary-space mode.
8.A	Specification exception due to R_1 being odd.
8.B.1	Access exceptions (fetch) for entry descriptor of the current linkage-stack entry.
8.B.2	Stack-type exception due to current entry not being a state entry or header entry. Note: Exceptions 8.B.3-8.B.7 can occur only if the current entry is a header entry.
8.B.3	Access exceptions (fetch) for second word of the header entry.
8.B.4	Stack-empty exception due to backward stack-entry validity bit in the header entry being zero.
8.B.5	Access exceptions (fetch) for entry descriptor of preceding entry, which is the entry designated by the backward stack-entry address in the current (header) entry.
8.B.6	Stack-specification exception due to preceding entry being a header entry.
8.B.7	Stack-type exception due to preceding entry not being a state entry.
8.B.8	Access exceptions (store) for the modifiable area of the state entry.

Figure 10-23. Priority of Execution: MODIFY STACKED STATE

MOVE PAGE



The first operand is replaced by the second operand. The first and second operands both are 4K bytes on 4K-byte boundaries. The results are indicated in the condition code. The accesses to the first-operand location or the second-operand location, but not to both locations, may be performed by using the key specified in general register 0; otherwise, the accesses to an operand location are performed by using the PSW key.

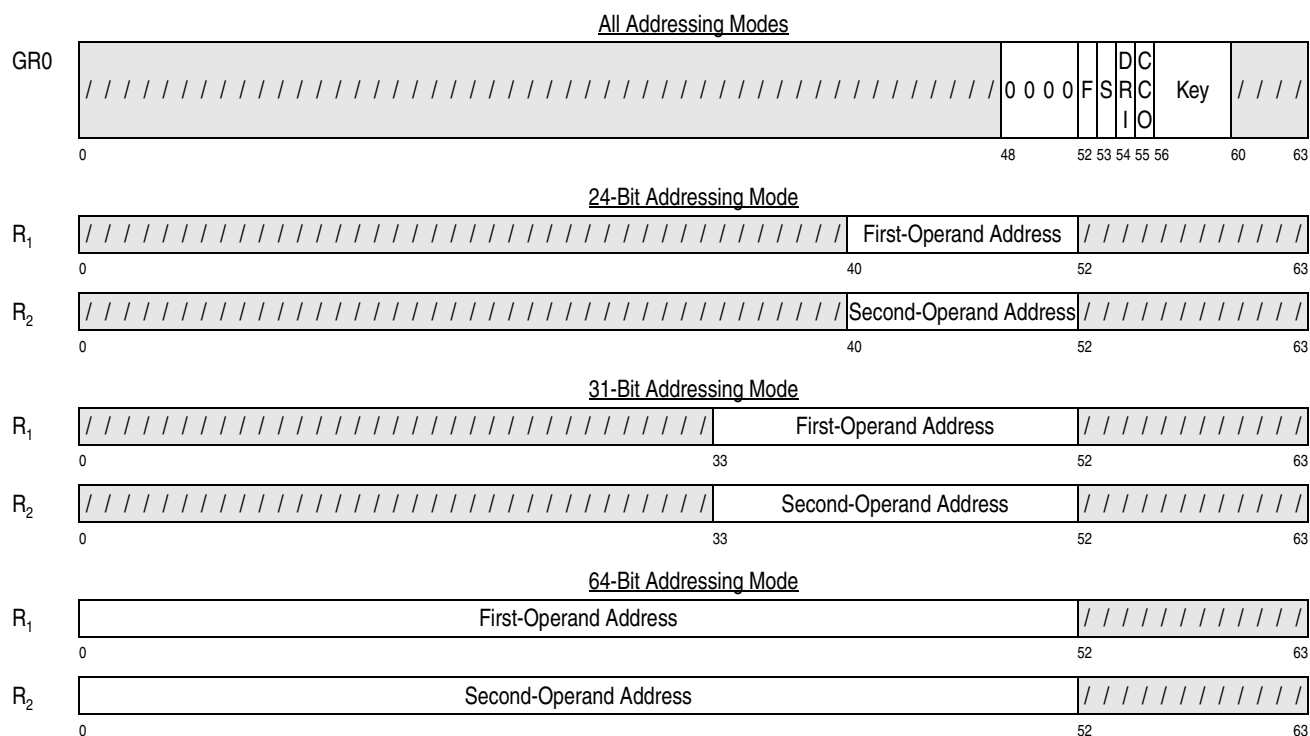
The location of the leftmost byte of the first operand and second operand is designated by the contents of general registers R_1 and R_2 , respectively.

The handling of the addresses in general registers R_1 and R_2 depends on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-51 of a general register, with 12 rightmost zeros

appended, are the address, and bits 0-39 and 52-63 in the register are ignored. In the 31-bit addressing mode, the contents of bit positions 33-51 of a general register, with 12 rightmost zeros appended, are the address, and bits 0-32 and 52-63 in the register are ignored. In the 64-bit addressing mode, the contents of bit positions 0-51 of a general register, with 12 rightmost zeros appended, are the address, and bits 52-63 in the register are ignored.

Bits 56-59 of general register 0 are used as the specified access key. Bit 52 of general register 0, when one, specifies that the specified access key is to be used for accessing the first operand, and bit 53 specifies the same for the second operand. A specification exception is recognized if bits 52 and 53 are both ones. Bit 54 of general register 0 is a destination-reference-intention bit, and bit 55 is a condition-code-option bit. Bits 48-51 of general register 0 must be zeros; otherwise, a specification exception is recognized. Bits 0-47 and 60-63 of general register 0 are ignored.

The contents of the registers just described are shown in Figure 10-24 on page 10-55.

**Explanation:**

CCO Condition-code-option bit.

DRI Destination-reference-intention bit.

F When one, specified access key applies to first operand. A specification exception is recognized if F and S both are one.

Key Specified access key.

S When one, specified access key applies to second operand. A specification exception is recognized if F and S both are one.

Figure 10-24. Register Contents for MOVE PAGE

When bit 52 of general register 0 is one, the fetch accesses to the second-operand location are performed by using the PSW key, and the store accesses to the first-operand location are performed by using the key specified in general register 0. When bit 53 of general register 0 is one, the fetch accesses to the second-operand location are performed by using the key specified in general register 0, and the store accesses to the first-operand location are performed by using the PSW key. When bits 52 and 53 are both zeros, the PSW key is used for accessing both operands.

When 4K bytes have been moved, condition code 0 is set.

When a page-translation-exception condition exists, the exception is not recognized if the condition-code-option bit, bit 55 in general register 0, is one; instead,

condition code 1 or 2 is set. Condition code 1 is set if a page-translation-exception condition exists for the first operand and not for the second operand. Condition code 2 is set if a page-translation-exception condition exists for the second operand, regardless of whether the condition exists for the first operand.

When an access exception can be recognized for both operands, it is unpredictable for which operand an exception is recognized. If one of the exceptions is a page-translation exception that would cause condition code 1 or 2 to be set, it is unpredictable whether the access exception for the other operand is recognized or condition code 1 or 2 is set.

The references to main storage are not necessarily single-access references and are not necessarily performed in a left-to-right direction, as observed by other CPUs and by channel programs.

Special Conditions

In the problem state, when either bit 52 or bit 53 in general register 0 is one, the operation is performed only if the access key specified in general register 0 is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the specified access key is valid. When bits 52 and 53 are both zeros, the access key in general register 0 is not tested for validity.

In the problem state, when bits 52 and 53 in general register 0 are both ones and the access key in general register 0 is not permitted by the PSW-key mask, it is unpredictable whether a specification exception or a privileged-operation exception is recognized.

Resulting Condition Code:

- 0 Data moved
- 1 Condition-code-option bit one, page-table entry for first operand invalid, and page-table entry for second operand valid
- 2 Condition-code-option bit one and page-table entry for second operand invalid
- 3 --

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Privileged operation (access key specified, and selected PSW-key-mask bit is zero in the problem state)
- Specification

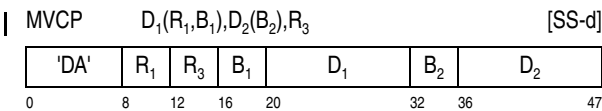
Programming Notes:

1. MOVE PAGE, or a loop of MOVE PAGE instructions that moves multiple pages, may provide, on most models, better performance than a MOVE LONG instruction or a loop of MOVE (MVC) instructions that performs the same function. Whether MOVE PAGE provides better performance depends on control-program specifications and the method by which the control program handles page-translation exceptions.
2. The destination-reference-intention bit should be set to one when there is an intention to reference the first operand by means of an instruction other than MOVE PAGE. The bit may allow the control

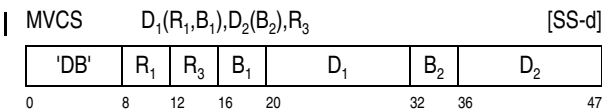
program to process a page-translation exception more efficiently.

3. The condition-code-option bit provides compatibility with the MOVE PAGE instruction of the ESA/390 move-page facility 1. The bit is for use by the MVS/ESA HSPSERV macro expansion.
4. The condition code set by the instruction normally need not be examined if the condition-code-option bit is zero or if DAT is off.
5. See the definitions of real locations 162 and 168-175 under "Assigned Storage Locations" in Chapter 3, "Storage" for a description of information stored during a program interruption due to a DAT-related translation exception recognized by MOVE PAGE.

MOVE TO PRIMARY



MOVE TO SECONDARY



The first operand is replaced by the second operand. One operand is in the primary address space, and the other is in the secondary address space. The accesses to the operand in the primary space are performed by using the PSW key; the accesses to the operand in the secondary space are performed by using the key specified by the third operand.

The addresses of the first and second operands are virtual, one operand address being translated by means of the primary address-space-control element and the other by means of the secondary address-space-control element. Operand-address translation is performed in the same way when the address-space-control bits in the current PSW specify either the primary-space mode or the secondary-space mode.

For MOVE TO PRIMARY, movement is to the primary space from the secondary space. The first-operand address is translated by using the primary address-

space-control element, and the second-operand address is translated by using the secondary address-space-control element.

For MOVE TO SECONDARY, movement is to the secondary space from the primary space. The first-operand address is translated by using the secondary address-space-control element, and the second-operand address is translated by using the primary address-space-control element.

Bit positions 56-59 of general register R_3 are used as the secondary-space access key. Bit positions 0-55 and 60-63 of the register are ignored.

General register R_1 contains an unsigned binary integer called the true length. In the 24-bit or 31-bit addressing mode, the true length is in bit positions 32-63 of the register, and the contents of bit positions 0-31 of the register are ignored. In the 64-bit addressing mode, the true length is in bit positions 0-63 of the register.

The contents of the general registers just described are shown in Figure 10-25 on page 10-57.

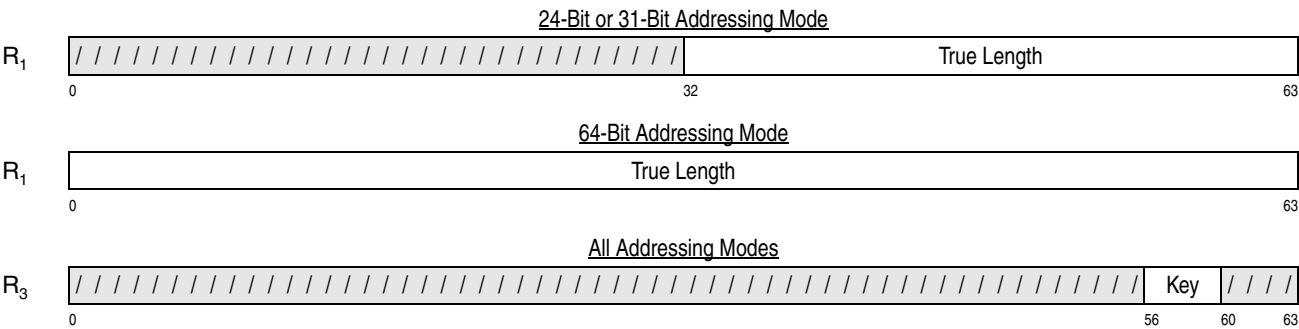


Figure 10-25. Register Contents of MOVE TO PRIMARY and MOVE TO SECONDARY

The first and second operands are the same length, called the effective length. The effective length is equal to the true length or 256, whichever is less. Access exceptions for the first and second operands are recognized only for that portion of the operand within the effective length. When the effective length is zero, no access exceptions are recognized for the first and second operands, and no movement takes place.

Each storage operand is processed left to right. The storage-operand-consistency rules are the same as for MOVE (MVC), except that when the operands overlap in real storage, the use of the common real-storage locations is not necessarily recognized.

As part of the execution of the instruction, the value of the true length is used to set the condition code. If the true length is 256 or less, including zero, the true length and effective length are equal, and condition code 0 is set. If the true length is greater than 256, the effective length is 256, and condition code 3 is set.

For both MOVE TO PRIMARY and MOVE TO SECONDARY, a serialization and checkpoint-synchroni-

zation function is performed before the operation begins and again after the operation is completed.

Special Conditions

Since the secondary space is accessed, the operation is performed only when the secondary-space control, bit 37 of control register 0, is one and DAT is on. When either the secondary-space control is zero or DAT is off, a special-operation exception is recognized. A special-operation exception is also recognized when the address-space-control bits in the current PSW specify the access-register or home-space mode.

In the problem state, the operation is performed only if the secondary-space access key is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the secondary-space access key is valid.

The priority of the recognition of exceptions and condition codes is shown in Figure 10-26 on page 10-58.

Resulting Condition Code:

- 0 True length less than or equal to 256
- 1 --
- 2 --
- 3 True length greater than 256

Program Exceptions:

- Access (fetch, primary virtual address, operand 2, MVCS; fetch, secondary virtual address, operand 2, MVCP; store, secondary virtual address, operand 1, MVCS; store, primary virtual address, operand 1, MVCP)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)
- Special operation

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second and third instruction halfwords.
7.B	Special-operation exception due to the secondary-space control, bit 37 of control register 0, being zero, to DAT being off, or to the CPU being in the access-register or home-space mode.
8.	Privileged-operation exception due to selected PSW-key-mask bit being zero in the problem state.
9.	Completion due to length zero.
10.	Access exceptions for operands.

Figure 10-26. Priority of Execution: MOVE TO PRIMARY and MOVE TO SECONDARY

Programming Notes:

- 1. MOVE TO PRIMARY and MOVE TO SECONDARY can be used in a loop to move a variable



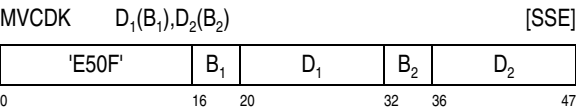
Figure 10-27. Register Contents of MOVE WITH DESTINATION KEY

L specifies the number of bytes to the right of the first byte of each operand. Therefore, the length in bytes of each operand is 1-256, corresponding to a length code in L of 0-255.

number of bytes of any length. See the programming note under MOVE WITH KEY.

- 2. MOVE TO PRIMARY and MOVE TO SECONDARY should be used only when movement is between different address spaces. The performance of these instructions on most models may be significantly slower than that of MOVE WITH KEY, MOVE (MVC), or MOVE LONG. In addition, the definition of overlapping operands for MOVE TO PRIMARY and MOVE TO SECONDARY is not compatible with the more precise definitions for MOVE (MVC), MOVE WITH KEY, and MOVE LONG.

MOVE WITH DESTINATION KEY



The first operand is replaced by the second operand. The accesses to the destination-operand location are performed by using the key specified in general register 1, and the accesses to the source-operand location are performed by using the PSW key.

The first and second operands are of the same length, which is specified by bits 56-63 of general register 0. Bits 0-55 of general register 0 are ignored.

Bits 56-59 of general register 1 are used as the specified access key. Bits 0-55 and 60-63 of general register 1 are ignored.

The contents of general registers 0 and 1 are shown in Figure 10-27 on page 10-58.

Each of the operands is processed left to right. When the operands overlap destructively in real storage, the results in the first-operand location are unpredictable. Except for this unpredictability in the case of destructive overlap, the storage-operand-consistency rules are the same as for the MOVE (MVC) instruction.

Special Conditions

In the problem state, the operation is performed only if the access key specified in general register 1 is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the specified access key is valid.

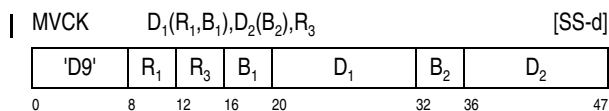
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)

Programming Note: See the programming notes for the MOVE WITH SOURCE KEY instruction.

MOVE WITH KEY



The first operand is replaced by the second operand. The fetch accesses to the second-operand location are performed by using the key specified in the third operand, and the store accesses to the first-operand location are performed by using the PSW key.

Bit positions 56-59 of general register R₃ are used as the source access key. Bit positions 0-55 and 60-63 of the register are ignored.

General register R₁ contains an unsigned binary integer called the true length. In the 24-bit or 31-bit addressing mode, the true length is in bit positions 32-63 of the register, and the contents of bit positions 0-31 of the register are ignored. In the 64-bit addressing mode, the true length is in bit positions 0-63 of the register.

The contents of the general registers just described are shown in Figure 10-28 on page 10-59.

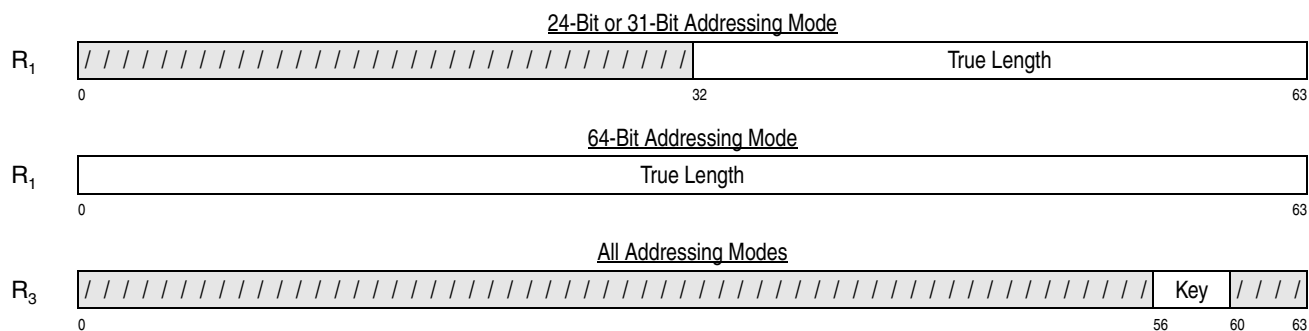


Figure 10-28. Register Contents of MOVE WITH KEY

The first and second operands are of the same length, called the effective length. The effective length is equal to the true length or 256, whichever is less. Access exceptions for the first and second operands are recognized only for that portion of the operand within the effective length. When the effective length is zero, no access exceptions are recognized for the first and second operands, and no movement takes place.

Each storage operand is processed left to right. When the storage operands overlap, the result is

obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte was fetched. The storage-operand-consistency rules are the same as for the MOVE (MVC) instruction.

As part of the execution of the instruction, the value of the true length is used to set the condition code. If the true length is 256 or less, including zero, the true length and effective length are equal, and condition code 0 is set. If the true length is greater than 256,

the effective length is 256, and condition code 3 is set.

Special Conditions

In the problem state, the operation is performed only if the source access key is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the source access key is valid.

The priority of the recognition of exceptions and condition codes is shown in Figure 10-29 on page 10-60.

Resulting Condition Code:

- 0 True length less than or equal to 256
- 1 --
- 2 --
- 3 True length greater than 256

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second and third instruction halfwords.
8.	Privileged-operation exception due to selected PSW-key-mask bit being zero in the problem state.
9.	Completion due to length zero.
10.	Access exceptions for operands.

Figure 10-29. Priority of Execution: MOVE WITH KEY

Programming Notes:

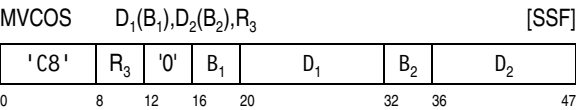
1. MOVE WITH KEY can be used in a loop to move a variable number of bytes of any length, as follows:

```
LOOP    MVCK  D1(R1,B1),D2(B2),R3
        BC    8,END
        AHI   B1,256
        AHI   B2,256
        AHI   R1,-256
        B     LOOP
END      [Any instruction]
```

The above program is for execution in the 24-bit or 31-bit addressing mode. In the 64-bit addressing mode, AGHI instructions should be substituted for the AHI instructions.

2. The performance of MOVE WITH KEY on most models may be significantly slower than that of the MOVE (MVC) and MOVE LONG instructions. Therefore, MOVE WITH KEY should not be used if the keys of the source and the target are the same.

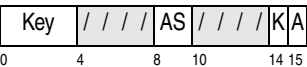
MOVE WITH OPTIONAL SPECIFICATIONS



The first operand is replaced by the second operand. Bits in general register 0 determine the address-space-control modes and protection keys that are used to access the first and second operands.

The addresses of the first and second operands are virtual.

Bits 32-47 of general register 0 contain the operand-access control for the first operand (OAC₁), and bits 48-63 of general register 0 contain the operand-access control for the second operand (OAC₂). The operand-access control has the following format.



Specified-Access Key (Key): Bits positions 0-3 contain the specified-access key that is used to

access the operand if the specified-access-key validity bit (K) is one; otherwise, the specified-access key is ignored.

Specified-Address-Space Control (AS): Bit positions 8-9 contain the address-space control that is used to access the operand when the specified-address-space-control validity bit (A) is one; otherwise, the specified-address-space control is ignored. The meaning of bits 8-9 is identical to that of the address-space control in bits 16-17 of the PSW.

Specified-Access-Key Validity Bit (K): Bit 14 controls whether the PSW key or the specified-access key is used to access the operand. When the K bit is zero, the PSW key is used. When the K bit is one, the specified-access key is used.

Specified-Address-Space-Control Validity Bit (A): Bit 15 controls whether the address-space control in the current PSW or the address-space control in the specified-ASC is used to access the oper-

and. When the A bit is zero, the address-space control in the current PSW is used. When the A bit is one, the specified-address-space control is used.

Bits 0-31 of general register 0 are ignored. Bits 4-7 and 10-14 of both operand-access controls (that is, bits 36-39, 42-45, 52-55, and 58-61 of general register 0) are reserved and should contain zeros; otherwise, the program may not operate compatibly in the future.

General register R_3 contains an unsigned binary integer called the true length. In the 24-bit or 31-bit addressing mode, the true length is in bit positions 32-63 of the register, and the contents of bit positions 0-31 of the register are ignored. In the 64-bit addressing mode, the true length is in bit positions 0-63 of the register.

The contents of the general registers just described are shown in Figure 10-30.

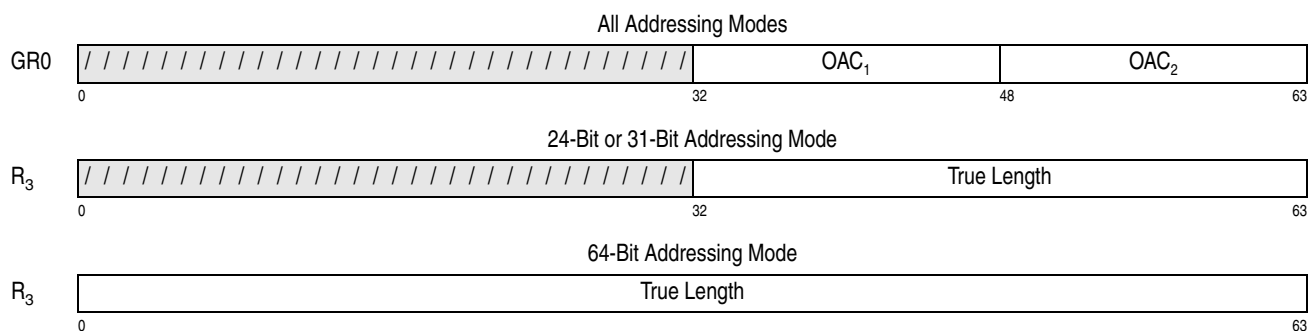


Figure 10-30. Register Contents for MOVE WITH OPTIONAL SPECIFICATIONS

The first and second operands are the same length, called the effective length. The effective length is equal to the true length or 4,096, whichever is less. Access exceptions for the first and second operands are recognized only for that portion of the operand within the effective length. When the effective length is zero, no access exceptions are recognized for the first and second operands, and no movement takes place.

As part of the execution of the instruction, the value of the true length is used to set the condition code. If the true length is 4,096 or less, including zero, then the true length and effective length are equal, and condition code 0 is set. If the true length is greater than 4,096, then the effective length is 4,096, and condition code 3 is set.

No test is made for destructive overlap, and the results in the first-operand location are unpredictable when destructive overlap exists. Operands are said to overlap destructively when the first-operand real location is used as a source after data has been moved into it.

Operands do not overlap destructively if the leftmost byte of the first operand does not coincide with any of the second-operand bytes participating in the operation other than the leftmost byte of the second operand. When an operand wraps around from location $2^{24} - 1$ (or $2^{31} - 1$ or $2^{64} - 1$) to location 0, operand bytes in locations up to and including $2^{24} - 1$ (or $2^{31} - 1$ or $2^{64} - 1$) are considered to be to the left of bytes in locations from 0 up.

In the 24-bit addressing mode, wraparound is from location $2^{24} - 1$ to location 0; in the 31-bit addressing mode, wraparound is from location $2^{31} - 1$ to location 0; and, in the 64-bit addressing mode, wraparound is from location $2^{64} - 1$ to location 0.

Special Conditions

A special-operation exception is recognized for any of the following conditions:

- DAT is off.
- The address-space control in OAC_1 designates the home-space mode, the address-space-control validity bit in OAC_1 is one, and the current PSW is in the problem state (that is, bits 40-41 of general register 0 are 11 binary, bit 47 of the register is one, and bit 15 of the current PSW is one).
- The secondary-space control, bit 37 of control register 0, is zero, and either of the following is true:
 - The A bit of the OAC for either operand is zero, and bits 16-17 of the current PSW are 10 binary.
 - The A bit of the OAC for either operand is one, and the AS bits in the same OAC are 10 binary.

The secondary-space control is not examined for an operand that is accessed in the secondary-space mode as a result of access-register translation (that is, DAT is on, bits 16-17 of the current PSW are 01 binary, and the access register corresponding to the operand's base register contains an ALET of 00000001 hex).

When MOVE WITH OPTIONAL SPECIFICATIONS is executed in the problem state, and either the first or second operand's specified-access key (from GR0) or the implied-access key (from the PSW) is invalid, a privileged-operation exception is recognized. The validity of an access key is determined as follows:

- The specified-access key in OAC_1 is invalid if the specified-access-key control, bit 46 of general register 0, is one; and the specified-access key, bits 32-35 of the register, designates a PSW-key-mask (PKM) bit position in control register 3 that contains zero.

- The specified-access key in OAC_2 is invalid if the specified-access-key control, bit 62 of general register 0, is one; and the specified-access key, bits 48-51 of the register, designates a PKM bit position that contains zero.
- The implied-access key is invalid if both of the following conditions are true:
 - Either or both bits 46 and 62 of general register 0 are zero.
 - The PSW key, bits 8-11 of the PSW, designates a PKM bit position that contains zero.

In the supervisor state, any value for the implied- or specified-access key is valid.

The priority of the recognition of exceptions and condition codes is shown in Figure 10-31.

Resulting Condition Code:

- | | |
|---|---|
| 0 | True length less than or equal to 4,096 |
| 1 | -- |
| 2 | -- |
| 3 | True length greater than 4,096 |

Program Exceptions:

- Access (fetch, second operand; store first operand)
- Operation (MVCOS not installed)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)
- Special operation

1.-5.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
6.1	Access exceptions for first instruction halfword.
6.2.A	Access exceptions for second instruction halfword.
6.2.B	Access exceptions for third instruction halfword.
7.	Operation exception.
8.	Special-operation exception.
9.	Privileged-operation exception.
10.	Completion due to length zero.
11.	Access exceptions for operands.

Figure 10-31. Priority of Execution: MOVE WITH OPTIONAL SPECIFICATIONS

Programming Notes:

1. MOVE WITH OPTIONAL SPECIFICATIONS can be used in a loop to move a variable number of bytes of any length.
2. An example of moving 16 K-bytes to the home address space from the secondary address space is shown below. The first operand is accessed using the PSW key, and the second operand is accessed using key 5.

```

                LA    3,OPERAND1
                LA    5,OPERAND2
                LHI   7,16384
                LLILF 0,X'00C15083'
LOOP           MVCOS 0(3),0(5),7
                AHI   3,4096
                AHI   5,4096
                AHI   7,-4096
                BP    LOOP

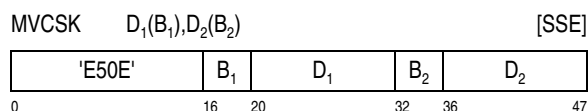
```

In the 64-bit addressing mode, AGHI instructions should be substituted for the AHI instructions.

3. The performance of MOVE WITH OPTIONAL SPECIFICATIONS may be significantly slower than that of individual MOVE (MVC) instructions.
4. When bits 46-47 and 62-63 of general register 0 are all zero, MOVE WITH OPTIONAL SPECIFICATIONS uses the key and address-space control in the current PSW to access both the first and second operands.
5. In the MVCDK, MVCK, MVCP, MVCS, and MVCSK instructions, one operand is accessed using an implied key from the PSW, and the

other operand is accessed using a specified key in a register. In the problem state, only the specified key is checked for validity in the PSW-key mask (PKM).

When MVCOS is executed in the problem state, the access keys for the first and second operands are both checked for validity in the PKM, regardless of whether the keys are implied or specified.

MOVE WITH SOURCE KEY

The first operand is replaced by the second operand. The accesses to the source-operand location are performed by using the key specified in general register 1, and the accesses to the destination-operand location are performed by using the PSW key.

The first and second operands are of the same length, which is specified by bits 56-63 of general register 0. Bits 0-55 of general register 0 are ignored.

Bits 56-59 of general register 1 are used as the specified access key. Bits 0-55 and 60-63 of general register 1 are ignored.

The contents of general registers 0 and 1 are shown in Figure 10-32 on page 10-63.

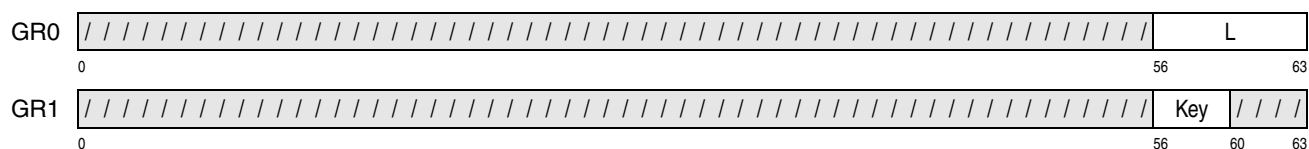


Figure 10-32. Register Contents of MOVE WITH SOURCE KEY

L specifies the number of bytes to the right of the first byte of each operand. Therefore, the length in bytes of each operand is 1-256, corresponding to a length code in L of 0-255.

The fetch accesses to the second-operand location are performed by using the key specified in general register 1, and the store accesses to the first-operand location are performed by using the PSW key.

Each of the operands is processed left to right. When the operands overlap destructively in real storage, the results in the first-operand location are unpredictable. Except for this unpredictability in the case of destructive overlap, the storage-operand-consistency rules are the same as for the MOVE (MVC) instruction.

Special Conditions

In the problem state, the operation is performed only if the access key specified in general register 1 is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the specified access key is valid.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2; store, operand 1)
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)

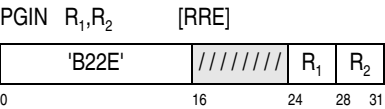
Programming Notes:

1. When data is to be moved alternately in both directions between two storage areas that are fetch protected by means of different keys, then MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY can be used while leaving the PSW key unchanged; and this may be, on most models, significantly faster than using MOVE WITH KEY along with SET PSW KEY FROM ADDRESS to change the PSW key.
2. MOVE WITH SOURCE KEY and MOVE WITH DESTINATION KEY should be used only when movement is between storage areas having different keys. The performance of these instructions on most models may be significantly slower than that of the MOVE (MVC) instruction.
3. MOVE WITH SOURCE KEY or MOVE WITH DESTINATION KEY can be used in a loop to move a variable number of bytes as shown in the following example. In the example, the specified access key, the first-operand address, the second-operand address, and the length of each operand are assumed to be in general registers 1-4, respectively, at the beginning of the example. The length of each operand is treated as a

32-bit signed value, and a negative value is treated as zero.

	LTR	4,4
	BC	12,END
	AHI	4,-256
	BC	12,LAST
	LA	0,255
LOOP	MVCSK	0(2),0(3)
	LA	2,256(2)
	LA	3,256(3)
	AHI	4,-256
	BC	2,LOOP
LAST	LA	0,255(4)
	MVCSK	0(2),0(3)
END		[Any instruction]

PAGE IN



A page-in operation is performed which transfers a 4K-byte block to the real-storage location designated by general register R₁ from the expanded-storage block designated by general register R₂.

Bits 32-63 of general register R₂ are a 32-bit unsigned binary integer called the expanded-storage-block number. This number designates the 4K-byte block of expanded storage which is to be transferred. If the expanded-storage-block number designates an inaccessible block in expanded storage, condition code 3 is set.

The contents of general register R₁ are a real address which designates a 4K-byte block in main storage. In the 24-bit-addressing mode, bits 40-51 designate the block, and bits 0-39 are ignored. In the 31-bit-addressing mode, bits 33-51 designate the block, and bits 0-32 are ignored. In the 64-bit-addressing mode, bits 0-51 designate the block. In all modes, bits 52-63 of the address are ignored.

Because it is a real address, the address designating the main-storage block is not subject to dynamic address translation. PAGE IN is not subject to key-controlled storage protection, but low-address protection does apply. PAGE IN is not subject to program-event recording for storage alteration.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

If the page-in operation is completed with no errors, condition code 0 is set.

If the page-in operation encounters an expanded-storage data error, condition code 1 is set. For an expanded-storage data-error condition, the contents of the entire 4K-byte block in real storage is unpredictable, but this condition does not result in the generation of invalid checking-block codes in real storage.

If the expanded-storage block is not available, that is, the block is not provided or is not currently in the configuration, then condition code 3 is set, and no other action is taken.

Operation of PAGE IN in a Multiple-CPU Configuration

The accesses to main storage and to expanded storage by PAGE IN are not necessarily single-access references and are not necessarily performed in a left-to-right direction, as observed by other CPUs and by channel programs.

See also the description under PAGE OUT.

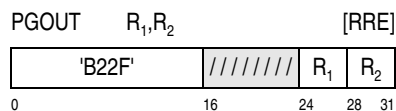
Resulting Condition Code:

- | | |
|---|--------------------------------------|
| 0 | Page-in operation completed |
| 1 | Expanded-storage data error |
| 2 | -- |
| 3 | Expanded-storage block not available |

Program Exceptions:

- Addressing (block designated by general register R₁)
- Operation (if the expanded-storage facility is not installed)
- Privileged operation
- Protection (block designated by general register R₁; low-address protection)

PAGE OUT



A page-out operation is performed which transfers a 4K-byte block from the real-storage location designated by general register R₁ to the expanded-storage block designated by general register R₂.

Bits 32-63 of general register R₂ are a 32-bit unsigned binary integer called the expanded-storage-block number. This number designates the 4K-byte block of expanded storage which is to be replaced. If the expanded-storage-block number designates an inaccessible block in expanded storage, condition code 3 is set.

The contents of general register R₁ are a real address which designates a 4K-byte block in main storage. In the 24-bit-addressing mode, bits 40-51 designate the block, and bits 0-39 are ignored. In the 31-bit-addressing mode, bits 33-51 designate the block, and bits 0-32 is ignored. In the 64-bit-addressing mode, bits 0-51 designate the block. In all modes, bits 52-63 of the address are ignored.

Because it is a real address, the address designating the main-storage block is not subject to dynamic address translation. PAGE OUT is not subject to key-controlled protection.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Depending on the model, after the data has been written to the expanded-storage block, a read-back-check operation may be performed to determine whether the data was written correctly. If the read-back-check operation determines that the data has been written correctly, condition code 0 is set. If the read-back-check operation encounters an expanded-storage data error, condition code 1 is set.

Most models do not perform the read-back-check operation, and, after the page-out operation is completed, condition code 0 is set.

Regardless of whether condition code 0 or condition code 1 is set, the entire 4K-byte block is written. Errors, if any, in the block after the block is written are

preserved. Thus, if a subsequent execution of PAGE IN addresses the same expanded-storage block, the expanded-storage data error will be detected and condition code 1 will be indicated.

If the expanded-storage block is not available, that is, the block is not provided or is not currently in the configuration, then condition code 3 is set, and no other action is taken.

Operation of PAGE OUT in a Multiple-CPU Configuration

The accesses to main storage and to expanded storage by PAGE OUT are not necessarily single-access references and are not necessarily performed in a left-to-right direction, as observed by other CPUs and by channel programs.

If two or more CPUs issue PAGE IN or PAGE OUT instructions at approximately the same instant in time, depending on the model, the operations may be performed one at a time, or the operations may be performed concurrently. Concurrent operation may occur even if the instructions address the same expanded-storage block.

When two or more PAGE OUT instructions addressing the same expanded-storage block are executed concurrently, the resulting values in the expanded-storage block for each group of bytes transferred may be from any of the instructions being executed simultaneously. The number of bytes transferred as a group depends on the model.

Similarly, for concurrent execution of a PAGE IN and a PAGE OUT instruction for the same expanded-storage block, the resulting values for each group of bytes transferred as a result of the execution of the PAGE IN instruction may be either the old or new values from the expanded-storage block.

Concurrent operation of paging instructions does not result in expanded-storage data errors.

Resulting Condition Code:

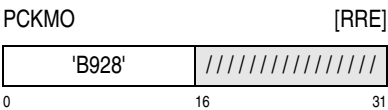
- 0 Page-out operation completed
- 1 Expanded-storage data error
- 2 --
- 3 Expanded-storage block not available

The function codes for PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATION are as

Program Exceptions:

- Addressing (block designated by general register R₁)
- Operation (if the expanded-storage facility is not installed)
- Privileged operation

PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATIONS



A function specified by the function code in general register 0 is performed.

Bits 16-31 of the instruction are ignored.

Bit positions 57-63 of general register 0 contain the function code. Figure 10-33 on page 10-67 shows the assigned function codes. All other function codes are unassigned. Bit 56 of general register 0 must be zero; otherwise, a specification exception is recognized. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

follows.

Code	Function	Parm. Block Size (bytes)	Data Block Size (bytes)
0	PCKMO-Query	16	—
1	PCKMO-Encrypt-DEA-Key	32	8
2	PCKMO-Encrypt-TDEA-128-Key	40	8
3	PCKMO-Encrypt-TDEA-192-Key	48	8
18	PCKMO-Encrypt-AES-128-Key	48	16
19	PCKMO-Encrypt-AES-192-Key	56	16
20	PCKMO-Encrypt-AES-256-Key	64	16
Explanation: — Not applicable			

Figure 10-33. Function Codes for PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATION

All other function codes are unassigned.

The query function provides the means of indicating the availability of the other functions.

Figure 10-34 on page 10-67 shows the contents of general registers 0 and 1.

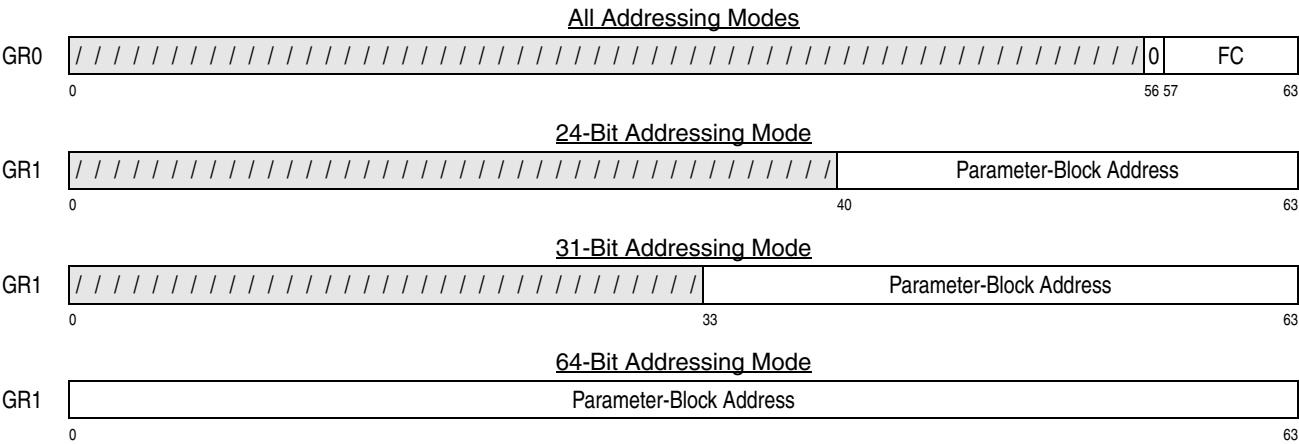


Figure 10-34. General Register Assignment for PCKMO

In the access-register mode, access register 1 specifies the address space containing the parameter block.

As observed by other CPUs and channel programs, reference to the parameter block may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

PCKMO-Query (Function Code 0)

The locations of the operands and addresses used by the instruction are as shown in Figure 10-34 on page 10-67.

The parameter block used for the function has the following format:

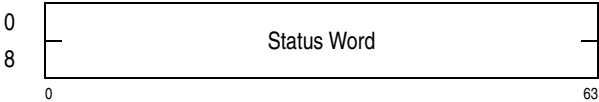


Figure 10-35. Parameter Block for PCKMO-Query

A 128-bit status word is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the PCKMO instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

PCKMO-Encrypt-DEA-Key (Function Code 1)

The locations of the operands and addresses used by the instruction are as shown in Figure 10-34 on page 10-67.

The parameter block used for the function has the following format:

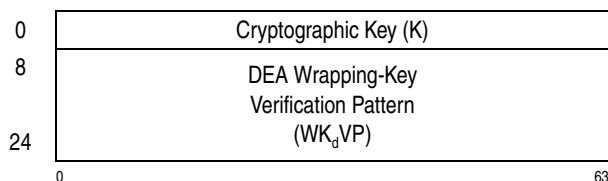


Figure 10-36. Parameter Block for PCKMO-Encrypt-DEA-Key

The 8-byte cryptographic key, K, in byte offsets 0-7 of the parameter block is encrypted using the DEA wrapping key. (See the section “Protection of Cryptographic Key” on page 7-339 for the encryption algorithm.) The result is placed back in byte offsets 0-7 of the parameter block. The contents of the DEA wrapping-key verification-pattern register are placed in byte offsets 8-31 of the parameter block.

PCKMO-Encrypt-TDEA-128-Key (Function Code 2)

The locations of the operands and addresses used by the instruction are as shown in Figure 10-34 on page 10-67.

The parameter block used for the function has the following format:

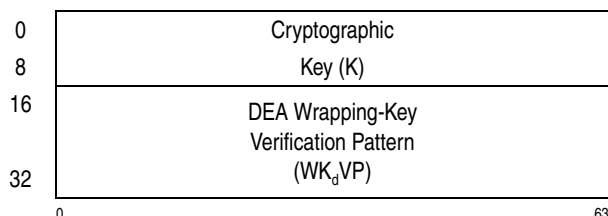


Figure 10-37. Parameter Block for PCKMO-Encrypt-TDEA-128-Key

The 16-byte cryptographic key, K, in byte offsets 0-15 of the parameter block is encrypted using the DEA wrapping key. (See the section “Protection of Cryptographic Key” on page 7-339 for the encryption algorithm.) The result is placed back in byte offsets 0-15 of the parameter block. The contents of the DEA

wrapping-key verification-pattern register are placed in byte offsets 16-39 of the parameter block.

PCKMO-Encrypt-TDEA-192-Key (Function Code 3)

The locations of the operands and addresses used by the instruction are as shown in Figure 10-34 on page 10-67.

The parameter block used for the function has the following format:

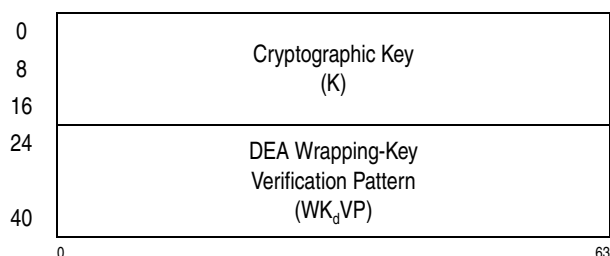


Figure 10-38. Parameter Block for PCKMO-Encrypt-TDEA-192-Key

The 24-byte cryptographic key, K, in byte offsets 0-23 of the parameter block is encrypted using the DEA wrapping key. (See the section “Protection of Cryptographic Key” on page 7-339 for the encryption algorithm.) The result is placed back in byte offsets 0-23 of the parameter block. The contents of the DEA wrapping-key verification-pattern register are placed in byte offsets 24-47 of the parameter block.

PCKMO-Encrypt-AES-128-Key (Function Code 18)

The locations of the operands and addresses used by the instruction are as shown in Figure 10-34 on page 10-67.

The parameter block used for the function has the following format:

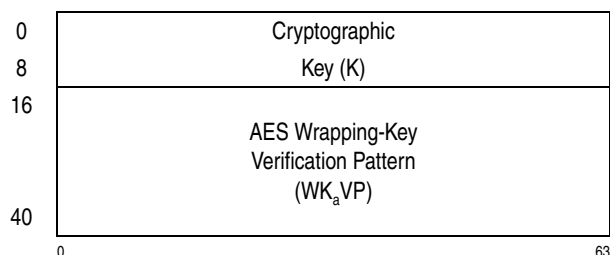


Figure 10-39. Parameter Block for PCKMO-Encrypt-AES-128-Key

The 16-byte cryptographic key, K, in byte offsets 0-15 of the parameter block is encrypted using the AES wrapping key. (See the section “Protection of Cryptographic Key” on page 7-339 for the encryption algorithm.) The result is placed back in byte offsets 0-15 of the parameter block. The contents of the AES wrapping-key verification-pattern register are placed in byte offsets 16-47 of the parameter block.

PCKMO-Encrypt-AES-192-Key (Function Code 19)

The locations of the operands and addresses used by the instruction are as shown in Figure 10-34 on page 10-67.

The parameter block used for the function has the following format:

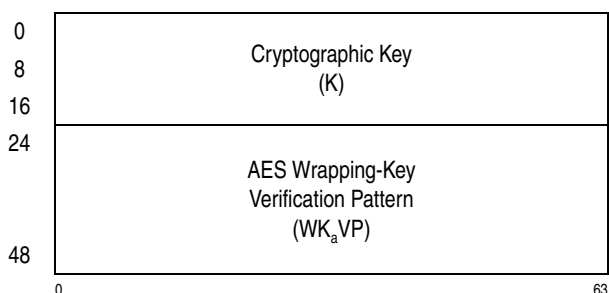


Figure 10-40. Parameter Block for PCKMO-Encrypt-AES-192-Key

The 24-byte cryptographic key, K, in byte offsets 0-23 of the parameter block is encrypted using the AES wrapping key. (See the section “Protection of Cryptographic Key” on page 7-339 for the encryption algorithm.) The result is placed back in byte offsets 0-23 of the parameter block. The contents of the AES wrapping-key verification-pattern register are placed in byte offsets 24-55 of the parameter block.

PCKMO-Encrypt-AES-256-Key (Function Code 20)

The locations of the operands and addresses used by the instruction are as shown in Figure 10-34 on page 10-67.

The parameter block used for the function has the following format:

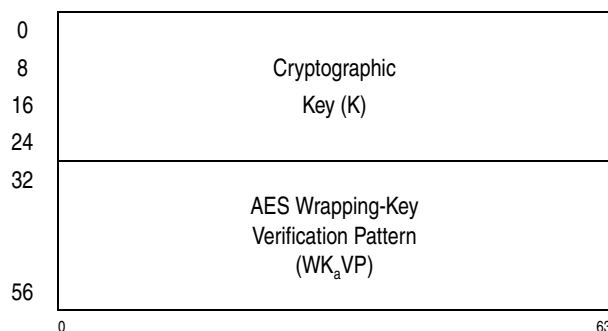


Figure 10-41. Parameter Block for PCKMO-Encrypt-AES-256-Key

The 32-byte cryptographic key, K, in byte offsets 0-31 of the parameter block is encrypted using the AES wrapping key. (See the section “Protection of Cryptographic Key” on page 7-339 for the encryption algorithm.) The result is placed back in byte offsets 0-31 of the parameter block. The contents of the AES wrapping-key verification-pattern register are placed in byte offsets 32-63 of the parameter block.

Special Conditions

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bit 56 of general register 0 is not zero.
2. Bits 57-63 of general register 0 specify an unassigned or uninstalled function code.

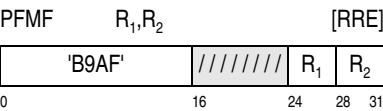
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, parameter block; store, parameter block)
- Operation (if the message-security-assist-extension-3 facility is not installed)
- Privileged operation
- Specification

Programming Note: Each set of PCKMO-Encrypt-DEA-Key functions and PCKMO-Encrypt-AES-Key functions can be independently disabled by an external means. When a set is disabled, functions in the set appear as if they were not installed.

PERFORM FRAME MANAGEMENT FUNCTION



Subject to the controls in the first-operand register, a frame-management function is performed for the storage frame designated by the second-operand address.

The contents of general register R₁ are as follows:

Frame-management function indications: Bit positions 44-47 of general register R₁ contain the frame-management function indications (FMFI), as follows:

Reserved: Bits 44 and 45 are reserved and must contain zeros; otherwise, a specification exception is recognized.

Set-Key Control (SK): Bit 46 controls whether the storage key for each 4K-byte block in the frame is set from bits 56-62 of general register R₁. When the SK control is zero, the keys are not set; when the SK control is one, the keys are set.

Clear-Frame Control (CF): Bit 47 controls whether the frame is set to zeros. When the CF control is zero, no clearing operation is performed. When the CF control is one, the frame is cleared to zeros.

Usage Indication (UI): Bit position 48 of general register R₁ contains the usage indication (UI). When bit 48 is zero, it indicates that the program does not anticipate immediate usage of the frame. When bit 48 is one, it indicates that program anticipates usage of the frame in the near future.

Frame-Size Code (FSC): Bits 49-51 of general register R₁ contain the frame-size code (FSC), as follows:

FSC	Meaning
0	4K-byte frame
1	1M-byte frame
2-7	Reserved

Nonquiescing Control (NQ): When the nonquiescing key-setting facility is installed and the set-key control, bit 46 of general register R₁ is one, bit 52 of general register R₁ controls whether a quiescing operation is performed, as described below. When the nonquiescing key-setting facility is installed and the set-key control is zero, bit 52 of general register R₁ is ignored.

When the nonquiescing key-setting facility is not installed, bit 52 of general register R₁ is reserved and must contain zero; otherwise a specification exception is recognized.

Reference-Bit Update Mask (MR): When the set-key control, bit 46 of general register R₁, is one, bit 53 of general register R₁ controls whether updates to the reference bit in the storage key may be bypassed, as described below. When the set-key control is zero, bit 53 of general register R₁ is ignored.

Change-Bit Update Mask (MC): When the set-key control, bit 46 of general register R₁, is one, bit 54 of general register R₁ controls whether updates to the change bit in the storage key may be bypassed. When the set-key control is zero, bit 54 of general register R₁ is ignored.

The handling of the MR and MC bits is identical to the handling of the corresponding bits of the M₃ field of the SET STORAGE KEY EXTENDED instruction (described on page 10-121), except that general register R₁ is not updated with the contents of the previous key, and the condition code is not changed.

Key: When the set-key control, bit 46 of general register R₁, is one, bits 56-62 of the register contain the storage key to be set for each 4K-byte block in the frame, with the access-protection bits, fetch-protection bit, reference bit, and change bit in bit positions 56-59, 60, 61, and 62, respectively. When the set-key control is zero, bits 56-62 of general register R₁ are ignored.

General register R₂ contains the real or absolute address of the storage frame upon which the frame-management function is to be performed. When the frame-size code designates a 4K-byte block, the second-operand address is real; when the frame-size code designates a 1M-byte block the second-operand address is absolute. The handling of the address in general register R₂ depends on the addressing mode. In the 24-bit addressing mode, the contents of bit positions 40-51 of the register, with 12 rightmost

- In the 31-bit addressing mode, bit position 32 of general register R_2 is set to zero.
- In either the 24-bit or 31-bit addressing mode, bits 0-31 of general register R_2 are unchanged.

When the clear-frame control is one, references to main storage within the second operand are not necessarily single-access references and are not necessarily performed in a left-to-right direction as observed by other CPUs and by channel programs. The clear operation is not subject to key-controlled protection; however low-address protection applies regardless of whether the frame-size code designates a 4 K-byte or 1 M-byte frame (that is, regardless of whether the second-operand address is real or absolute, respectively).

When the storage-key control is one, the operation for each 4K-byte block is similar to that described in “SET STORAGE KEY EXTENDED” on page 10-121, except that when the keys for multiple blocks are set, the condition code and the contents of general register R_1 are unchanged.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed, except that when the seven bits of all storage keys to be set are the same as bits 56-62 of general register R_1 , or when the MR and MC bits allow all the storage keys to remain unchanged, it is unpredictable whether the serialization and checkpoint-synchronization functions are performed after the operation completes. See “SET STORAGE KEY EXTENDED” on page 10-121 for details on the cases where setting of the key may be bypassed.

When the set-key control is one, and either (a) the nonquiescing key-setting facility is not installed, or (b) the facility is installed but the nonquiescing control (NQ) is zero, a quiescing operation is performed. When the set-key control is one, the nonquiescing key-setting facility is installed, and the NQ control is one, a quiescing operation is not necessarily performed. See “Storage-Key Accesses” on page 5-91 for a discussion the effects of quiescing on key-setting instructions, and see “Quiescing” on page 5-102 for details on the quiescing operation.

It is unpredictable whether the clear-frame or the set-key operation is performed first when both of the respective controls are one. When both the clear-frame and set-key controls are one, and either the MR or MC bit is also one, and the clear operation is performed first, the storage key that is compared with bits 56-62 of general register R_1 in the set-key operation is the key following the clear operation (that is, both the R and C bits will have been set to one by the clear operation). Provided that there is no other access to the storage by other CPUs or the channel subsystem, the final results of the instruction reflect the specified key value, including the specified R and C values when MR and MC are zero.

Special Conditions

A specification exception is recognized and the operation is suppressed for any of the following conditions:

- Bits 32-45, 55, or 63 of general register R_1 are not zero.
- When the nonquiescing key-setting facility is not installed, bit 52 of general register R_1 is not zero.
- The frame-size code specifies a reserved value.

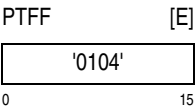
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, low-address-protection only, operand 2, when the clear-frame control is one)
- Addressing (operand 2)
- Operation (enhanced-DAT facility not installed)
- Privileged operation
- Specification

Programming Note: When PFMF is issued with the set-key control set to one, the program must ensure that no channel subsystem is simultaneously altering the storage designated by general register R_2 . The program also must ensure that no other CPU or channel subsystem is accessing the storage designated by general register R_2 when the nonquiescing (NQ) control is one. Otherwise, unpredictable results may be observed by the other CPUs and channel subsystem, including the alteration of the block designated by general register R_2 .

PERFORM TIMING FACILITY
FUNCTION



A timing facility function specified by the function code in general register 0 is performed. The condition code is set to indicate the outcome of the function. General register 1 contains the address of a parameter block in storage. PTFF query functions place information in the parameter block; PTFF control functions use information obtained from the parameter block.

As observed by other CPUs and channel programs, references to the parameter block may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

Bit positions 57-63 of general register 0 contain the function code. Figure 10-44 on page 10-74 shows the assigned function codes. Bit 56 of general register 0 must be zero; otherwise, a specification exception is recognized. All other bits of general register 0 are ignored.

General register 1 contains the logical address of the leftmost byte of the parameter block in storage. In the 24-bit addressing mode, the contents of bit positions 40-63 of general register 1 constitute the address, and the contents of bit positions 0-39 are ignored. In the 31-bit addressing mode, the contents of bit positions 33-63 of general register 1 constitute the address, and the contents of bit positions 0-32 are ignored. In the 64-bit addressing mode, the contents of bit positions 0-63 of general register 1 constitute the address.

Figure 10-43 on page 10-73 shows the contents of the general registers just described.

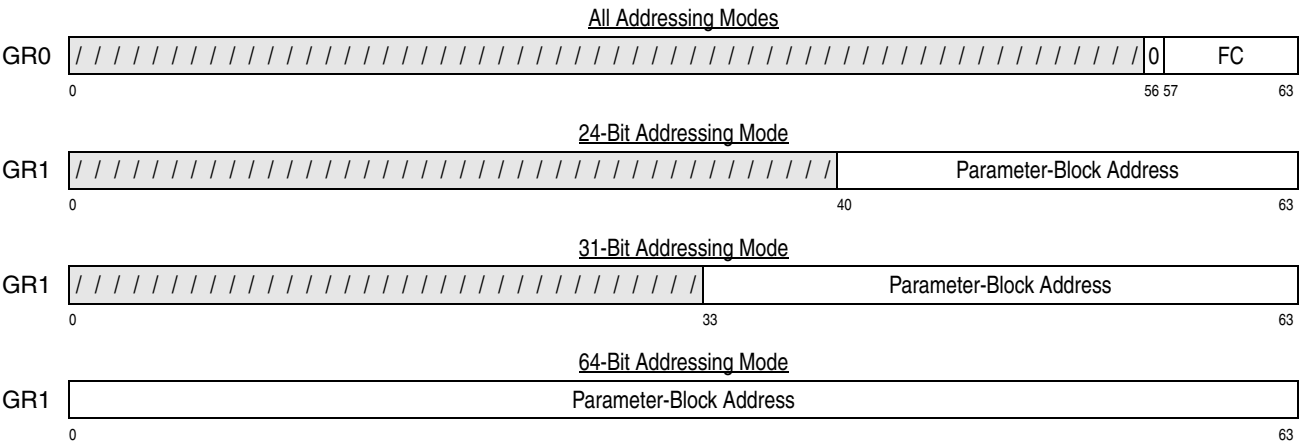


Figure 10-43. General Register Assignment for PERFORM TIMING FACILITY FUNCTION

In the access-register mode, access register 1 specifies the address space containing the parameter block.

The function codes for PERFORM TIMING FACILITY FUNCTION are as follows.

Code		Function	Parm. Block Size (bytes)	Action
Hex	Dec			
00	0	PTFF-QAF	16	P
01	1	PTFF-QTO	32	P
02	2	PTFF-QSI	56	P
03	3	PTFF-QPT	8	P
40	64	PTFF-ATO	8	Sb
41	65	PTFF-STO	8	Sh
42	66	PTFF-SFS	4	Sb
43	67	PTFF-SGS	4	Sb
Explanation:				
P	Available in problem state at all levels of CPU execution (basic-machine, logical-partition, and virtual-machine).			
Sb	Available in supervisor state at the basic-machine level.			
Sh	Available in supervisor state at the basic-machine level. May be available to the supervisor-state program at the logical-partition and virtual-machine levels, subject to emulation by the hypervisor.			

Figure 10-44. Function Codes for PERFORM TIMING FACILITY FUNCTION

All other function codes are unassigned.

The PTFF-QAF (Query Available Functions) function provides the means of indicating the availability of the other functions. The PTFF-QAF function indicates whether the function is installed for a program running in supervisor state at the basic-machine level.

PTFF-QAF (Query Available Functions)

The parameter block used for the function has the following format:

Hex	Dec		
00	0	Status Bits 0-31	pb.w1
04	4	Status Bits 32-63	pb.w2
08	8	Status Bits 64-95	pb.w3
0C	12	Status Bits 96-127	pb.w4
		0	31

Figure 10-45. Parameter Block for PTFF-QAF (Query Available Functions)

A 128-bit field is stored in the parameter block. Bits 0-127 of this field correspond to function codes 0-127, respectively, of the PTFF instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed. Bits 0-3 of pb.w1 are set to ones, as these bits represent function codes 0-3, which are currently assigned to query functions. Bits 4-31 of pb.w1 are reserved for additional query functions. Bits 0-3 of pb.w3 are set to ones, as these bits represent function codes 64-67, which are currently assigned to control functions. Bits 4-31 of pb.w3 are reserved for additional control functions. Parameter block words pb.w2 and pb.w4 are reserved for future extensions.

PTFF-QTO (Query TOD Offset)

The parameter block used for the function has the following format:

Hex	Dec		
00	0	Physical Clock (0:31)	pb.Tu
04	4	Physical Clock (32:63)	
08	8	TOD Offset (0:31)	pb.d
0C	12	TOD Offset (32:63)	
10	16	Logical TOD Offset (0:31)	pb.dl
14	20	Logical TOD Offset (32:63)	
18	24	TOD Epoch Difference (0:31)	pb.ed
1C	28	TOD Epoch Difference (32:63)	
		0	31

Figure 10-46. Parameter Block for PTFF-QTO (Query TOD Offset)

The 64-bit physical-clock value returned (pb.Tu) is the value of the physical clock at the most recent TOD-offset-update event.

The 64-bit TOD-offset value returned (pb.d) indicates the value of the TOD-offset (d). This is the current value being added to Tr (the physical clock) to obtain Tb (the basic-machine TOD clock); that is, $T_b = T_r + pb.d$.

The 64-bit logical-TOD-offset value returned (pb.dl) indicates the current value being added to Tr (the physical clock) to obtain Tc (the logical TOD clock for the current level of CPU execution); that is, $T_c = T_r + pb.dl$. Thus, when executed at the basic-machine level, $pb.dl = pb.d = d$; when executed at the logical-partition level, the Tp-Tb epoch difference (Dp) is added, resulting in $pb.dl = d + D_p$; and when executed at the virtual-machine level, the Tv-Tp epoch difference (Dv) is also added, resulting in $pb.dl = d +$

Dp + Dv. Carries, if any, out of bit position 0 are ignored in the addition for each of these equations.

The 64-bit TOD epoch difference value returned (pb.ed) is the TOD epoch difference for the current level of CPU execution. When executed at the basic-machine level, this value is zero; when executed at the logical-partition level, this value is the Tp-Tb epoch difference (Dp); when executed at the virtual-machine level, this value is the Tv-Tp epoch difference (Dv).

PTFF-QSI (Query Steering Information)

The parameter block used for the function has the following format:

Hex	Dec		
00	0	Physical Clock (0:31)	pb.Tu
04	4	Physical Clock (32:63)	
08	8	Old Episode Start Time (0:31)	pb.old.s
0C	12	Old Episode Start Time (32:63)	
10	16	Old Episode Base Offset (0:31)	pb.old.b
14	20	Old Episode Base Offset (32:63)	
18	24	Old Episode Fine-Steering Rate	pb.old.f
1C	28	Old Episode Gross-Steering Rate	pb.old.g
20	32	New Episode Start Time (0:31)	pb.new.s
24	36	New Episode Start Time (32:63)	
28	40	New Episode Base Offset (0:31)	pb.new.b
2C	44	New Episode Base Offset (32:63)	
30	48	New Episode Fine-Steering Rate	pb.new.f
34	52	New Episode Gross-Steering Rate	pb.new.g

Figure 10-47. Parameter Block for PTFF-QSI (Query Steering Information)

The 64-bit physical-clock value returned (pb.Tu) is the value of the physical clock at the most recent TOD-offset-update event. The remaining fields are the values of the old-episode and new-episode registers.

PTFF-QPT (Query Physical Clock)

The parameter block used for the function has the following format:

Hex	Dec		
00	0	Physical Clock (0:31)	pb.Tr
04	4	Physical Clock (32:63)	

Figure 10-48. Parameter Block for PTFF-QPT (Query Physical Clock)

The 64-bit physical-clock value returned (pb.Tr) is the current value of the physical clock. Zeros are stored for the rightmost bit positions that are not provided by the physical clock. When the clock is running, two executions of PTFF-QPT, either on the same or different CPUs, do not necessarily return different values of the clock.

PTFF-ATO (Adjust TOD Offset)

The parameter block used for the function has the following format:

Hex	Dec		
00	0	TOD Offset Adjustment (0:31)	pb.a
04	4	TOD Offset Adjustment (32:63)	

Figure 10-49. Parameter Block for PTFF-ATO (Adjust TOD Offset)

The 64-bit value (pb.a) from the parameter block, treated as an unsigned-binary value, is added to the base offset of the next episode. A carry if any, out of bit position 0 is ignored in this addition. The effect is not immediate, but is scheduled to coincide with the next TOD-offset-update event.

If the next episode has already been scheduled, and has not yet become active, then the sum of pb.a and new.b replaces new.b and no other action is taken.

If the next episode has not been scheduled (that is, the new-episode registers are the current episode), then the new-episode registers are saved in the old-episode registers and a new episode is scheduled (thus, making the old-episode registers the current episode). The new-episode start time (new.s) is set to the value the physical clock will have at the next TOD-offset-update event and the new-episode base offset (new.b) is set to the sum of pb.a and the value

the TOD-offset would have had at that same instant, computed using the current steering parameters.

The steering rate is not changed by this function, if a new episode is scheduled, the new-episode fine-steering and gross-steering rates are the same as the current values.

Execution of the adjust-TOD-offset function is interlocked such that the entire contents of the TOD-offset register appear to be updated concurrently and simultaneously as observed by all CPUs in the configuration. However, accesses to the logical TOD clocks (basic-machine TOD clock, logical-partition TOD clock, and virtual-machine TOD clock) by CPUs in the configuration are not artificially delayed; thus, addition of a large unsigned adjustment value may have the effect of a negative change and may cause the logical TOD clocks to appear to step backwards.

PTFF-STO (Set TOD Offset)

The parameter block used for the function has the following format:

Hex	Dec		
00	0	New TOD Offset (0:31)	pb.d
04	4	New TOD Offset (32:63)	
		0	31

Figure 10-50. Parameter Block for PTFF-STO (Set TOD Offset)

The 64-bit value (pb.d) from the parameter block replaces the TOD offset.

When issued at the basic-machine level, the effect is not immediate, but is scheduled to coincide with the next TOD-offset-update event.

If the next episode has already been scheduled, and has not yet become active, then pb.d replaces new.b and no other action is taken.

If the next episode has not been scheduled (that is, the new-episode registers are the current episode), then the new-episode registers are saved in the old-episode registers and a new episode is scheduled (thus, making the old-episode registers the current episode). The new-episode start time (new.s) is set to the value the physical clock will have at the next TOD-offset-update event and the new-episode base offset (new.b) is set to the value of pb.d.

The steering rate is not changed by this function, if a new episode is scheduled, the new-episode fine-steering and gross-steering rates are the same as the current values.

When issued at the logical-partition or virtual-machine level, the function may be simulated by the hypervisor and operates on the TOD epoch difference for the current level of CPU execution (Dp or Dv, respectively); no new episode is scheduled and the change takes effect immediately.

Execution of the set-TOD-offset function is interlocked such that the entire contents of the TOD-offset register appear to be updated concurrently and simultaneously as observed by all CPUs in the configuration. However, accesses to the logical TOD clocks (basic-machine TOD clock, logical-partition TOD clock, and virtual-machine TOD clock) by CPUs in the configuration are not artificially delayed; thus, replacement of the TOD offset by a smaller value may cause the logical TOD clocks to appear to step backwards.

PTFF-SFS (Set Fine-Steering Rate)

The parameter block used for the function has the following format:

Hex	Dec		
00	0	New Fine-Steering Rate	pb.f
		0	31

Figure 10-51. Parameter Block for PTFF-SFS (Set Fine-Steering Rate)

The 32-bit value (pb.f) from the parameter block becomes the fine-steering rate for the next episode. The effect is not immediate, but is scheduled to coincide with the next TOD-offset-update event.

If the next episode has already been scheduled, and has not yet become active, then pb.f replaces new.f and no other action is taken.

If the next episode has not been scheduled (that is, the new-episode registers are the current episode), then the new-episode registers are saved in the old-episode registers and a new episode is scheduled (thus, making the old-episode registers the current episode). The new-episode start time (new.s) is set to the value the physical clock will have at the next TOD-offset-update event and the new-episode base offset (new.b) is set to the value the TOD-offset will

have at that same instant, computed using the current steering parameters. The new-episode fine-steering rate (new.f) is set to pb.f and the new-episode gross-steering rate is the same as the current value.

When the new episode takes effect, accesses to the logical TOD clocks by CPUs in the configuration are interlocked to ensure that the logical TOD clocks appear to be unique and monotonically increasing as observed by all programs.

PTFF-SGS (Set Gross-Steering Rate)

The parameter block used for the function has the following format:

Hex	Dec	
00	0	<div style="border: 1px solid black; padding: 2px; display: inline-block;">New Gross-Steering Rate</div> pb.g
		0 31

Figure 10-52. Parameter Block for PTFF-SGS (Set Gross-Steering Rate)

The 32-bit value (pb.g) from the parameter block becomes the gross-steering rate for the next episode. The effect is not immediate, but is scheduled to coincide with the next TOD-offset-update event.

If the next episode has already been scheduled, and has not yet become active, then pb.g replaces new.g and no other action is taken.

If the next episode has not been scheduled (that is, the new-episode registers are the current episode), then the new-episode registers are saved in the old-episode registers and a new episode is scheduled (thus, making the old-episode registers the current episode). The new-episode start time (new.s) is set to the value the physical clock will have at the next TOD-offset-update event and the new-episode base offset (new.b) is set to the value the TOD-offset will have at that same instant, computed using the current steering parameters. The new-episode gross-

steering rate (new.g) is set to pb.g and the new-episode fine-steering rate is the same as the current value.

When the new episode takes effect, accesses to the logical TOD clocks by CPUs in the configuration are interlocked to ensure that the logical TOD clocks appear to be unique and monotonically increasing as observed by all programs.

Special Conditions

The priority of execution for the PTFF instruction is shown in Figure 10-53.

A privileged operation exception is recognized if a PTFF control function is issued in the problem state.

A specification exception is recognized and no other action is taken if any of the following occurs:

1. Bit 56 of general register 0 is not zero.
2. Bits 57-63 of general register 0 specify an unsigned or uninstalled function code.

Resulting Condition Code:

- | | |
|---|----------------------------------|
| 0 | Requested function performed |
| 1 | -- |
| 2 | -- |
| 3 | Requested function not available |

Program Exceptions:

- Access (fetch, parameter block for control functions; store, parameter block for query functions)
- Operation (if the PERFORM TIMING FACILITY FUNCTION instruction is not installed)
- Privileged operation (attempt to execute a PTFF control function in the problem state)
- Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.	Operation exception.
8.	Specification exception due to bit 56 of general register 0 being one or bits 57-63 of general register 0 specifying an unassigned or uninstalled function code.
9.	Privileged-operation exception due to attempt to issue PTFF control function in the problem state.
10.	Condition code 3 due to a control function not available in supervisor state.
11.	Access exceptions for an access to the parameter block.
12.	Condition code 0 indicating successful completion.

Figure 10-53. Priority of Execution: *PERFORM TIMING FACILITY FUNCTION (PTFF)*

Programming Notes:

- The status bits returned by the PTFF-QAF function indicate the PTFF functions which are installed and available for a program running in supervisor state at the basic-machine level. Since all installed PTFF query functions are available in both the problem state and supervisor state at all levels of CPU execution, these status bits can be used directly by any program to determine which PTFF query functions are available. PTFF control functions are never available to a program running in problem state. Condition code 3 is set when a particular PTFF control function is issued and the function is installed but is not available in that state or at that level of CPU execution. Thus, the status bits for PTFF control functions indicate whether it is safe to issue a particular PTFF control function without the danger of a specification exception.
- The physical time (pb.Tu) returned in the parameter block for PTFF-QTO and PTFF-QSI is the current value of the physical clock, but with zeroes in positions to the right of the bit position incremented concurrently with each TOD-offset-update event. This value is not necessarily unique. The same value may be returned if the PTFF instruction is issued multiple times on the same CPU or issued at approximately the same time on different CPUs.
- The 64-bit value of the physical clock at the most recent TOD-offset-update event (pb.Tu) returned by PTFF-QTO and PTFF-QSI may be used by the program to ensure that all events during a short routine occurred with no intervening TOD-offset-update events. This can be accomplished by issuing one of these queries at the beginning of the routine and the other at the end and reiterating if the value of pb.Tu before and the value of pb.Tu after are different.
- Given information from PTFF-QTO, the following equations show the relationship between Tr (the physical clock), Tb (basic-machine TOD clock), and Tc (logical TOD clock for the current level of CPU execution):

$$Tb = Tr + pb.d$$

$$Tc = Tr + pb.dl$$

These equations hold only for the time between two TOD-offset-update events. The information from PTFF-QSI can be used to extend this relationship for the entire time span of two episodes, beginning with the old-episode start time (pb.old.s) through the new episode and continuing on to that instant in the future when a subsequent episode takes effect.

- For PTFF-QTO (Query TOD Offset) and PTFF-QSI (Query Steering Information), the 64-bit physical-clock value returned (pb.Tu) is the value of the physical clock at the most recent TOD-offset-update event. This is the current value of the physical clock, but with zeroes in positions to the right of the bit position incremented concurrently with each TOD-offset-update event. The program can use this value to determine the time span over which the various offsets returned by Query TOD Offset can be used without any error. For Query Steering Information, pb.Tu can be used to determine whether the machine is operating in the old or new episode. When $pb.new.s \leq pb.Tu$, the machine is operating in the new episode, and none of the information returned by the query has changed since the most recent TOD-offset-update event. When $pb.new.s > Pb.Tr$, the machine is operating in the old episode, a new episode has been scheduled and has not yet taken effect. The program should avoid any dependency on the new episode values returned

in this case, as it is possible the values may change again before the new episode takes effect.

6. Two or more executions of PTFF-QPT within a short period of time, either on the same or different CPUs, do not appear to step backwards, but are not necessarily unique. Thus, it is not always possible to use the values to determine the sequence of execution.
7. The Network Time Protocol (NTP) is used extensively in the internet. A description of NTP may be found in Request for Comments (RFC) 1305. Figure 10-55 on page 10-80 shows the value of the TOD clock and NTP timestamp for 00:00:00

(0 am), UTC time, for several dates: January 1, 1900, January 1, 1972, and for that instant in time just after each of the 23 leap seconds that have occurred through January, 2006. Each of these leap seconds was inserted in the UTC time scale beginning at 23:59:60 UTC of the day previous to the one listed and ending at 00:00:00 UTC of the day listed. Also included are values for the beginning of each year since the most recent leap second. Figure 10-54 on page 10-79 shows the values of the TOD clock and NTP timestamp for those instants just after the leftmost bit of one or the other has changed. For instants in the future, the number of leap seconds must be applied and this value is not yet known.

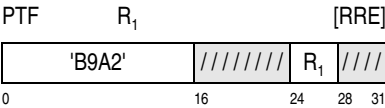
YYYY	MM	DD	hh mm ss	TOD Clock (Hex)	NTP Timestamp (Hex)
1968	01	20	03:14:08.000000	7A12 0000 0000 0000	8000 0000 0000 0000
1971	05	11	11:56:53.685248	8000 0000 0000 0000	8637 BD05 AF6C 69B5*
2036	02	07	06:28:16.000000	F424 0000 0000 0000+	0000 0000 0000 0000
2042	09	17	23:53:47.370496-	0000 0000 0000 0000	0C6F 7A0B 5ED8 D36B*-
Explanation:					
* Value is not exact.					
+ Leap seconds must be added to this value.					
- Leap seconds must be subtracted from this value.					

Figure 10-54. TOD Clock and NTP Timestamps When a Leftmost Bit Changes

Year	Month	Day	Leap Sec.	TOD Clock (Hex)	NTP Timestamp (Hex)
1900	1	1		0000 0000 0000 0000	0000 0000 0000 0000
1972	1	1		8126 D60E 4600 0000	876C E580 0000 0000
1972	7	1	1	820B A981 1E24 0000	885C D680 0000 0000
1973	1	1	2	82F3 00AE E248 0000	894F 6A80 0000 0000
1974	1	1	3	84BD E971 146C 0000	8B30 9E00 0000 0000
1975	1	1	4	8688 D233 4690 0000	8D11 D180 0000 0000
1976	1	1	5	8853 BAF5 78B4 0000	8EF3 0500 0000 0000
1977	1	1	6	8A1F E595 20D8 0000	90D5 8A00 0000 0000
1978	1	1	7	8BEA CE57 52FC 0000	92B6 BD80 0000 0000
1979	1	1	8	8DB5 B719 8520 0000	9497 F100 0000 0000
1980	1	1	9	8F80 9FDB B744 0000	9679 2480 0000 0000
1981	7	1	10	9230 5C0F CD68 0000	994A 4900 0000 0000
1982	7	1	11	93FB 44D1 FF8C 0000	9B2B 7C80 0000 0000
1983	7	1	12	95C6 2D94 31B0 0000	9D0C B000 0000 0000
1985	7	1	13	995D 40F5 17D4 0000	A0D0 6880 0000 0000
1988	1	1	14	9DDA 69A5 57F8 0000	A585 6380 0000 0000
1990	1	1	15	A171 7D06 3E1C 0000	A949 1C00 0000 0000
1991	1	1	16	A33C 65C8 7040 0000	AB2A 4F80 0000 0000
1992	7	1	17	A5EC 21FC 8664 0000	ADFB 7400 0000 0000
1993	7	1	18	A7B7 0ABE B888 0000	AFDC A780 0000 0000
1994	7	1	19	A981 F380 EAAC 0000	B1BD DB00 0000 0000
1996	1	1	20	AC34 336F ECD0 0000	B581 9380 0000 0000
1997	7	1	21	AEE3 EFA4 02F4 0000	B674 2780 0000 0000
1999	1	1	22	B196 2F93 0518 0000	BA36 8E80 0000 0000
2000	1	1	22	B361 1854 4318 0000	BC17 C200 0000 0000
2001	1	1	22	B52D 42F2 F718 0000	BDFA 4700 0000 0000
2002	1	1	22	B6F8 2BB4 3518 0000	BFDB 7A80 0000 0000
2003	1	1	22	B8C3 1475 7318 0000	C1BC AE00 0000 0000
2004	1	1	22	BA8D FD36 B118 0000	C39D E180 0000 0000
2005	1	1	22	BC5A 27D5 6518 0000	C580 6680 0000 0000
2006	1	1	23	BE25 1097 973C 0000	C761 9A00 0000 0000

Figure 10-55. TOD Clock and NTP Timestamps

PERFORM TOPOLOGY FUNCTION



The contents of general register R_1 specify a function code in bit positions 56-63, as illustrated in Figure 10-56.

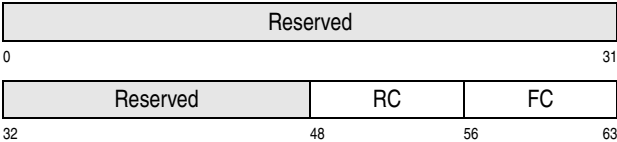


Figure 10-56. General-Register R_1 Format

The defined function codes are as follows:

FC Meaning

0	Request horizontal polarization.
1	Request vertical polarization.
2	Check topology-change status.

Undefined function codes in the range 0-255 are reserved for future extensions.

Upon completion, if condition code 2 is set, a reason code is stored in bit positions 48-55 of general register R₁.

Bits 16-23 and 28-31 of the instruction are ignored.

Operation of Function Codes 0 and 1

When no exceptional conditions are detected, a process is initiated to place all CPUs in the configuration into the polarization specified by the function code, and condition code 0 is set. Completion of the process is asynchronous with respect to execution of the instruction and may or may not be completed when execution of the instruction completes.

Execution completes with condition code 2, and the reason code is set, for any of the following reasons:

RC Reason

0	No reason specified.
1	The configuration is already polarized as specified by the function code.
2	A topology change is already in process

Operation of Function Code 2:

The topology-change-report-pending condition is checked. When a topology-change-report is not pending, condition code 0 is set. When a topology-change report is pending, condition code 1 is set.

A topology change is any alteration such that the contents of a SYSIB 15.1.2 would be different from the contents of the SYSIB 15.1.2 prior to the topology change.

A topology-change-report-pending condition is created when a topology-change process completes. A topology-change-report-pending condition is cleared for the configuration when any of the following is performed:

- Execution of PERFORM TOPOLOGY FUNCTION specifies function-code 2 that completes with condition code 1.
- Subsystem reset is performed.

Special Conditions

A specification exception is recognized for either of the following conditions:

- Bit positions 0-55 of general register R₁ are not zeros.
- An undefined function code is specified.

Resulting Condition Code: When the function code is 0 or 1, the condition code is set as follows:

0	Topology-change initiated
1	--
2	Request rejected
3	--

When the function code is 2, the condition code is set as follows:

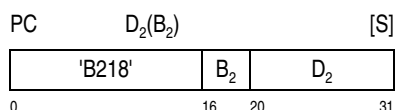
0	Topology-change-report not pending
1	Topology-change report pending
2	--
3	--

Program Exceptions:

- Operation (configuration-topology facility is not installed)
- Privileged operation
- Specification

Programming Note: Further information on configuration topology may be found in "SYSIB 15.1.2 - 15.1.6 (Configuration Topology)" on page 10-142.

PROGRAM CALL



A program-call number specified by the second-oper- and address is used in a two-level or three-level lookup to locate an entry-table entry (ETE). The program is authorized to use the ETE when the AND of

the PSW-key mask in control register 3 and the authorization key mask in the ETE is nonzero or when the CPU is in the supervisor state.

When the PC-type bit, bit 128 of the ETE, is zero, an operation called basic PROGRAM CALL is performed. When the PC-type bit is one, an operation called stacking PROGRAM CALL is performed.

Basic PROGRAM CALL, in the 24-bit or 31-bit addressing mode, loads the basic-addressing-mode bit, bits 33-62 of the updated instruction address, and the problem-state bit from the PSW into bit positions 32-63 of general register 14, and it leaves bits 0-31 of this register unchanged. In the 64-bit addressing mode, bits 0-62 of the updated instruction address and the problem-state bit are placed in bit positions 0-63 of general register 14. In any addressing mode, the PSW-key mask and PASN are placed in bit positions 32-63 of general register 3, and bits 0-31 of this register remain unchanged.

Stacking PROGRAM CALL places the entire PSW contents, except with an unpredictable PER mask, and also the PSW-key mask, PASN, SASN, and EAX in a linkage-stack program-call state entry that it forms. A called-space identification, an indication of whether the resulting addressing mode is the 64-bit mode, the numeric part of the program-call number, and the contents of general registers 0-15 and access registers 0-15 also are placed in the state entry. If the ASN-and-LX-reuse facility is installed and enabled, the PASTEIN and SASTEIN also are placed in the state entry.

For basic PROGRAM CALL, the extended-addressing-mode bit, bit 31 of the PSW, must have the same value as the entry-extended-addressing-mode bit, bit 129 of the ETE; otherwise, a special-operation exception is recognized. Basic PROGRAM CALL does not change bit 31 of the PSW and, therefore, does not switch between a basic addressing mode (the 24-bit or 31-bit mode) and the extended addressing mode (the 64-bit mode). In the 24-bit or 31-bit addressing mode, basic PROGRAM CALL sets the basic-addressing-mode bit, bit 32 of the PSW, with the value of the entry-basic-addressing-mode bit, bit 32 of the ETE, and, thus, it may switch between the 24-bit and 31-bit addressing modes. In the 64-bit addressing mode, bit 32 of the PSW remains unchanged.

Stacking PROGRAM CALL, when bit 129 of the ETE is zero, sets bit 31 of the PSW to zero and sets bit 32

of the PSW with the value of bit 32 of the ETE. When bit 129 of the ETE is one, stacking PROGRAM CALL sets bits 31 and 32 of the PSW to one. Thus, stacking PROGRAM CALL can set the 24-bit, 31-bit, or 64-bit addressing mode.

When the resulting addressing mode is the 24-bit or 31-bit mode, both basic and stacking PROGRAM CALL place bits 33-62 of the entry instruction address in the ETE, which are bits 33-62 of the ETE, with 33 leftmost and one rightmost zeros appended, in bit positions 64-127 of the PSW as the new instruction address, and they place the entry-problem-state bit, bit 63 of the ETE, in bit position 15 of the PSW as the new problem-state bit. Bits 32-63 of the entry parameter in the ETE are placed in bit positions 32-63 of general register 4, and bits 0-31 of this register remain unchanged.

When the resulting addressing mode is the 64-bit mode, both basic and stacking PROGRAM CALL place bits 0-62 of the entry instruction address, bits 0-62 of the ETE, with one rightmost zero appended, in bit positions 64-127 of the PSW, and they place bit 63 of the ETE in bit position 15 of the PSW. Bits 0-63 of the entry parameter in the ETE are placed in general register 4.

Basic PROGRAM CALL ORs the entry key mask from the ETE into the PSW-key mask in control register 3. Stacking PROGRAM CALL does the same, or it replaces the PSW-key mask with the entry key mask, as determined by the PSW-key-mask control in the ETE.

Stacking PROGRAM CALL optionally replaces the PSW key in the PSW and the EAX in control register 8 from the ETE, and it sets the address-space-control bits in the PSW, as determined by control bits in the ETE.

The ETE causes a space-switching operation to occur if it contains a nonzero ASN. When the ETE contains a zero ASN, the operation is called PROGRAM CALL to current primary (PC-cp); when the ETE contains a nonzero ASN, the operation is called PROGRAM CALL with space switching (PC-ss). When space switching is specified, the new PASN is loaded into control register 4 from the ETE, and a new primary-ASTE origin (PASTEO) is loaded into control register 5, also from the ETE. From the PASTE, a new primary ASCE (PASCE) and AX are loaded into control registers 1 and 4, respectively. If ASN-and-LX reuse is enabled, a new primary ASTE

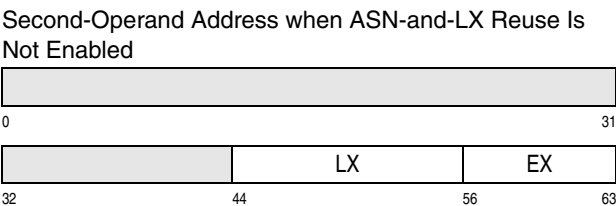
instance number (PASTEIN) is loaded into control register 4, also from the PASTE.

In both PC-cp and PC-ss, the SASN and secondary ASCE (SASCE) are set equal to the original PASN and PASCE, respectively, and, if ASN-and-LX reuse is enabled, the secondary ASTE instance number (SASTEIN) in control register 3 is set equal to the original PASTEIN. However, the space-switching stacking PROGRAM CALL operation may instead set the SASN and SASCE equal to the new PASN and PASCE, respectively, and, if ASN-and-LX reuse is enabled, set the SASTEIN equal to the new PASTEIN, as determined by a control bit in the ETE.

In a PC-ss to the base space of the dispatchable unit when the dispatchable unit is subspace active, bits 0-55 and 58-63 of the new PASCE are replaced by the same bits of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. This occurs before the possible setting of the SASCE equal to the PASCE.

PROGRAM CALL PC-Number Translation

The second-operand address is not used to address data; instead, the rightmost 20 or 32 bits of the address are used as a PC number divided into two fields: a linkage index (LX) and an entry index (EX). The entry index is always the rightmost eight bits of the PC number. When the ASN-and-LX-reuse is not installed or is not enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, the PC number is 20 bits, and the leftmost 12 bits of the number are the linkage index. In this case, the second-operand address has the following format:

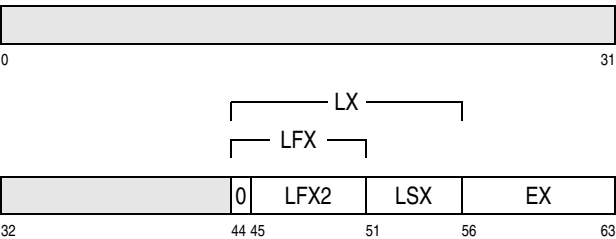


Bit 44 of the effective address has no special meaning and may be zero or one.

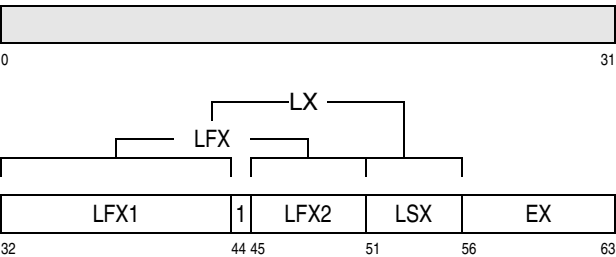
When the ASN-and-LX-reuse facility is installed and enabled, referred to simply by saying that ASN-and-LX reuse is enabled, the linkage index is further divided into a linkage first index (LFX) and a linkage second index (LSX). The linkage second index is

always the five bits immediately on the left of the entry index. The size and format of the linkage first index depend on whether the PC number is 20 bits or 32 bits, which in turn depends on whether bit 44 of the second-operand address is zero or one, respectively. In these cases, the second-operand address has the following formats:

Second-Operand Address when ASN-and-LX Reuse Is Enabled and Bit 44 Is Zero



Second-Operand Address when ASN-and-LX Reuse Is Enabled and Bit 44 Is One



When ASN-and-LX reuse is enabled and bit 44 of the second-operand address is zero, the PC number is 20 bits, and the linkage first index is bits 44-50, or bits 45-50 (LFX2) with a zero appended on the left. Thus, the linkage first index is seven bits of which the leftmost bit is always zero. When bit 44 is one, the PC number is 32 bits, and the linkage first index is bits 45-50 (LFX2) with bits 32-43 (LFX1) appended on the left, or 18 bits, and bit 44 is not a numeric part of the PC number. However, a linkage first table can contain at most 16,384 entries, and, therefore, the leftmost four bits of the linkage first index, bits 32-35 of the second-operand address, must always be zeros; otherwise, an LFX-translation exception is recognized.

Linkage Index (LX): When ASN-and-LX reuse is not enabled, bits 44-55 of the second-operand address are the linkage index and are used to select an entry from the linkage table designated by the linkage-table designation in the primary ASTE.

Linkage First Index (LFX): When ASN-and-LX reuse is enabled and bit 44 of the second-operand address is zero, bits 44-50 of the second-operand address are the linkage first index. When ASN-and-LX reuse is enabled and bit 44 of the second-operand address is one, bits 45-50 with bits 32-43 appended on the left are the linkage first index. In either case, the linkage first index is used to select an entry from the linkage first table designated by the linkage-first-table designation in the primary ASTE.

Linkage Second Index (LSX): When ASN-and-LX reuse is enabled, bits 51-55 of the second-operand address are the linkage second index and are used to select an entry from the linkage second table designated by the linkage-first-table entry.

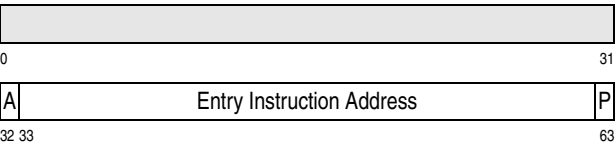
Entry Index (EX): Bits 56-63 of the second-operand address are the entry index and are used to select an entry from the entry table designated by the linkage-table entry or the linkage-second-table entry.

When the PC number is 20 bits, bits 32-43 of the second-operand address are ignored. Bits 0-31 of the second-operand address are always ignored.

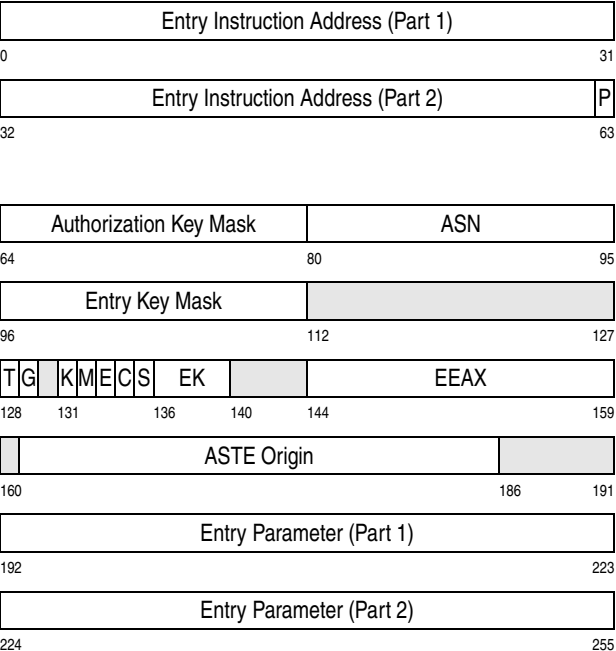
When ASN-and-LX reuse is enabled, the second word of the linkage-second-table entry used contains a linkage-second-table-entry sequence number (LSTESN). When this LSTESN is nonzero, it must be equal to an LSTESN specified in bit positions 0-31 of general register 15; otherwise, an LSTE-sequence exception is recognized. When ASN-and-LX reuse is not enabled (including when it is not installed) or the LSTESN in the linkage-second-table entry is zero, bits 0-31 of general register 15 are ignored.

The 32-byte entry-table entry (ETE) has the following format:

When Resulting Addressing Mode Is the 24-Bit or 31-Bit Mode



When Resulting Addressing Mode Is the 64-Bit Mode



For basic PROGRAM CALL in the 24-bit or 31-bit addressing mode when bit 32 of the ETE (A) is zero (specifying the 24-bit mode), and for stacking PROGRAM CALL when bits 32 (A) and 129 (G) are zeros (specifying the 24-bit mode), bits 33-39 must be zeros; otherwise, a PC-translation-specification exception is recognized.

After the ETE has been fetched, if the current PSW specifies the problem state, the current PSW-key mask in control register 3 is tested against the AKM field in the ETE to determine whether the program is authorized to access this entry. The AKM and PSW-key mask are ANDed, and, if the result is zero, a privileged-operation exception is recognized. The PSW-key mask in control register 3 remains unchanged. When PROGRAM CALL is executed in the supervisor state, the AKM field is ignored.

If the result of the AND of the AKM and the PSW-key mask is not zero, or if the CPU is in the supervisor state, the execution of the instruction continues.

If bit 128 of the ETE (T) is zero, the basic PROGRAM CALL operation is specified. If bit 128 of the ETE is one, the stacking PROGRAM CALL operation is specified.

Basic PROGRAM CALL

The following operations are performed when basic PROGRAM CALL is specified.

Bit 31 of the current PSW (the extended-addressing-mode bit) must equal bit 129 (G) of the ETE; otherwise, a special-operation exception is recognized.

In the 24-bit or 31-bit addressing mode, bits 97-126 of the PSW (bits 33-62 of the updated instruction address) are placed in bit positions 33-62 of general register 14, bit 32 of the PSW (the basic-addressing-mode bit) is placed in bit position 32 of the register, and bit 15 of the PSW (the problem-state bit) is placed in bit position 63 of the register. Bits 0-31 of the register remain unchanged.

In the 64-bit addressing mode, bits 64-126 of the PSW (bits 0-62 of the updated instruction address) are placed in bit positions 0-62 of general register 14, and bit 15 of the PSW (the problem-state bit) is placed in bit position 63 of the register.

In the 24-bit or 31-bit addressing mode, bits 32 and 33-62 of the ETE (A and the EIA), with a zero appended on the right of bits 33-62, are placed in PSW bit positions 32 and 97-127, respectively (the basic-addressing-mode bit and bits 33-63 of the instruction address). In the 64-bit addressing mode, bits 0-62 of the ETE, with a zero appended on the right, are placed in PSW bit positions 64-127 (the instruction address), and PSW bit 32 remains unchanged. In any addressing mode, bit 63 of the ETE (P) is placed in PSW bit position 15 (the problem-state bit).

The PSW-key mask, bits 32-47 of control register 3, is placed in bit positions 32-47 of general register 3, and the current PASN, bits 48-63 of control register 4, is placed in bit positions 48-63 of general register 3. Bits 0-31 of general register 3 remain unchanged.

Bits 96-111 of the ETE (the EKM) are ORed with the PSW-key mask, bits 32-47 of control register 3, and the result replaces the PSW-key mask in control register 3.

In the 24-bit or 31-bit addressing mode, bits 224-255 of the ETE (bits 32-63 of the entry parameter) are loaded into bit positions 32-63 of general register 4, and bits 0-31 of the register remain unchanged. In the 64-bit addressing mode, bits 192-255 of the ETE (the entry parameter), are loaded into bit positions 0-63 of general register 4.

Stacking PROGRAM CALL

The following operations are performed when stacking PROGRAM CALL is specified.

The stacking process is performed to form a linkage-stack program-call state entry and place the following information in the state entry: current PSW (with an unpredictable PER mask), PSW-key mask, PASN, SASN, EAX, called-space identification, an indication of whether the resulting addressing mode is the 64-bit mode, numeric part of the program-call number, contents of general registers 0-15, and contents of access registers 0-15, and, if ASN-and-LX reuse is enabled, PASTEIN and SASTEIN. This is described in "Stacking Process" on page 5-80. The entry-type code in the state entry is 0001101 binary.

When bit 129 of the ETE (G) is zero, bit 31 of the PSW (the extended-addressing-mode bit) is set to zero, and bit 32 of the ETE (A) is placed in bit position 32 of the PSW (the basic-addressing-mode bit). (The addressing mode is set to the 24-bit mode if bit 32 is zero or to the 31-bit mode if bit 32 is one.) When bit 129 of the ETE is one, bits 31 and 32 of the PSW are set to one. (The 64-bit addressing mode is set.)

When the resulting addressing mode is the 24-bit or 31-bit mode, bits 33-62 of the ETE (the EIA), with 33 leftmost and one rightmost zeros appended, are placed in PSW bit positions 64-127 (the instruction address). When the resulting addressing mode is the 64-bit mode, bits 0-62 of the ETE (the EIA), with one rightmost zero appended, are placed in PSW bit positions 64-127.

Bit 63 of the ETE (P) is placed in PSW bit position 15 (the problem-state bit).

When bit 131 of the ETE (K) is zero, bits 8-11 of the PSW (the PSW key) remain unchanged. When bit 131 of the ETE is one, bits 136-139 of the ETE (the EK) replace the PSW key in the PSW.

When bit 132 of the ETE (M) is zero, bits 96-111 of the ETE (the EKM) are ORed with the PSW-key mask, bits 32-47 of control register 3, and the result replaces the PSW-key mask in control register 3. When bit 132 of the ETE is one, bits 96-111 of the ETE replace the PSW-key mask in control register 3.

When bit 133 of the ETE (E) is zero, the EAX, bits 32-47 of control register 8, remains unchanged. When bit 133 of the ETE is one, bits 144-159 of the

ETE (the EEAX) replace the EAX in control register 8.

When bit 134 of the ETE (C) is zero, bits 16 and 17 of the PSW (the address-space-control bits) are set to 00 binary (primary-space mode). When bit 134 of the ETE is one, the address-space-control bits in the PSW are set to 01 binary (access-register mode).

When the resulting addressing mode is the 24-bit or 31-bit mode, bits 224-255 of the ETE (bits 32-63 of the entry parameter) are loaded into bit positions 32-63 of general register 4, and bits 0-31 of this register remain unchanged. When the resulting addressing mode is the 64-bit mode, bits 192-255 of the ETE (the entry parameter), are loaded into bit positions 0-63 of general register 4.

Key-controlled protection does not apply to references to the linkage stack, but low-address and DAT protection do apply.

PROGRAM CALL to Current Primary (PC-cp)

If bits 80-95 of the ETE (the ASN), are zeros, PROGRAM CALL to current primary (PC-cp) is specified, and the execution of the instruction is completed after the operations described in “PROGRAM CALL PC-Number Translation” and either “Basic PROGRAM CALL” or “Stacking PROGRAM CALL” have been performed and the following operations have been performed.

The current PASN, bits 48-63 of control register 4, is placed in bit positions 48-63 of control register 3 to become the current SASN.

The current PASCE in control register 1 is placed in control register 7 to become the current SASCE.

If ASN-and-LX reuse is enabled, the current PASTEIN, bits 0-31 of control register 4, is placed in bit positions 0-31 of control register 3 to become the current SASTEIN.

The basic PC-cp operation is depicted in parts 1-3 of Figure 10-58 on page 10-90. The stacking PC-cp operation is depicted in parts 1, 4, and 5 of the figure.

PROGRAM CALL with Space Switching (PC-ss)

If the ASN in the ETE is nonzero, PROGRAM CALL with space switching (PC-ss) is specified, and the execution of the instruction is completed after the

operations described in “PROGRAM CALL PC-Number Translation” and either “Basic PROGRAM CALL” or “Stacking PROGRAM CALL” have been performed and the following operations have been performed.

Bits 80-95 of the ETE (the ASN) are placed in bit positions 48-63 of control register 4 as the new PASN.

Bits 161-185 of the ETE, with six zeros appended on the right, are used as the real address of the ASTE designated by the new PASN. An ASX-translation exception is recognized if bit 0 of the ASTE is one.

Bits 64-127 of the ASTE (the ASCE) are placed in control register 1 as the new PASCE.

Bits 32-47 of the ASTE (the AX) are placed in bit positions 32-47 of control register 4 as the new authorization index.

If ASN-and-LX reuse is enabled, bits 352-383 of the ASTE (the ASTEIN) are placed in bit positions 0-31 of control register 4 as the new PASTEIN.

Bits 33-57 of the ASTE address are placed in bit positions 33-57 of control register 5 as the new primary-ASTE origin, and zeros are placed in bit positions 32 and 58-63. Bits 0-31 of the register remain unchanged.

In basic PROGRAM CALL, or in stacking PROGRAM CALL when bit 135 of the ETE (S) is zero, the PASN existing before the PASN is replaced from the ETE is placed in bit positions 48-63 of control register 3 to become the current SASN, and the PASCE existing before the PASCE is replaced from the ASTE is placed in control register 7 to become the current SASCE. If ASN-and-LX reuse is enabled, the PASTEIN existing before the PASTEIN is replaced from the ASTE is placed in bit positions 0-31 of control register 3 to become the current SASTEIN. (The SASN and SASCE are set equal to the old PASN and PASCE, respectively, and, if ASN-and-LX reuse is enabled, the SASTEIN is set equal to the old PASTEIN.)

In stacking PROGRAM CALL when bit 135 of the ETE (S) is one, the SASN is replaced by the PASN after the PASN is replaced from the ETE, and the SASCE is replaced by the PASCE after the PASCE is replaced from the ASTE. If ASN-and-LX reuse is enabled, the SASTEIN is replaced by the PASTEIN after the PASTEIN is replaced from the ASTE. (The

SASN and SASCE are set equal to the new PASN and PASCE, respectively, and, if ASN-and-LX reuse is enabled, the SASTEIN is set equal to the new PASTEIN.)

The description in this paragraph applies to use of the subspace-group facility. After the new PASCE has been placed in control register 1 and the new primary-ASTE origin has been placed in control register 5, if (1) the subspace-group-control bit, bit 54, in the PASCE is one, (2) the dispatchable unit is subspace active, and (3) the primary-ASTE origin designates the ASTE for the base space of the dispatchable unit, then bits 0-55 and 58-63 of the PASCE are replaced by the same bits of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. This replacement occurs before a replacement of the SASCE in control register 7 by the PASCE. Further details are in “Subspace-Replacement Operations” on page 5-65.

The PC-ss operation is depicted in parts 1 and 4-6 of Figure 10-58 on page 10-90.

PROGRAM CALL Serialization

For both the PC-cp and PC-ss operations, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed. However, it is unpredictable whether or not a store into a trace-table entry or linkage-stack entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store.

Special Conditions

The basic PROGRAM CALL operation can be performed successfully only when (1) the CPU is in the primary-space mode at the beginning of the operation, (2) the subsystem-linkage control, bit 0 of the linkage-table designation or linkage-first-table designation in the current primary ASN-second-table entry, is one, and (3) the extended-addressing-mode bit, bit 31 of the current PSW, equals the entry-extended-addressing-mode bit, bit 129 of the entry-table entry. Stacking PROGRAM CALL can be performed successfully only when the CPU is in the primary-space mode or access-register mode at the beginning of the operation and the subsystem-linkage control is one. In addition, PC-ss can be performed successfully only when the ASN-translation control, bit 44 of control register 14, is one. If any of these rules is violated, a special-operation exception is recognized.

When ASN-and-LX reuse is enabled and the LSTESN in the linkage-second-table entry is non-zero, that LSTESN must be equal to the LSTESN specified in bit positions 0-31 of general register 15; otherwise, an LSTE-sequence exception is recognized.

A stack-full or stack-specification exception may be recognized during the stacking process.

When, for PC-ss, the primary space-switch-event-control bit, bit 57 of control register 1, is one either before or after the execution of the instruction, a space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PC-ss operation if a PER event is reported.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-57 on page 10-88.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch or store, except for key-controlled protection, linkage-stack entry)
- Addressing (linkage-table or linkage-first-table designation in primary ASN-second-table entry; linkage-table entry; linkage-first-table entry; linkage-second-table entry, entry-table entry; ASN-second-table entry, PC-ss only)
- ASX translation (PC-ss only)
- EX translation
- LFX translation
- LSTE sequence
- LSX translation
- LX translation
- PC-translation specification
- Privileged operation (AND of AKM and PSW-key mask is zero in the problem state)
- Space-switch event (PC-ss only)
- Special operation
- Stack full (stacking PC only)
- Stack specification (stacking PC only)
- Subspace replacement (PC-ss only)
- Trace

Programming Note: The effective address from which a PC number is derived is subject to the

addressing mode in the current PSW. Therefore, bits 0-39 of the effective address in the 24-bit addressing mode, and bits 0-32 in the 31-bit addressing mode, are treated as containing zeros.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off or the CPU being in secondary-space mode or home-space mode.
8.A	Trace exceptions.
8.B.1	Addressing exception for access to linkage-table designation or linkage-first-table designation in primary ASN-second-table entry.
8.B.2	Special-operation exception due to subsystem-linkage control in linkage-table designation or linkage-first-table designation being zero. Note: The LFX-translation, LSX-translation, and LSTE-sequence exceptions can be recognized only if ASN-and-LX reuse is enabled, and the LX-translation exception cannot be recognized in that case.
8.B.3	LX-translation or LFX-translation exception due to linkage-table or linkage-first-table entry, respectively, being outside table.
8.B.4	Addressing exception for access to linkage-table or linkage-first-table entry.
8.B.5	LX-translation or LFX-translation exception due to I bit (bit 0) in linkage-table or linkage-first-table entry, respectively, being one.
8.B.6	Addressing exception for access to linkage-second-table entry.
8.B.7	LSX-translation exception due to I bit (bit 0) in linkage-second-table entry being one.
8.B.8	LSTE-sequence exception due to LSTE sequence number in linkage-second-table entry being nonzero and not equal to bits 0-31 of general register 15.
8.B.9	EX-translation exception due to entry-table entry being outside table.
8.B.10	Addressing exception for access to entry-table entry.
8.B.11	Special-operation exception due to the CPU being in access-register mode or extended-addressing-mode bit, bit 31 of PSW, not being equal to entry-extended-addressing-mode bit, bit 129 of entry-table entry (basic PC only).
8.B.12	PC-translation-specification exception due to invalid combination (bits 33-39 not zeros when resulting addressing mode is 24 bit) in entry-table entry.
8.B.13	Privileged-operation exception due to zero result from ANDing PSW-key mask and AKM in the problem state.
8.B.14	Special-operation exception due to ASN-translation control, bit 44 of control register 14, being zero (PC-ss only).
8.B.15	Addressing exception for access to ASN-second-table entry (PC-ss only).
8.B.16	ASX-translation exception due to I bit (bit 0) in ASN-second-table entry being one (PC-ss only). Note: Subspace-replacement exceptions, which are not shown in detail in this figure, can occur with any priority after 8.B.16 and before 9.

Figure 10-57. Priority of Execution: PROGRAM CALL (Part 1 of 2)

8.B.17	Access exceptions (fetch) for entry descriptor of the current linkage-stack entry (stacking PC only).
	Note: Exceptions 8.B.18-8.B.23 can occur only if there is not enough remaining free space in the current linkage-stack section.
8.B.18	Stack-specification exception due to remaining-free-space value in current linkage-stack entry not being a multiple of 8.
8.B.19	Access exceptions (fetch) for second word of the trailer entry of the current section. The entry is presumed to be a trailer entry; its entry-type field is not examined (stacking PC only).
8.B.20	Stack-full exception due to forward-section validity bit in the trailer entry being zero (stacking PC only).
8.B.21	Access exceptions (fetch) for entry descriptor of the header entry of the next section (stacking PC only). This entry is presumed to be a header entry; its entry-type field is not examined.
8.B.22	Stack-specification exception due to not enough remaining free space in the next section (stacking PC only).
8.B.23	Access exceptions (store) for second word of the header entry of the next section. If there is no exception, the header is now called the current entry.
8.B.24	Access exceptions (store) for entry descriptor of the current entry and for the new state entry (stacking PC only).
9.	Space-switch event (PC-ss only).

Figure 10-57. Priority of Execution: PROGRAM CALL (Part 2 of 2)

Basic PC-cp and PC-ss in 24-Bit or 31-Bit Addressing Mode

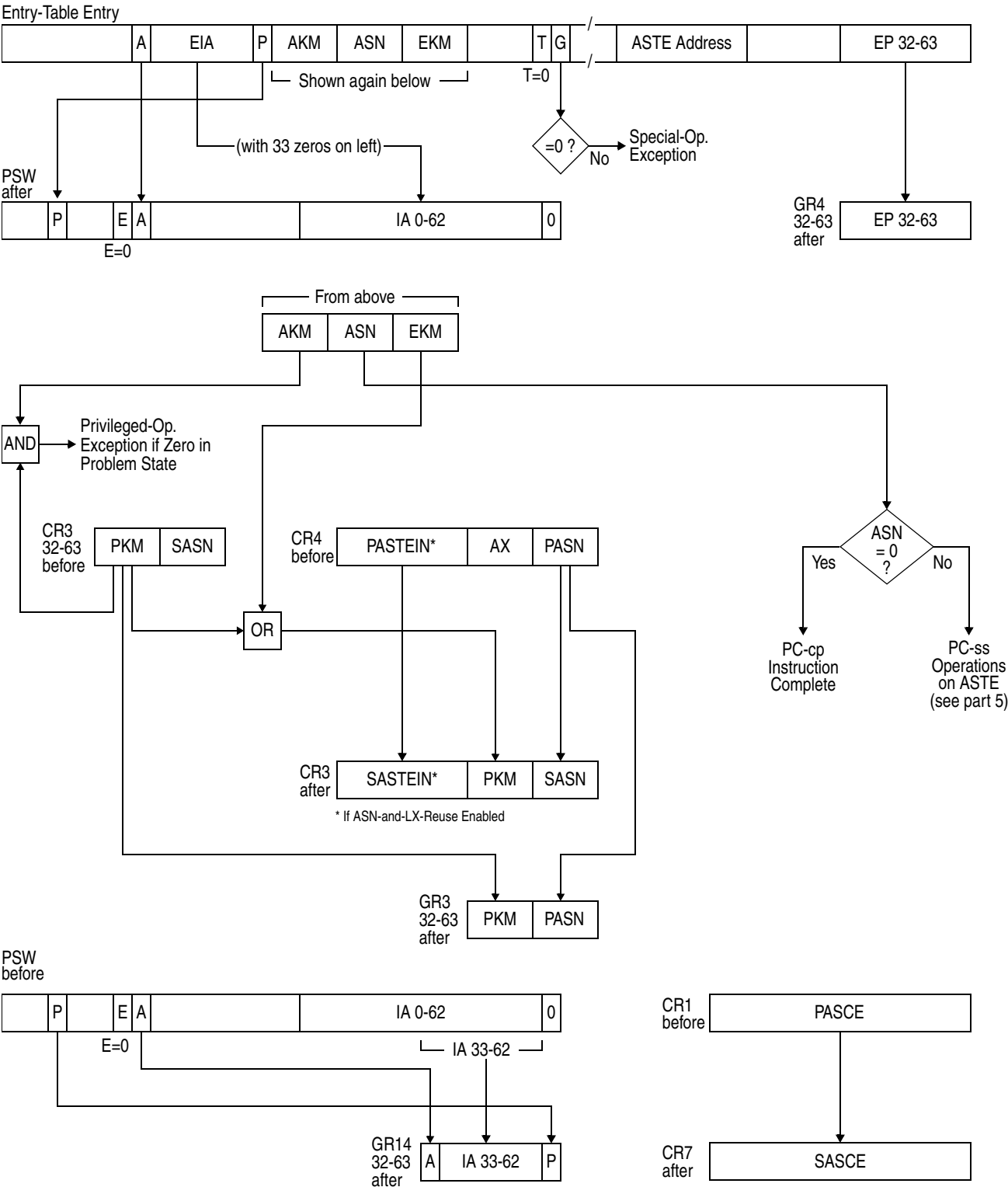


Figure 10-58. Execution of PROGRAM CALL (Part 1 of 5).

Basic PC-cp and PC-ss in 64-Bit Addressing Mode

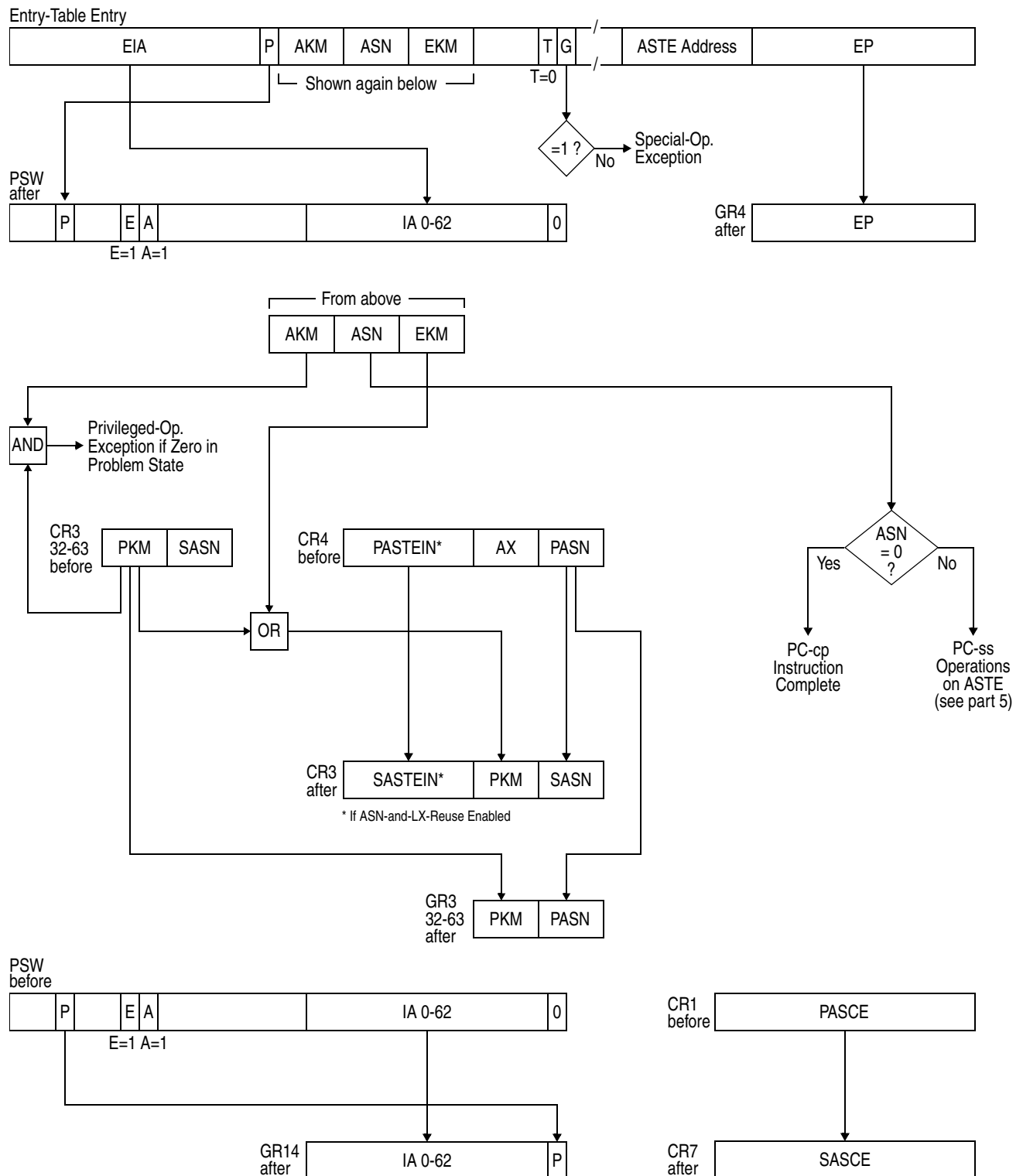


Figure 10-58. Execution of PROGRAM CALL (Part 2 of 5).

Stacking PC-cp and PC-ss from 24-Bit or 31-Bit Addressing Mode to 64-Bit Addressing Mode

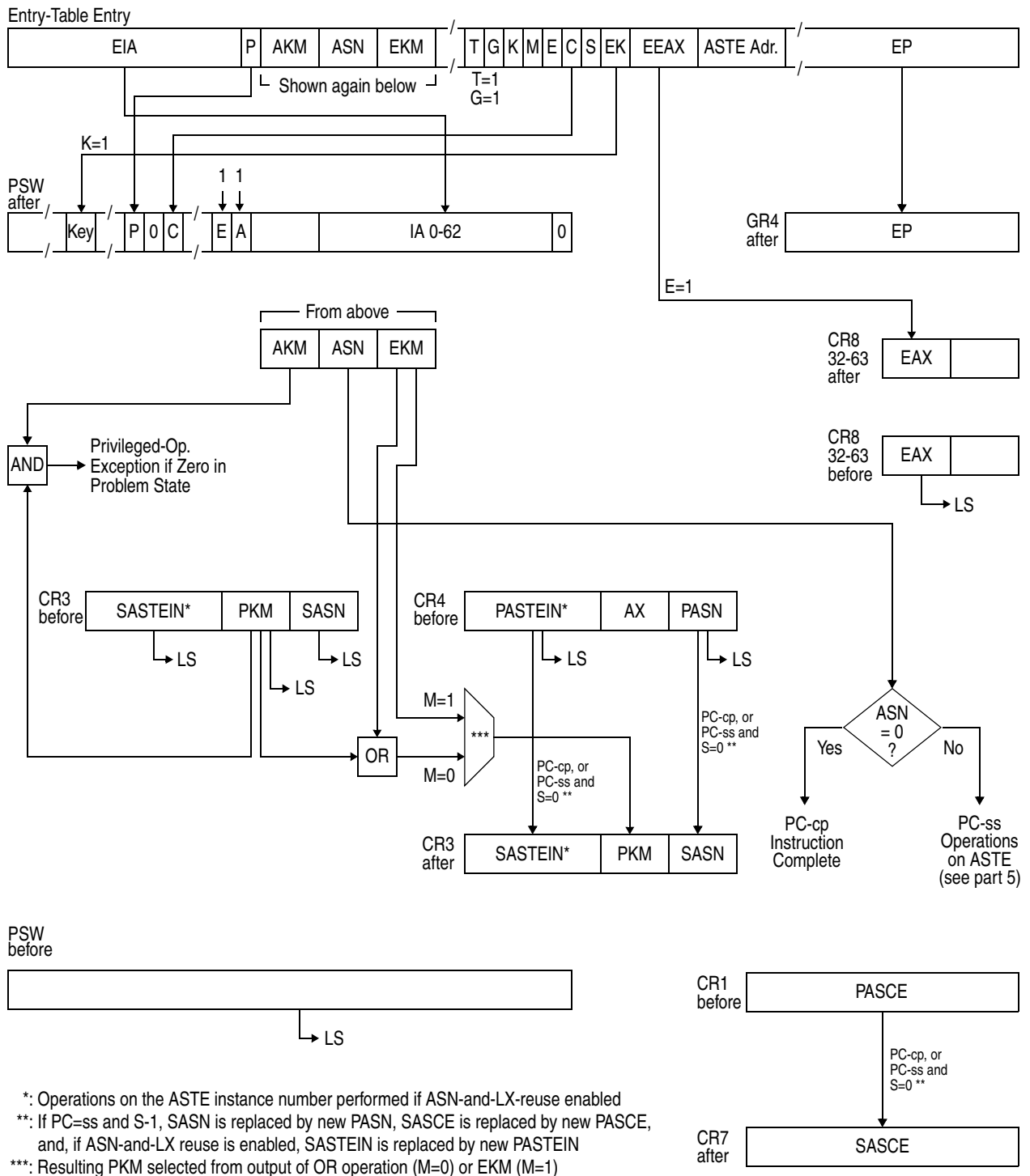


Figure 10-58. Execution of PROGRAM CALL (Part 3 of 5).

Stacking PC-cp and PC-ss from 64-Bit Addressing Mode to 24-Bit or 31-Bit Addressing Mode

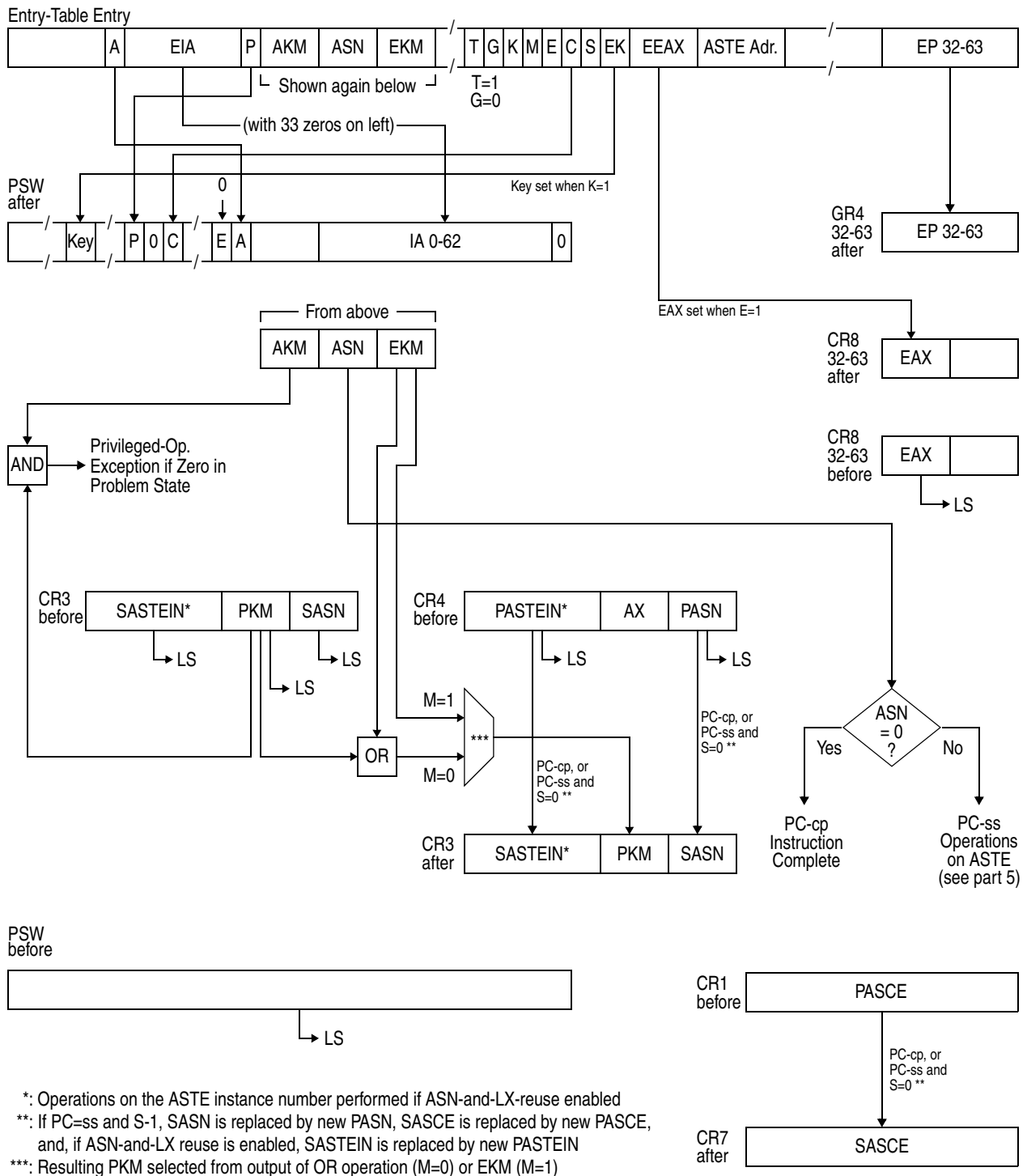
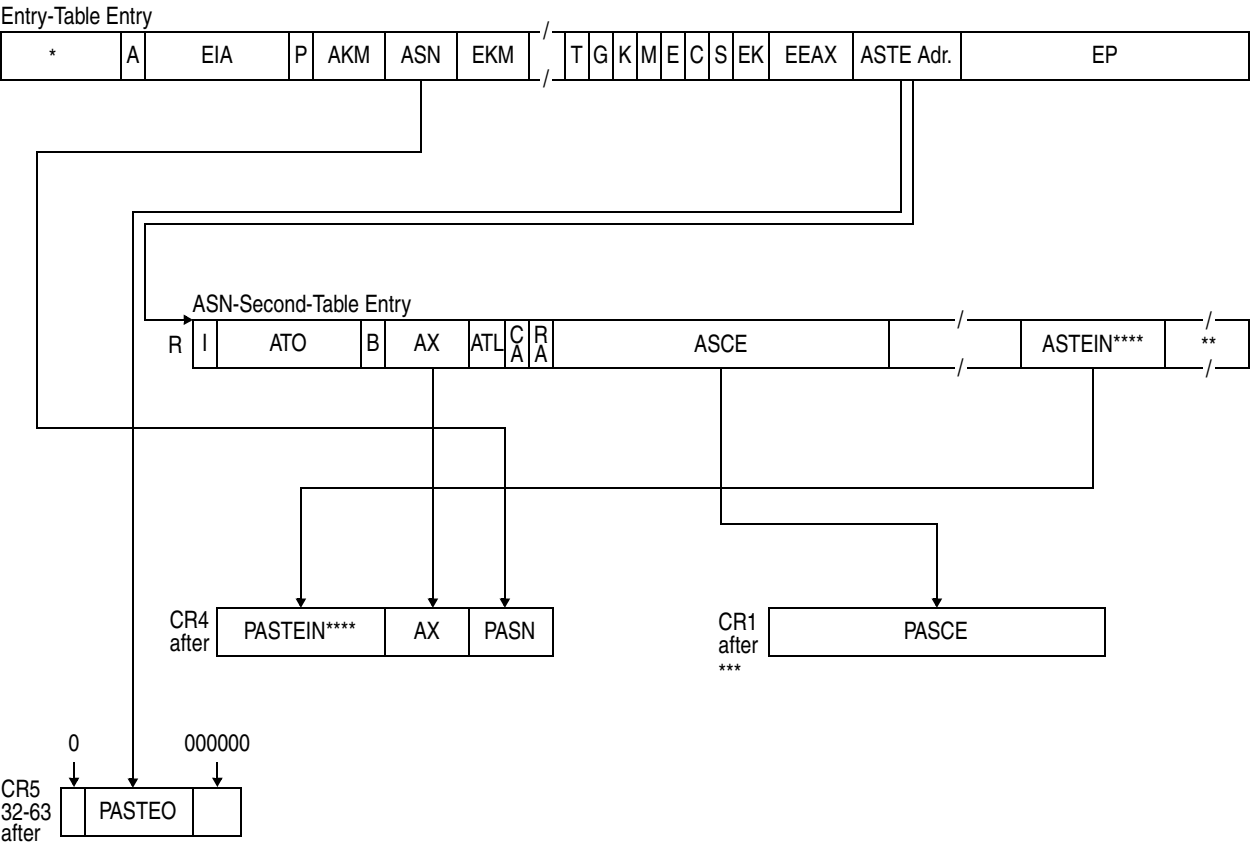


Figure 10-58. Execution of PROGRAM CALL (Part 4 of 5).

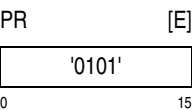
Operations on ASN-Second-Table-Entry for PC-ss



R: Address is real
*: First word and A of ETE are bits 0-32 of EIA if resulting addressing mode is the 64-bit mode.
**: ASTE is 64 bytes; selected fields and the last 16 bytes are not shown.
***: Bits 0-55 and 58-63 of PASCE may be replaced from a subspace ASCE
****: Operations on the ASTE instance number performed if ASN-and-LX-reuse enabled

Figure 10-58. Execution of PROGRAM CALL (Part 5 of 5).

PROGRAM RETURN



The PSW, except for the PER-mask bit, saved in the last linkage-stack state entry is restored as the current PSW. The PER mask in the current PSW remains unchanged. The contents of general registers 2-14 and access registers 2-14 also are restored from the state entry. When the entry-type code in the entry descriptor of the state entry is 0001101 binary, indicating a program-call state entry, (1) the primary ASN (PASN), secondary ASN (SASN), PSW-key

mask (PKM), and extended authorization index (EAX) in the control registers also are restored from the state entry and (2) if the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, the primary ASTE instance number (PASTEIN) and secondary ASTE instance number (SASTEIN) in the control registers also are restored from the state entry. When the entry-type code is 0001100 binary, indicating a branch state entry, the current PASN, SASN, PKM, EAX, PASTEIN, and SASTEIN remain unchanged.

The last state entry is located, and information in it is restored, as described in “Unstacking Process” on page 5-82. The state entry is logically deleted from

the linkage stack, and the linkage-stack-entry address in control register 15 is replaced by the address of the next preceding state or header entry. This also is described in “Unstacking Process”.

When the state entry is a program-call state entry, it causes a space-switching operation to occur if it contains a PASN that is not equal to the current PASN. When the state entry contains a PASN that is equal to the current PASN, the operation is called PROGRAM RETURN to current primary (PR-cp); when the state entry contains a PASN that is not equal to the current PASN, the operation is called PROGRAM RETURN with space switching (PR-ss). PASN translation occurs in PR-ss. SASN translation and authorization may occur in either PR-cp or PR-ss. The terms PR-cp and PR-ss do not apply when the state entry is a branch state entry.

When the ASN-and-LX-reuse facility is installed and enabled and PASN or SASN translation occurs, the PASTEIN or SASTEIN, respectively, saved in the state entry is compared to the ASTEIN in the located ASTE.

Key-controlled protection does not apply to accesses to the linkage stack, but low-address and DAT protection do apply.

The sections “PASN Translation,” “SASN Translation,” “SASN Authorization,” and “PROGRAM RETURN Serialization” apply only when the unstacked state entry is a program-call state entry. The functions described in those sections are not performed when the state entry is a branch state entry.

The actions involving the PASTEIN and SASTEIN occur only when ASN-and-LX reuse is enabled by the ASN-and-LX reuse control in control register 0.

PASN Translation

If the new PASN is equal to the old PASN in bit positions 48-63 of control register 4, PASN translation is not performed, the PASTEIN in control register 4 remain as restored from the state entry, and the PASCE in control register 1 and primary-ASTE origin (PASTE0) in control register 5 are not changed. In this case, there is not a test of whether the new PASTEIN is equal to the old PASTEIN.

If the new PASN is not equal to the old PASN, the new PASN replaces the PASN in bit positions 48-63 of control register 4 and is translated to locate a

64-byte ASTE. The ASN table-lookup process is described in “ASN Translation” on page 3-26. The exceptions associated with ASN translation are collectively called ASN-translation exceptions. These exceptions and their priority are described in Chapter 6, “Interruptions.”

If ASN-and-LX reuse is enabled, the PASTEIN saved in bytes 180-183 of the state entry must equal the ASTEIN in bit positions 352-383 of the located ASTE; otherwise, an ASTE-instance exception is recognized.

Bits 64-127 of the ASTE are placed in control register 1 as the new PASCE.

Bits 32-47 of the ASTE are placed in bit positions 32-47 of control register 4 as the new AX. The PASN and PASTEIN in control register 4 remain as restored from the state entry.

Bits 33-57 of the ASTE address are placed in bit positions 33-57 of control register 5 as the new primary-ASTE origin, and zeros are placed in bit positions 32 and 58-63. Bits 0-31 of this register remain unchanged.

The description in this paragraph applies to use of the subspace-group facility when PASN translation has occurred. If (1) the subspace-group-control bit, bit 54, in the new PASCE is one, (2) the dispatchable unit is subspace active, and (3) the new primary-ASTE origin designates the ASTE for the base space of the dispatchable unit, then bits 0-55 and 58-63 of the new PASCE in control register 1 are replaced by the same bits of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. This replacement occurs, in the case when the new SASN is equal to the new PASN, before the SASCE is set equal to the PASCE. Further details are in “Subspace-Replacement Operations” on page 5-65.

SASN Translation

If the new SASN is equal to the new PASN, the SASCE in control register 7 is set equal to the new PASCE in control register 1. The SASTEIN, PKM, and SASN in control register 3 remain as restored from the state entry. In this case, there is not a test of whether the new SASTEIN is equal to the new PASTEIN.

If the new SASN is not equal to the new PASN, the new SASN is translated to locate a 64-byte ASTE.

If ASN-and-LX reuse is enabled, the SASTEIN saved in bytes 176-179 of the state entry must equal the ASTEIN in bit positions 352-383 of the located ASTE; otherwise, an ASTE-instance exception is recognized.

Bits 64-127 of the ASTE are placed in control register 7 as the new SASCE.

Control register 3 remains as restored from the state entry.

SASN Authorization

If the new SASN is not equal to the new PASN, the authority-table origin (ATO) from the ASTE for the new SASN is used as the base for a third table lookup. The new authorization index, bits 32-47 of control register 4, is used, after it has been checked against the authority-table length, as the index to locate the entry in the authority table. The authority-table lookup is described in “ASN Authorization” on page 3-31.

The description in this paragraph applies to use of the subspace-group facility when SASN translation and authorization have occurred. If (1) the subspace-group-control bit, bit 54, in the new SASCE is one, (2) the dispatchable unit is subspace active, and (3) the ASTE origin obtained by SASN translation designates the ASTE for the base space of the dispatchable unit, then bits 0-55 and 58-63 are replaced by the same bits of the ASCE of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. Further details are in “Subspace-Replacement Operations” on page 5-65.

PROGRAM RETURN Serialization

When the unstacked state entry is a program-call state entry, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed. However, it is unpredictable whether or not a store into a trace-table entry or linkage-stack entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store.

Special Conditions

The instruction can be executed successfully only when the CPU is in the primary-space mode or access-register mode at the beginning of the operation. In addition, the ASN-translation process can be

performed, for either the PASN or the SASN, only when the ASN-translation control, bit 44 of control register 14, is one. If either of these rules is violated, a special-operation exception is recognized.

A stack-empty, stack-operation, stack-specification, or stack-type exception may be recognized during the unstacking process.

If ASN-and-LX reuse is enabled, the restored PASTEIN must equal the ASTEIN in the located ASTE if PASN translation is performed, and the restored SASTEIN must equal the ASTEIN in the located ASTE if SASN translation is performed; otherwise, an ASTE-instance exception is recognized.

When, for PR-ss, the primary space-switch-event control, bit 57 of control register 1, is one either before or after the execution of the instruction, a space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PR-ss operation if a PER event is reported.

The PSW which is to be loaded by the instruction is not checked for validity before it is loaded. However, after loading, a specification exception is recognized, and a program interruption occurs, if any of bits 0, 2-4, 12, 24-30, and 33-63 of the PSW is a one, if bits 31 and 32 are zero and one, respectively, and bits 64-96 are not all zeros, if bits 31 and 32 are both zero and bits 64-103 are not all zeros, or if bits 31 and 32 are one and zero, respectively. In these cases, the operation is completed, and the resulting instruction-length code is 0. The specification exception, which in this case is listed as a program exception in this instruction, is described in “Early Exception Recognition” on page 6-8. It may be considered as occurring early in the process of preparing to execute the following instruction.

If a space-switch event is indicated and the PSW that was loaded by the instruction is invalid because of a reason described in the preceding paragraph, it is unpredictable whether the resulting instruction-length code is 0 or 1, or 0 or 2 if EXECUTE was used, or 0 or 3 if EXECUTE RELATIVE LONG was used.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-59 on page 10-97.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.	Special-operation exception due to DAT being off or the CPU being in secondary-space mode or home-space mode.
8.A	Trace exceptions.
8.B.1	Access exceptions (fetch) for entry descriptor of the current linkage-stack entry.
8.B.2	Stack-type exception due to current entry not being a state entry or header entry. Note: Exceptions 8.B.3-8.B.7 can occur only if the current entry is a header entry.
8.B.3	Stack-operation exception due to unstack-suppression bit in the header entry being one.
8.B.4	Access exceptions (fetch) for second word of the header entry.
8.B.5	Stack-empty exception due to backward stack-entry validity bit in the header entry being zero.
8.B.6	Access exceptions (fetch) for entry descriptor of preceding entry, which is the entry designated by the backward stack-entry address in the current (header) entry.
8.B.7	Stack-specification exception due to preceding entry being a header entry.
8.B.8	Stack-type exception due to preceding entry not being a state entry.
8.B.9	Stack-operation exception due to unstack-suppression bit being one in the state entry.
8.B.10	Access exceptions (fetch) for the state entry, and access exceptions (store) for entry descriptor of the entry preceding the state entry. Note: Exceptions 8.B.11-8.B.15 and the event 9 can occur only if the state entry is a program-call state entry.
8.B.11	Special-operation exception due to the ASN-translation control, bit 44 of control register 14, being zero (if PASN or SASN translation occurs).
8.B.12	ASN-translation exceptions (if PASN translation occurs).
8.B.13	ASTE-instance exception due to new PASTEIN not being equal to ASTEIN in ASN-second-table entry located by PASN translation (if ASN-and-LX reuse enabled). Note: Subspace-replacement exceptions for replacement of bits in the PASCE, which are not shown in detail in this figure, can occur with any priority after 8.B.13 and before 9.
8.B.14	ASN-translation exceptions (if SASN translation occurs).
8.B.15	ASTE-instance exception due to new SASTEIN not being equal to ASTEIN in ASN-second-table entry located by SASN translation (if ASN-and-LX reuse enabled). Note: Subspace-replacement exceptions for replacement of bits in the SASCE, which are not shown in detail in this figure, can occur with any priority after 8.B.15 and before 9.
8.B.16	Secondary-authority exception due to authority-table entry being outside table (if SASN translation occurs).
8.B.17	Addressing exception for access to authority-table entry (if SASN translation occurs).
8.B.18	Secondary-authority exception due to S bit in authority-table entry being zero (if SASN translation occurs).
9.	Space-switch event (PR-ss only).
10.	Specification exception due to any PSW error of the type that causes an immediate interruption.

Figure 10-59. Priority of Execution: PROGRAM RETURN

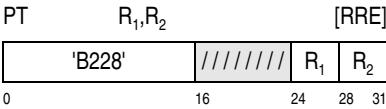
Resulting Condition Code: The code is set as specified in the new PSW loaded.

Program Exceptions:

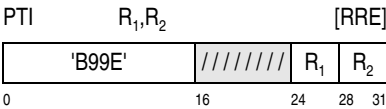
- Access (fetch and store, except key-controlled protection, linkage-stack entry)
- Addressing (authority-table entry, if SASN translation occurs)
- ASN translation (if PASN or SASN translation occurs)
- ASTE instance (if ASN-and-LX reuse is enabled and PASN or SASN translation occurs)
- Secondary authority (if SASN translation occurs)
- Space-switch event
- Special operation
- Specification
- Stack empty
- Stack operation
- Stack specification
- Stack type
- Subspace replacement (if PASN or SASN translation occurs)
- Trace

Programming Note: Because PROGRAM CALL cannot be executed successfully in the secondary-space or home-space mode, PROGRAM RETURN is not intended to load a PSW specifying one of these translation modes. PROGRAM RETURN, unlike SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST, does not recognize a space-switch event because of loading a PSW that specifies the home-space mode.

PROGRAM TRANSFER



PROGRAM TRANSFER WITH INSTANCE



Note: In this instruction definition, the name “PROGRAM TRANSFER (WITH INSTANCE)” refers to the PROGRAM TRANSFER instruction and the PROGRAM TRANSFER WITH INSTANCE instruction.

Bits 32-63 of general register R₁ are used as the new values for the PSW-key mask, the PASN, and the SASN. Bits 32-63 or 0-63 of general register R₂, depending on the current addressing mode, are used as the new values for the problem-state bit, basic-addressing-mode bit, and instruction address in the current PSW. In the PROGRAM TRANSFER WITH INSTANCE operation, bits 0-31 of general register R₁ are an ASTEIN and are compared against the new value of the PASTEIN if the PASN is changed. In the PROGRAM TRANSFER operation, bits 0-31 of general register R₁ are ignored.

The format of general registers R₁ and R₂ are shown in Figure 10-60.

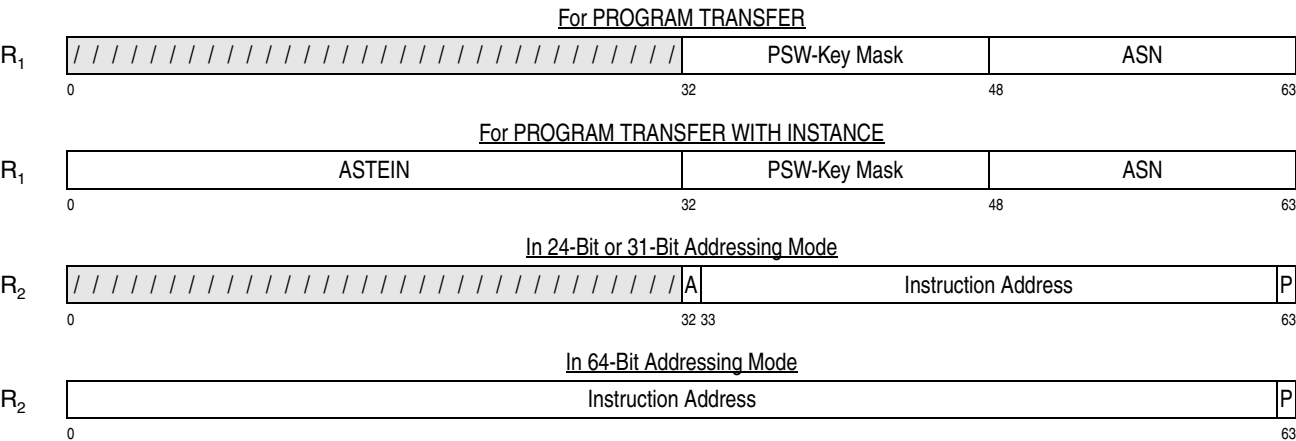


Figure 10-60. Register Contents for PROGRAM TRANSFER (WITH INSTANCE)

When the contents of bit positions 48-63 of general register R_1 are equal to the current PASN, the operation is called PROGRAM TRANSFER (WITH INSTANCE) to current primary (PT-cp or PTI-cp); when the fields are not equal, the operation is called PROGRAM TRANSFER (WITH INSTANCE) with space switching (PT-ss or PTI-ss).

The contents of general register R_2 are used to update the problem-state bit and the instruction address in the current PSW and, in the 24-bit or 31-bit addressing mode, also the basic-addressing-mode bit in the current PSW. Bit 63 of general register R_2 is placed in the problem-state bit position, PSW bit position 15, unless the operation would cause PSW bit 15 to change from one to zero (problem state to supervisor state). If such a change would occur, a privileged-operation exception is recognized.

In the 24-bit or 31-bit addressing mode, bit 32 of general register R_2 replaces the basic-addressing-mode bit, bit 32 of the current PSW, and bits 33-62 of the register, with one rightmost zero appended, replace bits 33-63 of the instruction address in the PSW, bits 97-127 of the PSW. In the 64-bit addressing mode, bits 0-62 of general register R_2 , with one rightmost zero appended, replace the instruction address, and the basic-addressing-mode bit remains unchanged.

Bits 32-47 of general register R_1 are ANDed with the PSW-key mask, bits 32-47 of control register 3, and the result replaces the PSW-key mask. Bits 0-31 of general register R_1 are ignored.

In any of the PT-cp, PTI-cp, PT-ss, and PTI-ss operations, the ASN specified by bits 48-63 of general register R_1 replaces the SASN in control register 3, and the SASCE in control register 7 is replaced by the final contents of control register 1.

In the PROGRAM TRANSFER operation, if the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, the SASTEIN in control register 3 is replaced by the final value of the PASTEIN in control register 4. This replacement occurs in the PROGRAM TRANSFER WITH INSTANCE operation regardless of the value of the ASN-and-LX-reuse control.

PROGRAM TRANSFER (WITH INSTANCE) to Current Primary (PT-cp or PTI-cp)

The PT-cp operation is depicted in part 1 of Figure 10-62 on page 10-103. The PTI-cp operation is depicted in part 1 of Figure 10-63 on page 10-105. The PT-cp or PTI-cp operation is completed when the common portion of the PROGRAM TRANSFER (WITH INSTANCE) operation, described above, is completed. The PASTEIN, authorization index, ASN, primary ASCE, and contents of control register 5 (primary-ASN-second-table-entry origin) are not changed by PT-cp or PTI-cp. In this case in the PTI-cp operation, there is not a test of whether the current PASTEIN equals the ASTEIN specified in bit positions 0-31 of general register R_1 ; the ASTEIN is ignored.

PROGRAM TRANSFER (WITH INSTANCE) with Space Switching (PT-ss or PTI-ss)

If the ASN in bit positions 48-63 of general register R_1 is not equal to the current PASN, a PROGRAM TRANSFER (WITH INSTANCE) with space switching (PT-ss or PTI-ss) operation is specified, and the ASN is translated by means of a two-level table lookup.

The PT-ss operation is depicted in parts 1 and 2 of Figure 10-62 on page 10-103. The PTI-ss operation is depicted in parts 1 and 2 of Figure 10-63 on page 10-105. The PT-ss or PTI-ss operation is completed as follows.

In PT-ss or PTI-ss, the contents of bit positions 48-63 of general register R_1 are used as an ASN, which is translated by means of a two-level table lookup.

Bits 48-57 of general register R_1 are a 10-bit AFX that is used to select an entry from the ASN first table. Bits 58-63 are a six-bit ASX that is used to select an entry from the ASN second table. The ASN table-lookup process is described in "ASN Translation" on page 3-26. The exceptions associated with ASN translation are collectively called "ASN-translation exceptions." These exceptions and their priority are described in Chapter 6, "Interruptions."

In PT-ss if the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, the reusable-ASN bit, bit 63, in the located ASN-second-table entry (ASTE) must be zero; otherwise, a special-operation exception is recognized. In PTI-ss, regardless of the ASN-and-LX-reuse control, the controlled-ASN bit, bit 62, in the ASTE must be zero if the CPU is in the problem state at the beginning of the operation; otherwise, a special-operation exception is rec-

ognized. Also in PTI-ss, and regardless of the ASN-and-LX-reuse control and the reusable-ASN bit, the ASTEIN in bit positions 0-31 of general register R_1 must equal the ASTEIN in bit positions 352-383 of the ASTE; otherwise, an ASTE-instance exception is recognized.

The authority-table origin from the ASN-second-table entry (ASTE) is used as the base for a third table lookup. The current authorization index, bits 32-47 of control register 4, is used, after it has been checked against the authority-table length, as the index to locate the entry in the authority table. The authority-table lookup is described in “ASN Authorization” on page 3-31.

The PT-ss or PTI-ss operation is completed by placing bits 64-127 of the ASTE in control register 1 as the new PASCE and in control register 7 as the new SASCE. The contents of bit positions 32-47 of the ASTE replace the authorization index in bit positions 32-47 of control register 4. Bits 33-57 of the ASTE address are placed in bit positions 33-57 of control register 5 as the new primary-ASTE origin, and zeros are placed in bit positions 32 and 58-63. Bits 0-31 of this register remain unchanged. The ASN, bits 48-63 of general register R_1 , replaces the SASN and PASN in bit positions 48-63 of control registers 3 and 4. In PT-ss if ASN-and-LX reuse is enabled, and in PTI-ss regardless of that enablement, the ASTEIN in the ASTE replaces the SASTEIN and PASTEIN in bit positions 0-31 of control registers 3 and 4.

The description in this paragraph applies to use of the subspace-group facility. After the new PASCE has been placed in control register 1 and the new primary-ASTE origin has been placed in control register 5, if (1) the subspace-group-control bit, bit 54, in the PASCE is one, (2) the dispatchable unit is subspace active, and (3) the primary-ASTE origin designates the ASTE for the base space of the dispatchable unit, then bits 0-55 and 58-63 of the PASCE in control register 1 are replaced by the same bits of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. This replacement occurs before a replacement of the SASCE in control register 7 by the PASCE. Further details are in “Subspace-Replacement Operations” on page 5-65.

PROGRAM TRANSFER (WITH INSTANCE) Serialization

For any of the PT-cp, PTI-cp, PT-ss, and PTI-ss operations, a serialization and checkpoint-synchroniza-

tion function is performed before the operation begins and again after the operation is completed. However, it is unpredictable whether or not a store into a trace-table entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store.

Special Conditions

The instruction can be executed only when the CPU is in the primary-space mode and the subsystem-linkage control, bit 0 of the linkage-table designation, or linkage-first-table designation in the current primary ASN-second-table entry, is one. If the CPU is in the real mode, secondary-space mode, access-register mode, or home-space mode, or if the subsystem-linkage control is zero, a special-operation exception is recognized.

In PT-ss when ASN-and-LX reuse is enabled, the reusable-ASN bit in the ASN-second-table entry (ASTE) must be zero; otherwise, a special-operation exception is recognized. In PTI-ss, regardless of the ASN-and-LX-reuse control, a special-operation exception is recognized if the controlled-ASN bit in the ASTE is one and the CPU is in the problem state at the beginning of the operation, and an ASTE-instance exception is recognized if the ASTEIN in general register R_1 is not equal to the ASTEIN in the ASTE.

Bit 63 of general register R_2 is placed in the problem-state bit position, PSW bit position 15, unless the operation would cause PSW bit 15 to change from one to zero (problem state to supervisor state). If such a change would occur, a privileged-operation exception is recognized.

In the 24-bit or 31-bit addressing mode, the instruction is completed only if bits 32-39 of general register R_2 specify a valid combination of PSW bits 32 and 97-103. If bit 32 of general register R_2 is zero and bits 33-39 are not all zeros, a specification exception is recognized.

In addition to the above requirements, when a PT-ss or PTI-ss instruction is specified, the ASN-translation control, bit 44 of control register 14, must be one; otherwise, a special-operation exception is recognized.

When, for PT-ss or PTI-ss, the primary space-switch-event-control bit, bit 57 of register 1, is one either before or after the execution of the instruction, a

space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PT-ss or PTI-ss operation if a PER event is reported.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-61 on page 10-102.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (linkage-table or linkage-first-table designation in primary ASN-second-table entry; authority-table entry, PT-ss and PTI-ss only)
- ASN translation (PT-ss and PTI-ss only)
- ASTE instance (PTI-ss only)
- Operation (if ASN-and-LX-reuse facility is not installed, PTI only)
- Primary authority (PT-ss and PTI-ss only)
- Privileged operation (attempt to set the supervisor state when in the problem state)
- Space-switch event (PT-ss and PTI-ss only)
- Special operation
- Specification
- Subspace replacement (PT-ss and PTI-ss only)
- Trace

Programming Notes:

1. The operation of PROGRAM TRANSFER (PT) is such that it may be used to restore the CPU to the state saved by a previous basic PROGRAM CALL operation. This restoration is accomplished by issuing PT 3,14. Though general registers 3 and 14 are not restored to their original values, the PASN, PSW-key mask, problem-state bit, and instruction address are restored, and the

authorization index, PASCE, and primary-ASN-second-table-entry origin are made consistent with the restored PASN. In the 24-bit or 31-bit addressing mode, the basic-addressing-mode bit also is restored. If ASN-and-LX reuse is enabled, the PASTEIN also is made consistent with the restored PASN. Note that the SASN is not saved by PROGRAM CALL or restored by PROGRAM TRANSFER; PROGRAM TRANSFER sets the SASN equal to the restored PASN. PROGRAM TRANSFER WITH INSTANCE is the same as PROGRAM TRANSFER except that the PASTEIN is made consistent regardless of ASN-and-LX-reuse enablement.

2. With proper authority, and while being executed in a common area, PROGRAM TRANSFER (WITH INSTANCE) may be used to change the primary address space to any desired space. The secondary address space is also changed to be the same as the new primary address space.
3. Unlike the RR-format branch instructions, a value of zero in the R_2 field for PROGRAM TRANSFER (WITH INSTANCE) designates general register 0, and branching occurs.
4. A program given control by a basic PROGRAM CALL operation can use EXTRACT SECONDARY ASN AND INSTANCE to obtain the ASTEIN to be used by PROGRAM TRANSFER WITH INSTANCE to return to the calling program or by SET SECONDARY ASN WITH INSTANCE to restore its secondary address space after a change of that space. This EXTRACT SECONDARY ASN AND INSTANCE instruction should be executed while the original secondary space remains continuously the secondary space; otherwise, depending on actions by the control program, EXTRACT SECONDARY ASN AND INSTANCE may return an ASTEIN that allows return to or use of a conceptually incorrect secondary space for which the ASTEIN has been changed.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B.1	Operation exception (PTI only, if the ASN-and-LX-reuse facility is not installed).
7.B.2	Special-operation exception due to DAT being off or the CPU being in secondary-space mode, access-register mode, or home-space mode.
8.A	Trace exceptions.
8.B.1	Addressing exception for access to linkage-table designation or linkage-first-table designation in primary ASN-second-table entry.
8.B.2	Special-operation exception due to subsystem-linkage control in linkage-table designation or linkage-first-table designation being zero.
8.B.3	Privileged-operation exception due to attempt to set the supervisor state when in the problem state.
8.B.4	Specification exception due to invalid combination (bit 32 is zero and bits 33-39 not zeros) in general register R_2 in 24-bit or 31-bit addressing mode.
8.B.5	Special-operation exception due to the ASN-translation control, bit 44 of control register 14, being zero (PT-ss or PTI-ss only).
8.B.6	ASN-translation exceptions (PT-ss and PTI-ss only).
8.B.7	Special-operation exception due to ASN-and-LX reuse enabled and reusable-ASN bit in ASN-second-table entry being one (PT-ss only).
8.B.8	Special-operation exception due to controlled-ASN bit in ASN-second-table entry being one and CPU being in problem state at the beginning of the operation (PTI-ss only).
8.B.9	ASTE-instance exception due to ASTEIN in general register R_1 not being equal to ASTEIN in ASN-second-table entry (PTI-ss only).
	Note: Subspace-replacement exceptions, which are not shown in detail in this figure, can occur with any priority after 8.B.9 and before 9.
8.B.10	Primary-authority exception due to authority-table entry being outside table (PT-ss and PTI-ss only).
8.B.11	Addressing exception for access to authority-table entry (PT-ss and PTI-ss only).
8.B.12	Primary-authority exception due to P bit in authority-table entry being zero (PT-ss and PTI-ss only).
9.	Space-switch event (PT-ss and PTI-ss only).

Figure 10-61. Priority of Execution: PROGRAM TRANSFER (WITH INSTANCE)

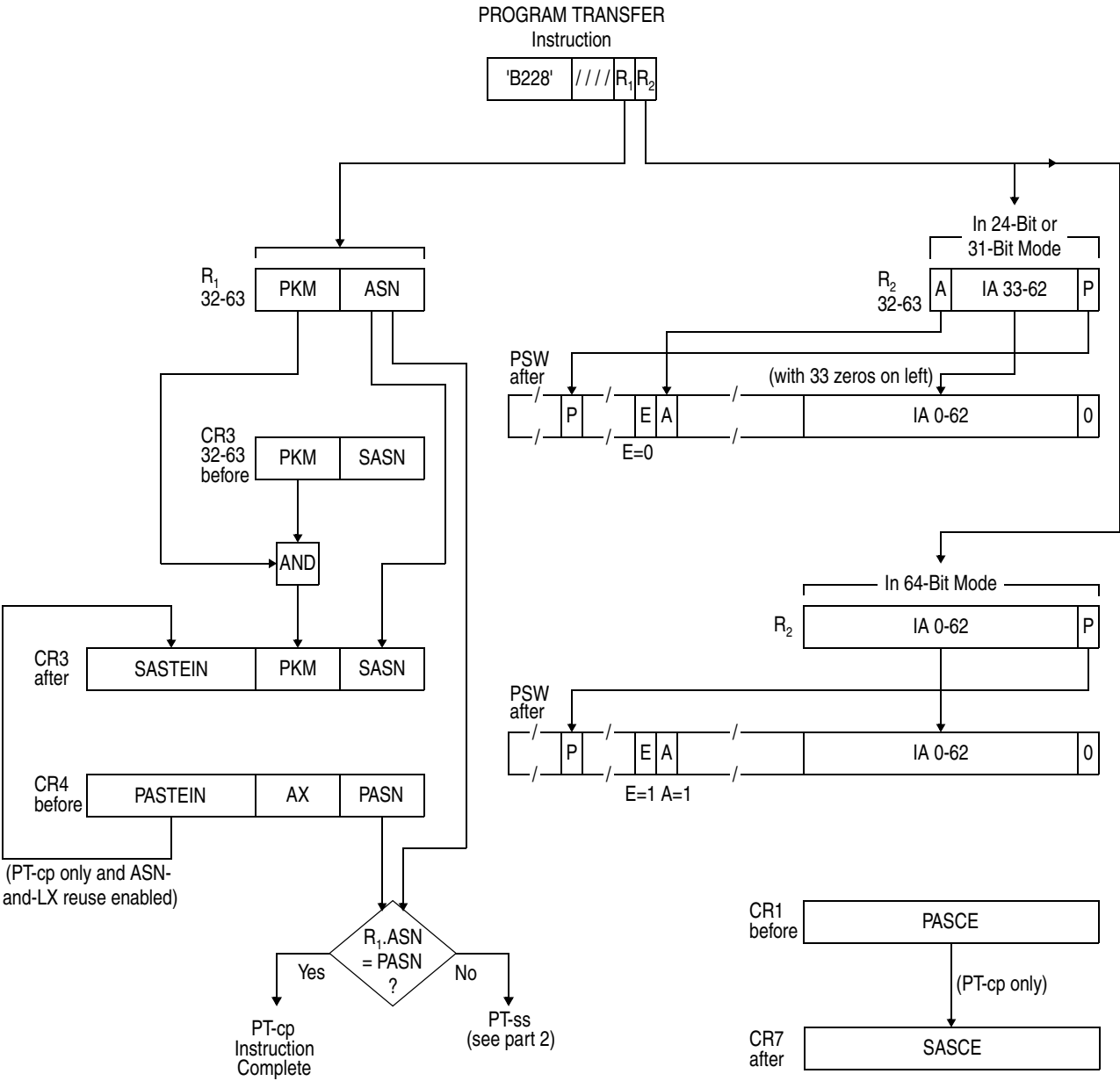
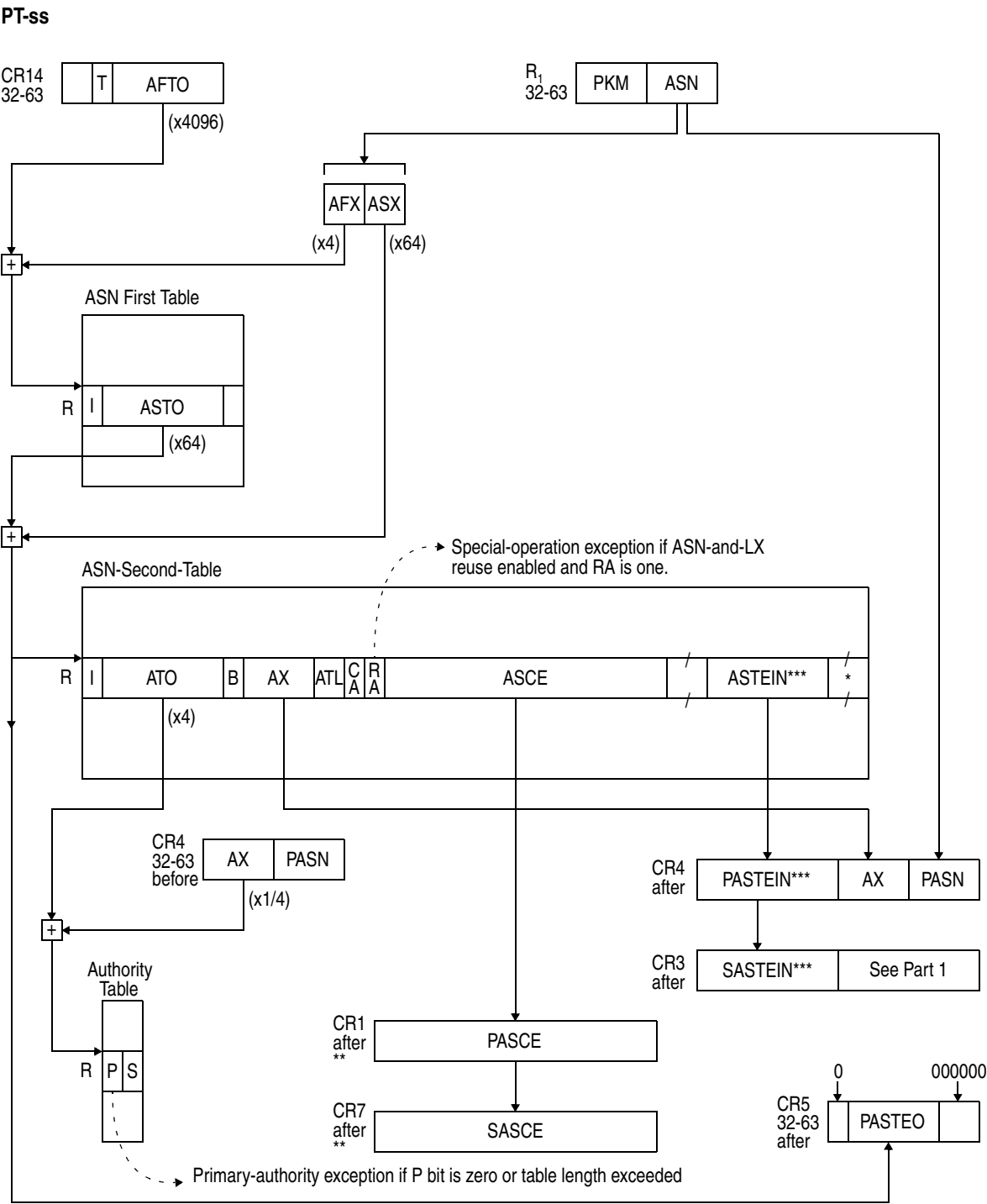


Figure 10-62. Execution of PROGRAM TRANSFER (Part 1 of 2).



R: Address is real
*: ASTE is 64 bytes; selected fields and the last 16 bytes are not shown.
**: Bits 0-55 and 58-63 of PASCE and SASCE may be replaced from a subspace ASCE
***: Operations on the ASTE instance number performed if ASN-and-LX-reuse enabled

Figure 10-62. Execution of PROGRAM TRANSFER (Part 2 of 2).

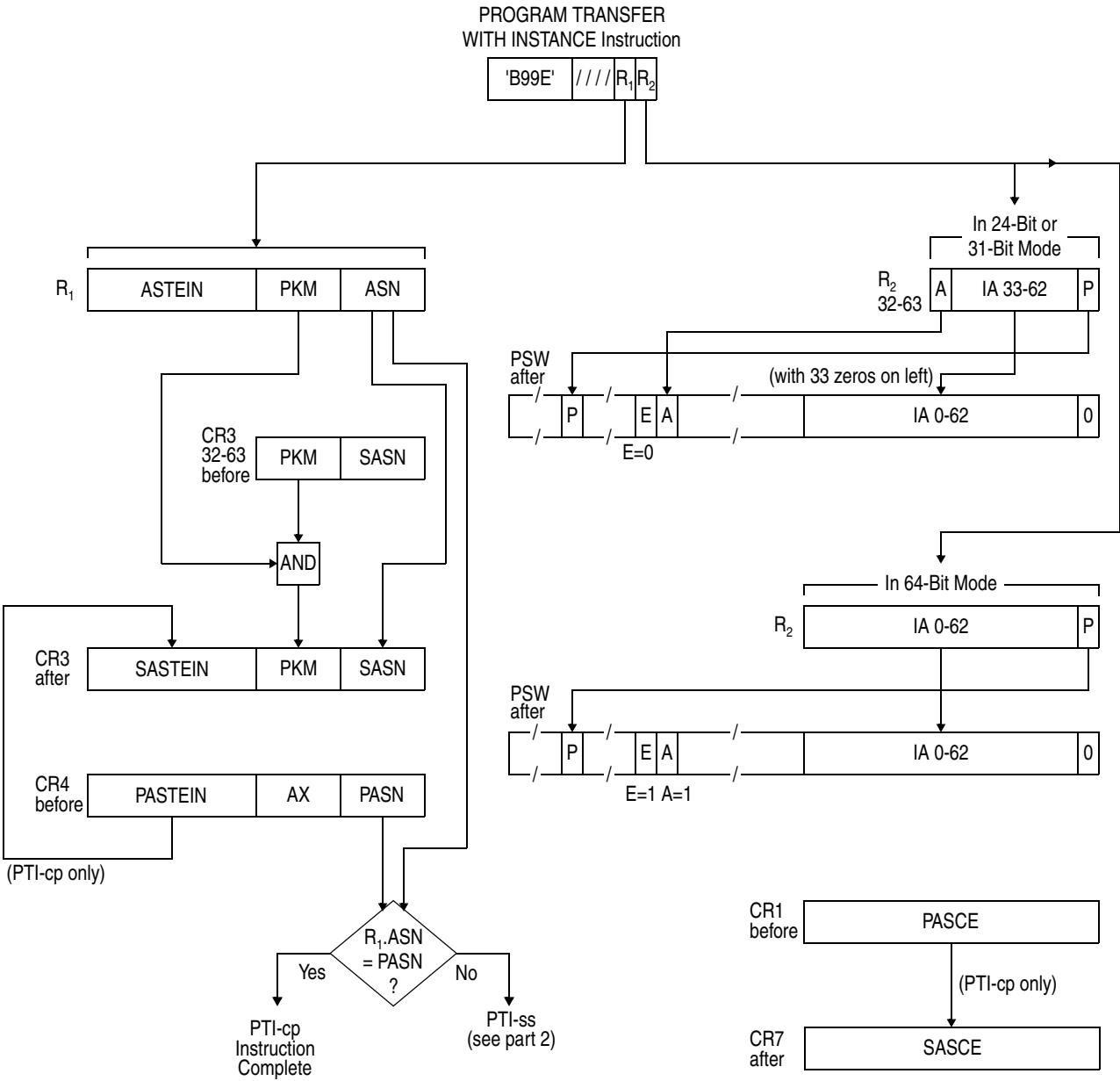
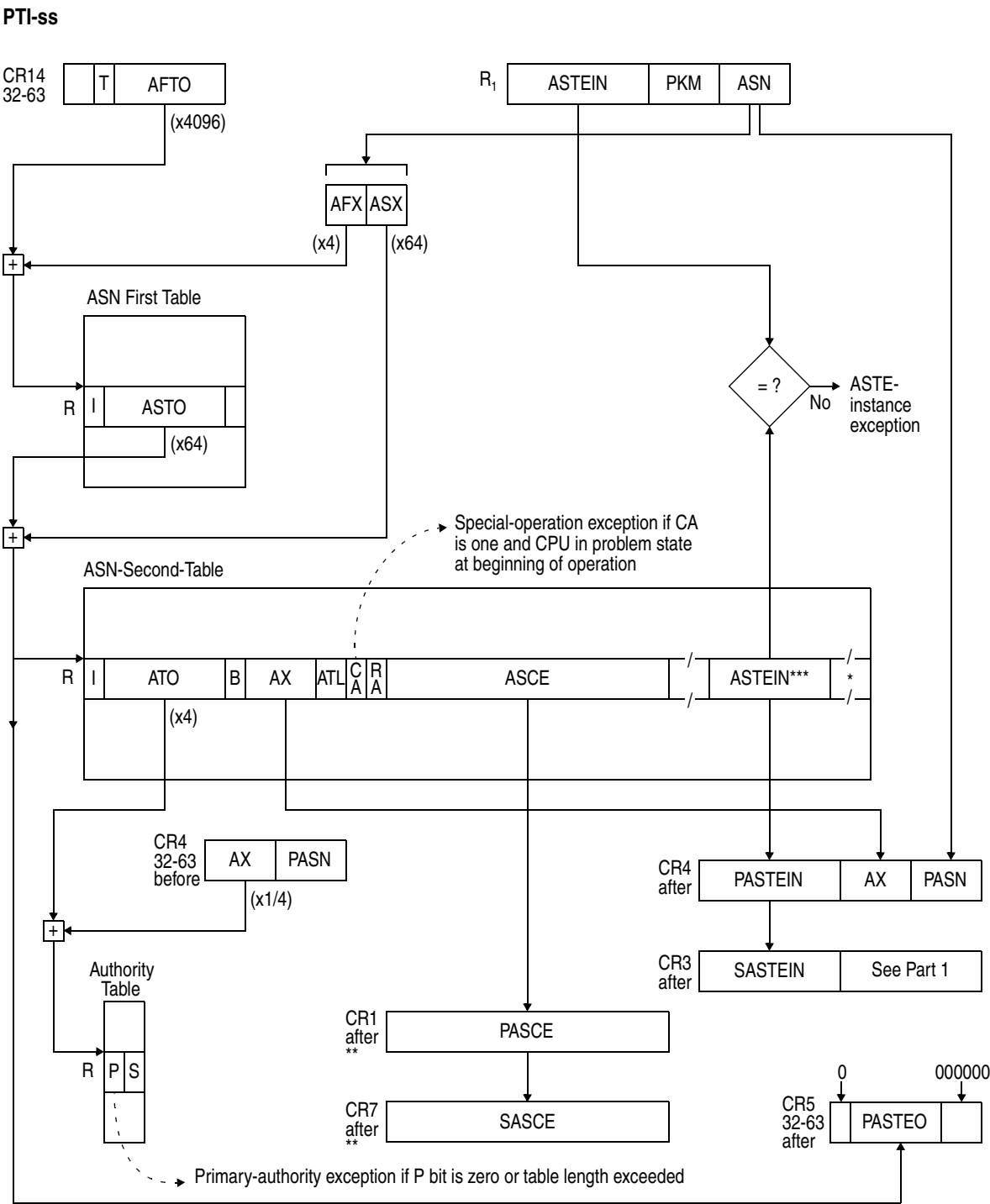


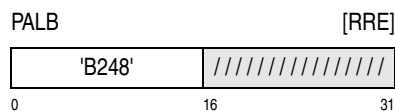
Figure 10-63. Execution of PROGRAM TRANSFER WITH INSTANCE (Part 1 of 2).



R: Address is real
*: ASTE is 64 bytes; selected fields and the last 16 bytes are not shown.
**: Bits 0-55 and 58-63 of PASCE and SASCE may be replaced from a subspace ASCE

Figure 10-63. Execution of PROGRAM TRANSFER WITH INSTANCE (Part 2 of 2).

PURGE ALB



The ART-lookaside buffer (ALB) of this CPU is cleared of entries. No change is made to the contents of addressable storage or registers.

The ALB appears cleared of its original contents beginning with the execution of the next sequential instruction. The operation is not signaled to any other CPU.

A serialization function is performed.

Condition Code: The code remains unchanged.

Program Exceptions:

- Privileged operation

PURGE TLB



The translation-lookaside buffer (TLB) of this CPU is cleared of entries. No change is made to the contents of addressable storage or registers.

The TLB appears cleared of its original contents beginning with the fetching of the next sequential instruction. The operation is not signaled to any other CPU.

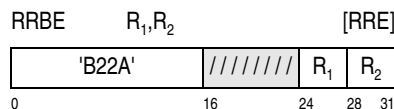
A serialization function is performed.

Condition Code: The code remains unchanged.

Program Exceptions:

- Privileged operation

RESET REFERENCE BIT EXTENDED



The reference bit in the storage key for the 4K-byte block that is addressed by the contents of general register R_2 is set to zero. The contents of general register R_1 are ignored.

In the 24-bit addressing mode, bits 40-51 of general register R_2 designate a 4K-byte block in real storage, and bits 0-39 and 52-63 of the register are ignored. In the 31-bit addressing mode, bits 33-51 of general register R_2 designate a 4K-byte block in real storage, and bits 0-32 and 52-63 of the register are ignored. In the 64-bit addressing mode, bits 0-51 of general register R_2 designate a 4K-byte block in real storage, and bits 52-63 of the register are ignored.

Because it is a real address, the address designating the storage block is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The remaining bits of the storage key, including the change bit, are not affected.

The condition code is set to reflect the state of the reference and change bits before the reference bit is set to zero.

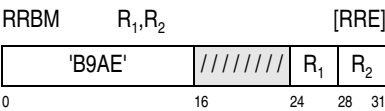
Resulting Condition Code:

- 0 Reference bit zero; change bit zero
- 1 Reference bit zero; change bit one
- 2 Reference bit one; change bit zero
- 3 Reference bit one; change bit one

Program Exceptions:

- Addressing (address specified by general register R_2)
- Privileged operation

RESET REFERENCE BITS MULTIPLE



Beginning with the block designated by the address in general register R₂, the reference bits in the storage keys of the 64 consecutive 4 K-byte blocks are inspected and reset. For each of the 64 blocks, the reference bit is placed in an ascending bit position of general register R₁, beginning with bit position 0 of the register. Subsequent to the inspection of each reference bit, the reference bit in the storage key is reset to zero.

General register R₂ designates the first of 64 blocks in absolute storage on a 64-block (256 K-byte) boundary. In the 24-bit addressing mode, bits 40-45 of the register, with six binary zeros appended on the right, designate the first block, and bits 0-39 and 46-63 of the register are ignored. In the 31-bit addressing mode, bits 33-45 of the register, with six binary zeros appended on the right, designate the first block, and bits 0-32 and 46-63 of the register are ignored. In the 64-bit addressing mode, bits 0-45 of the register, with six binary zeros appended on the right, designate the first block, and bits 46-63 of the register are ignored.

Because it is an absolute address, the address designating the first storage block is not subject to dynamic address translation. The references to the storage keys are not subject to protection exceptions.

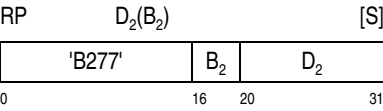
The remaining bits of the storage keys are not affected.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (address specified by general register R₂)
- Operation (reset-reference-bits-multiple facility not installed)
- Privileged operation

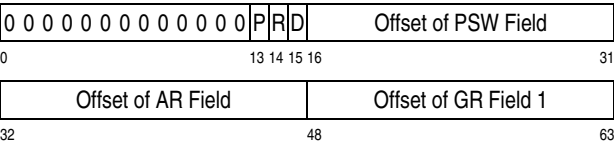
RESUME PROGRAM



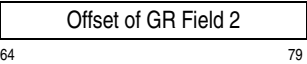
Certain contents of the current PSW and of access register and general register B₂ are replaced from three or four corresponding fields in the second operand. The size of the PSW field in the second operand, the size or number of general-register fields in the second operand, and the offsets of the fields in the second operand are specified in a parameter list that immediately follows the instruction in the instruction address space.

The instruction address space is the address space from which instructions are fetched. It is composed of real addresses if DAT is off.

The first 64 bits of the parameter list have the following format:



When bits 14 (R) and 15 (D) of the parameter list are both one, the list is an additional 16 bits in length, as follows:



Bit 13 of the parameter list (P) specifies the size of the PSW field in the second operand. The field is eight bytes if bit 13 is zero or 16 bytes if bit 13 is one.

Bits 14 and 15 of the parameter list (R and D) provide specifications about one or two general-register fields in the second operand, as follows:

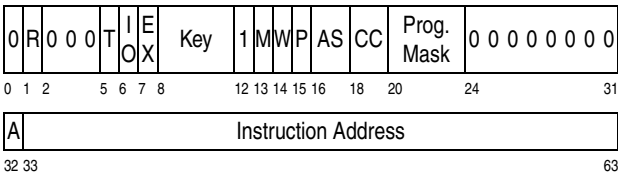
- When bit 14 is zero, then bit 15 is ignored, the general-register field 1 in the second operand is four bytes, from which bits 32-63 of general register B₂ will be replaced, there is not a general-register field 2 in the second operand, and bits 0-31 of general register B₂ will remain unchanged.

- When bit 14 is one and bit 15 is zero, then the general-register field 1 is eight bytes, from which bits 0-63 of general register B₂ will be replaced, and there is not a general-register field 2.
- When bits 14 and 15 are both one, then the general-register fields 1 and 2 are both four bytes, bits 32-63 of general register B₂ will be replaced from field 1, and bits 0-31 of the register will be replaced from field 2. (The letter “D” stands for disjoint.)

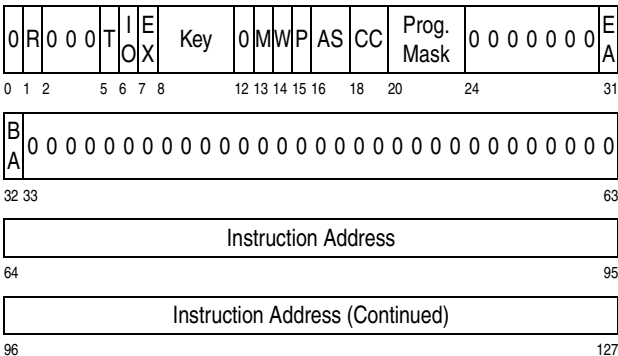
Bits 16-31 of the parameter list are an unsigned binary integer that is the offset in bytes from the beginning of the second operand to a field that has the format of an eight-byte or 16-byte PSW, depending on bit 13, and from which fields in the current PSW will be replaced. Bits 32-47 similarly are an offset to a four-byte field from which the contents of access register B₂ will be replaced. Bits 48-63 similarly are an offset to a four-byte or eight-byte field, depending on bits 14 and 15, from which bits 32-63 or 0-63, respectively, of general register B₂ will be replaced. If bits 64-79 of the parameter list exist, they similarly are an offset to a four-byte field from which bits 0-31 of general register B₂ will be replaced.

Bits 0-12 of the parameter list must be zeros; otherwise, a specification exception is recognized.

An eight-byte second-operand PSW field has the ESA/390 PSW format, as follows:



A 16-byte second-operand PSW field has the z/Architecture PSW format, as follows:



Fields in the current PSW are replaced from the corresponding fields in the PSW field in the second operand. The PSW fields that are replaced are as follows:

PSW Bits	Field Name
16 and 17	Address-space control (AS)
18 and 19	Condition code (CC)
20-23	Program mask
31	Extended addressing mode (EA)
32	Basic addressing mode (BA)
64-127	Instruction address

The remaining fields in the PSW field in the second operand are ignored. Specifically, there is no test for whether bit 12 is one in an eight-byte PSW or zero in a 16-byte PSW. There is also no test for whether bit 31 is zero in an eight-byte PSW.

Unassigned fields in the PSW may be assigned in the future and may then be among those restored by RESUME PROGRAM. Therefore, these fields in the PSW field in the second operand should contain zeros; otherwise, the program may not operate compatibly in the future.

When PSW bits 64-127 are replaced from an eight-byte PSW field in the second operand, they are replaced with bits 33-63 of the field, with 33 zeros appended on the left.

The fields in the second operand are fetched before the contents of access register B₂ and general register B₂ are changed.

When RESUME PROGRAM is the target of an execute-type instruction, the parameter list immediately follows the RESUME PROGRAM instruction, not the execute-type instruction.

The references to the parameter list are storage-operand fetches, not instruction fetches.

Special Conditions

The instruction is completed only if the bits 31, 32, and 64-127 that are to be placed in the current PSW are valid for placement in the PSW. If bits 31 and 32 are both zero and bits 64-103 are not all zeros, if bits 31 and 32 are zero and one, respectively, and bits 64-96 are not all zeros, if bits 31 and 32 are one and zero, respectively, or if bit 127 is one, a specification exception is recognized.

The CPU must be in the supervisor state when the operation is to set the home-space mode; otherwise, a privileged-operation exception is recognized. When DAT is off, the values of bits 16 and 17 of the PSW field in the second operand are not tested.

When the CPU is in the home-space mode either before or after the operation, but not both before and after the operation, a space-switch-event program interruption occurs after the operation is completed if any of the following is true: (1) the primary space-switch-event control, bit 57 of the primary address-

space-control element (ASCE) in control register 1, is one; (2) the home space-switch-event control, bit 57 of the home ASCE in control register 13, is one; or (3) a PER event is to be indicated.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-64 on page 10-110.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.	Access exceptions for second instruction halfword.
8.A	Trace exceptions.
8.B.1	Access exceptions for bits 0-63 of parameter list.
8.B.2	Specification exception due to bits 0-12 of parameter list not being all zeros.
8.B.3	Access exceptions for bits 64-79 of parameter list, if these bits exist.
8.B.4	Access exceptions for second operand.
8.B.5	Privileged-operation exception due to attempt to set the home-space mode when in the problem state.
8.B.6	Specification exception due to invalid values in bit positions 31, 32, and 64-127 of PSW in second operand.
9.	Space-switch event.

Figure 10-64. Priority of Execution: RESUME PROGRAM

Resulting Condition Code: The code is set as specified by the new condition code loaded.

Program Exceptions:

- Access (fetch, parameter list and operand 2)
- Privileged operation (attempt to set the home-space mode when in the problem state)
- Space-switch event
- Specification
- Trace

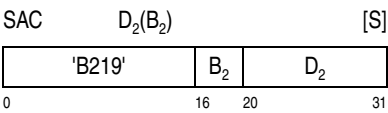
Programming Notes:

1. As described in “Instruction Fetching” on page 5-89, the bytes of an instruction may be fetched piecemeal, and the instruction may be fetched multiple times for a single execution. Therefore, the results are unpredictable when instructions are fetched for execution from storage that is being changed by another CPU or a channel program. This warning is particularly

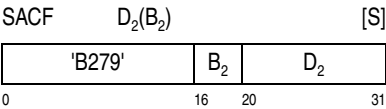
applicable when RESUME PROGRAM is the target of an execute-type instruction since the execute-type instruction may be refetched in order to generate, from its second operand, the address of the parameter list used by RESUME PROGRAM. If the execute-type instruction is refetched, there is not necessarily a test for whether storage still contains either the execute-type instruction or the RESUME PROGRAM instruction.

2. The storage-operand references for RESUME PROGRAM may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

SET ADDRESS SPACE CONTROL

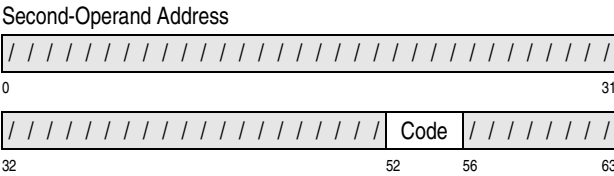


SET ADDRESS SPACE CONTROL FAST



Bits 52-55 of the second-operand address are used as a code to set the address-space-control bits in the PSW. The second-operand address is not used to address data; instead, bits 52-55 form the code. Bits 0-51 and 56-63 of the second-operand address are ignored. Bits 52 and 53 of the second-operand address must be zeros; otherwise, a specification exception is recognized.

The following figure summarizes the operation of SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST:



Code	Name of Mode	Result in PSW Bits 16 and 17
0000	Primary space	00
0001	Secondary space	10
0010	Access register	01
0011	Home space	11
All others	Invalid	

The CPU must be in the supervisor state when the operation is to set the home-space mode; otherwise, a privileged-operation exception is recognized.

For SET ADDRESS SPACE CONTROL, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed. This function is not performed for SET ADDRESS SPACE CONTROL FAST.

Special Conditions

For SET ADDRESS SPACE CONTROL, the operation is performed only when the secondary-space control, bit 37 of control register 0, is one and DAT is on. When either the secondary-space control is zero or DAT is off, a special-operation exception is recog-

nized. The same rules apply also to SET ADDRESS SPACE CONTROL FAST, except that whether the secondary-space control is tested is unpredictable.

When the CPU is in the home-space mode either before or after the operation, but not both before and after the operation, a space-switch-event program interruption occurs after the operation is completed if any of the following is true: (1) the primary space-switch-event control, bit 57 of the primary address-space-control element (ASCE) in control register 1, is one; (2) the home space-switch-event control, bit 57 of the home ASCE in control register 13, is one; or (3) a PER event is to be indicated.

The priority of recognition of program exceptions for the instructions is shown in Figure 10-65 on page 10-111.

Condition Code: The code remains unchanged.

Program Exceptions:

- Privileged operation (attempt to set the home-space mode in the problem state)
- Space-switch event
- Special operation
- Specification

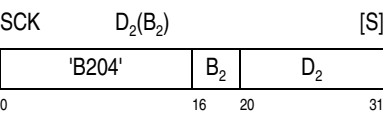
1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Special-operation exception due to DAT being off.
7.C	Special-operation exception due to the secondary-space control, bit 37 of control register 0, being zero. May be omitted for SET ADDRESS SPACE CONTROL FAST.
8.	Privileged-operation exception due to attempt to set home-space mode when in problem state.
9.	Specification exception due to non-zero value in bit positions 52 and 53 of second-operand address.
10.	Space-switch event.

Figure 10-65. Priority of Execution: SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST

Programming Notes:

1. SET ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL FAST are defined in such a way that the mode to be set can be placed directly in the displacement field of the instruction or can be specified from the same bit positions of a general register as those in which the mode is saved by INSERT ADDRESS SPACE CONTROL.
2. SET ADDRESS SPACE CONTROL FAST may provide better performance than SET ADDRESS SPACE CONTROL, depending on the model.
3. Because SET ADDRESS SPACE CONTROL FAST does not perform the serialization function, it does not cause copies of prefetched instructions to be discarded. To ensure predictable results after SET ADDRESS SPACE CONTROL FAST is used to switch to or from the home-space mode, the program must cause prefetched instructions to be discarded before an instruction is executed in a location that does not contain the same instruction in both the primary and home address spaces. The operations that cause prefetched instructions to be discarded are described in "Instruction Fetching" on page 5-89.
4. If a program stores into the instruction stream at a location following a subsequent SET ADDRESS SPACE CONTROL FAST instruction, and the SET ADDRESS SPACE CONTROL FAST instruction changes the translation mode either from or to either the access-register mode or the home-space mode, a copy of a prefetched instruction may be executed instead of the value that was stored. To avoid this situation, either SET ADDRESS SPACE CONTROL must be used instead of SET ADDRESS SPACE CONTROL FAST or some other means must be used to cause prefetched instructions to be discarded after the conceptual store occurs.

SET CLOCK



The current value of the TOD clock is replaced by the contents of the doubleword designated by the sec-

ond-operand address, and the clock enters the stopped state.

The doubleword operand replaces the contents of the clock, as determined by the resolution of the clock. Only those bits of the operand are set in the clock that correspond to the bit positions which are updated by the clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock. In some models, starting at or to the right of bit position 52, the rightmost bits of the second operand are ignored, and the corresponding positions of the clock which are implemented are set to zeros. Zeros are also placed in positions to the right of bit position 63 of the clock.

After the clock value is set, the clock enters the stopped state. The clock leaves the stopped state to enter the set state and resume incrementing under control of the TOD-clock-sync control, bit 34 of control register 0, of the CPU which most recently caused the clock to enter the stopped state. When the bit is zero, the clock enters the set state at the completion of the instruction. When the bit is one, the clock remains in the stopped state until the bit is set to zero or until another CPU executes a SET CLOCK instruction affecting the clock. If an external time reference (ETR) is installed, a signal from the ETR may be used to set the set state from the stopped state. When the system is not in the interpretive-execution mode, and an external time reference (ETR) is not attached to the configuration, the TOD-clock-sync control is treated as being zero, regardless of its actual value; in this case, the clock enters the set state and resumes incrementing upon completion of the instruction.

The value of the clock is changed and the clock is placed in the stopped state only if the manual TOD-clock control of any CPU in the configuration is set to the enable-set position or the TOD-clock-control-override control, bit 42 of control register 14, is one. If the TOD-clock control of all CPUs is set to the secure position and the TOD-clock-control-override control is zero, the value and state of the clock are not changed. Whether the clock is set or remains unchanged is distinguished by condition codes 0 and 1, respectively.

When the clock is not operational, the value and state of the clock are not changed, regardless of the settings of the TOD-clock control and the TOD-clock-control-override control, and condition code 3 is set.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Clock value set
- 1 Clock value secure
- 2 --
- 3 Clock in not-operational state

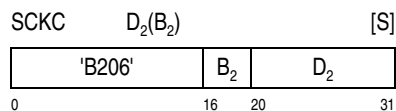
Program Exceptions:

- Access (fetch, operand 2)
- Privileged operation
- Specification

Programming Notes:

1. SET CLOCK should be issued only while all other activity on all CPUs in the configuration has been suspended. This activity should not be resumed until after the TOD clock has entered the set state.
2. To ensure predictable operation of the CPU timer after the TOD clock enters the set state, SET CPU TIMER should be issued on all CPUs in the configuration.
3. To ensure predictable operation of clock-comparator interruptions after the TOD clock enters the set state, one of the following instructions should be issued on each CPU in the configuration: SET CLOCK COMPARATOR (SCKC), STORE CLOCK (STCK), or STORE CLOCK EXTENDED (STCKE). (Execution of STCKF does not necessarily include the interlocks required for this special situation.)
4. SET CLOCK is provided for compatibility purposes. It is recommended that, where possible, the program use the PTFE control function PTFE-STO (Set TOD Offset), rather than SET CLOCK.

SET CLOCK COMPARATOR



The current value of the clock comparator is replaced by the contents of the doubleword designated by the second-operand address.

Only those bits of the operand are set in the clock comparator that correspond to the bit positions to be compared with the TOD clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock comparator.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

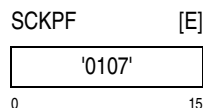
The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Privileged operation
- Specification

SET CLOCK PROGRAMMABLE FIELD



Bits 48-63 of general register 0 are placed in bit positions 16-31 of the TOD programmable register. Zeros are placed in bit positions 0-15 of the TOD programmable register.

Special Conditions

Bits 32-47 of general register 0 must be zeros; otherwise, a specification exception is recognized. Bits 0-31 of general register 0 are ignored.

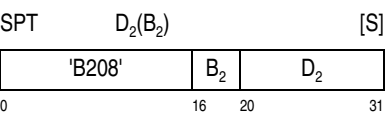
Condition Code: The code remains unchanged.

Program Exceptions:

- Privileged operation
- Specification

Programming Note: The values in the TOD programmable registers of a configuration should be the same and should be unique within a multiple-configuration system.

SET CPU TIMER



The current value of the CPU timer is replaced by the contents of the doubleword designated by the second-operand address.

Only those bits of the operand are set in the CPU timer that correspond to the bit positions to be updated; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the CPU timer.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

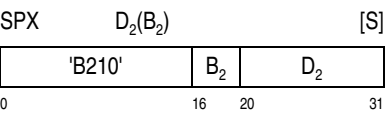
The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Privileged operation
- Specification

SET PREFIX



The contents of bit positions 33-50 of the prefix register are replaced by the contents of bit positions 1-18 of the word at the location designated by the second-operand address. The ART-lookaside buffer (ALB) and translation-lookaside buffer (TLB) of this CPU are cleared of entries.

After the second operand is fetched, the value is tested for validity before it is used to replace the contents of the prefix register. Bits 1-18 of the operand with 13 zeros appended on the right and 33 zeros appended on the left are used as an absolute address of the 8K-byte new prefix area in storage. This address is treated as a 64-bit address regardless of the addressing mode specified by the current PSW. The two 4K-byte blocks within the new prefix area are accessed; if either is not available in the configuration, an addressing exception is recognized, and the operation is suppressed. The accesses to the blocks are not subject to protection; however, the accesses may cause the reference bits for the blocks to be set to ones.

If the operation is completed, the new prefix is used for any interruptions following the execution of the instruction and for the execution of subsequent instructions. The contents of bit positions 0 and 19-31 of the second operand are ignored.

The ART-lookaside buffer (ALB) and translation-lookaside buffer (TLB) are cleared of entries. The ALB and TLB appear cleared of their original contents, beginning with the fetching of the next sequential instruction.

A serialization function is performed before or after the second operand is fetched and again after the operation is completed.

Special Conditions

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

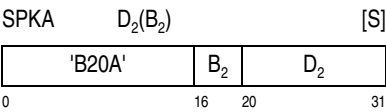
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Addressing (new prefix area)

- Privileged operation
- Specification

SET PSW KEY FROM ADDRESS



The four-bit PSW key, bits 8-11 of the current PSW, is replaced by bits 56-59 of the second-operand address.

The second-operand address is not used to address data; instead, bits 56-59 of the address form the new PSW key. Bits 0-55 and 60-63 of the second-operand address are ignored.

Special Conditions

In the problem state, the execution of the instruction is subject to control by the PSW-key mask in control register 3. When the bit in the PSW-key mask corresponding to the PSW-key value to be set is one, the instruction is executed successfully. When the selected bit in the PSW-key mask is zero, a privileged-operation exception is recognized. In the supervisor state, any value for the PSW key is valid.

Condition Code: The code remains unchanged.

Program Exceptions:

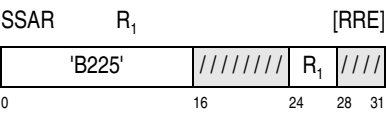
- Privileged operation (selected PSW-key-mask bit is zero in the problem state)

Programming Notes:

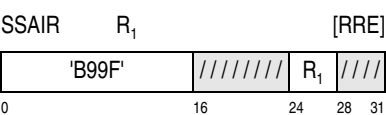
1. The format of SET PSW KEY FROM ADDRESS permits the program to set the PSW key either from the general register designated by the B₂ field or from the D₂ field in the instruction itself.
2. When one program requests another program to access a location designated by the requesting program, SET PSW KEY FROM ADDRESS can be used by the called program to verify that the requesting program is authorized to make this access, provided the storage location of the called program is not protected against fetching. The called program can perform the verification by replacing the PSW key with the requesting-

program PSW key before making the access and subsequently restoring the called-program PSW key to its original value. Caution must be exercised, however, in handling any resulting protection exceptions since such exceptions may cause the operation to be terminated. See TEST PROTECTION and the associated programming notes for an alternative approach to the testing of addresses passed by a calling program.

SET SECONDARY ASN



SET SECONDARY ASN WITH INSTANCE



Note: In this instruction definition, the name “SET SECONDARY ASN (WITH INSTANCE)” refers to the SET SECONDARY ASN instruction and the SET SECONDARY ASN WITH INSTANCE instruction.

The ASN specified in bit positions 48-63 of general register R₁ replaces the secondary ASN in control register 3, and the address-space-control element corresponding to that ASN replaces the SASCE in control register 7.

The contents of bit positions 48-63 of general register R₁ are called the new ASN. For SET SECONDARY ASN WITH INSTANCE, bits 0-31 of the register are an STEIN and are compared against the new SAS-TEIN if the SASTEIN is not set equal to the PASTEIN, and bits 32-47 of the register are ignored. For SET SECONDARY ASN, bits 0-47 of the register are ignored.

When the new ASN is equal to the current PASN, the operation is called SET SECONDARY ASN (WITH INSTANCE) to current primary (SSAR-cp or SSAIR-cp); when the fields are not equal, the operation is called SET SECONDARY ASN (WITH INSTANCE) with space switching (SSAR-ss or SSAIR-ss). The SSAR-cp and SSAR-ss operations are depicted in

Figure 10-67 on page 10-119. The SSAIR-cp and SSAIR-ss operations are depicted in Figure 10-68 on page 10-120.

SET SECONDARY ASN (WITH INSTANCE) to Current Primary (SSAR-cp or SSAIR-cp)

In the SSAR-cp or SSAIR-cp operation, the new ASN is equal to the PASN. The new ASN replaces the SASN, bits 48-63 of control register 3; the PASCE in control register 1 replaces the SASCE in control register 7.

In SSAR-cp, if the ASN-and-LX-reuse facility is installed and is enabled by a one value of the ASN-and-LX-reuse control, bit 44 of control register 0, the SASTEIN in bit positions 0-31 of control register 3 is replaced by the PASTEIN in bit positions 0-31 of control register 4. This replacement occurs in SSAIR-cp regardless of the value of the ASN-and-LX-reuse control. In SSAIR-cp, there is not a test of whether the current PASTEIN equals the ASTEIN in bit positions 0-31 of general register R₁; the ASTEIN is ignored. The operation is completed.

SET SECONDARY ASN (WITH INSTANCE) with Space Switching (SSAR-ss or SSAIR-ss)

In the SSAR-ss or SSAIR-ss operation, the new ASN is not equal to the PASN, and the new ASN is translated by means of a two-level table lookup. Bits 0-9 of the new ASN (bits 48-57 of the register) are a 10-bit AFX which is used to select an entry from the ASN first table. Bits 10-15 of the new ASN (bits 58-63 of the register) are a six-bit ASX which is used to select an entry from the ASN second table. The two-level lookup is described in “ASN Translation” on page 3-26. The exceptions associated with ASN translation are collectively called “ASN-translation exceptions.” These exceptions and their priority are described in Chapter 6, “Interruptions.”

In SSAR-ss, if the ASN-and-LX-reuse facility is installed and is enabled by the ASN-and-LX-reuse control in control register 0, the reusable-ASN bit, bit 63, in the located ASN-second-table entry (ASTE) must be zero; otherwise, a special-operation exception is recognized. In SSAIR-ss, regardless of the ASN-and-LX-reuse control, the controlled-ASN bit, bit 62, in the ASTE must be zero in the problem state; otherwise, a special-operation exception is recognized. Also in SSAIR-ss, and regardless of the ASN-

and-LX-reuse control and the reusable-ASN bit, the ASTEIN in bit positions 0-31 of general register R₁ must equal the ASTEIN in bit positions 352-383 of the ASTE; otherwise, an ASTE-instance exception is recognized.

The ASN-second-table entry (ASTE) obtained as a result of the second lookup contains the address-space-control element and the authority-table origin and length associated with the ASN.

The authority-table origin from the ASTE is used as a base for a third table lookup. The current authorization index, bits 32-47 of control register 4, is used, after it has been checked against the authority-table length, as the index to locate the entry in the authority table. The authority-table lookup is described in “ASN Authorization” on page 3-31.

The new ASN, bits 48-63 of general register R₁, replaces the SASN, bits 48-63 of control register 3. The address-space-control element in the ASTE replaces the SASCE in control register 7. In SSAR-ss if ASN-and-LX reuse is enabled, and in SSAIR-ss regardless of that enablement, the ASTEIN in the ASTE replaces the SASTEIN in bit positions 0-31 of control register 3.

The description in this paragraph applies to use of the subspace-group facility. After the new SASCE has been placed in control register 7, if (1) the subspace-group-control bit, bit 54, in the SASCE is one, (2) the dispatchable unit is subspace active, and (3) the ASTE obtained by ASN translation is the ASTE for the base space of the dispatchable unit, then bits 0-55 and 58-63 of the SASCE are replaced by the same bits of the ASCE in the ASTE for the subspace in which the dispatchable unit last had control. Further details are in “Subspace-Replacement Operations” on page 5-65.

SET SECONDARY ASN (WITH INSTANCE) Serialization

For any of the SSAR-cp, SSAIR-cp, SSAR-ss, and SSAIR-ss operations, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed. However, it is unpredictable whether or not a store into a trace-table entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store.

Special Conditions

The operation is performed only when the ASN-translation control, bit 44 of control register 14, is one and DAT is on. When either the ASN-translation-control bit is zero or DAT is off, a special-operation exception is recognized.

In SSAR-ss when ASN-and-LX reuse is enabled, the reusable-ASN bit in the ASN-second-table entry (ASTE) must be zero; otherwise, a special-operation exception is recognized. In SSAIR-ss, regardless of the ASN-and-LX-reuse control, a special-operation exception is recognized if the controlled-ASN bit in the ASTE is one and the CPU is in the problem state, and an ASTE-instance exception is recognized if the ASTEIN in general register R_1 is not equal to the ASTEIN in the ASTE.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-66 on page 10-118.

Condition Code: The code remains unchanged.

Program Exceptions:

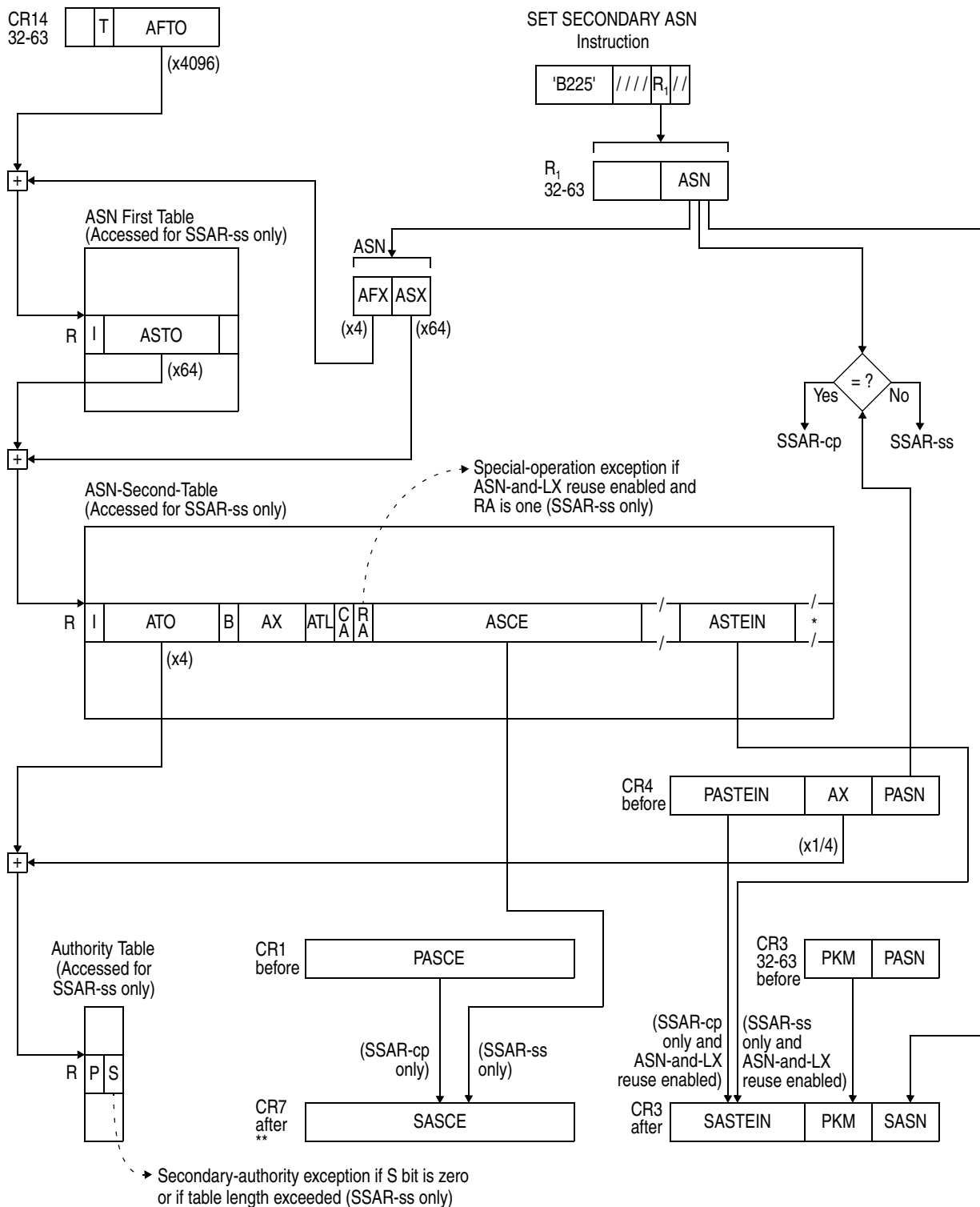
- Addressing (authority-table entry, SSAR-ss and SSAIR-ss only)

- ASN translation (SSAR-ss and SSAIR-ss only)
- ASTE instance (SSAIR-ss only)
- Operation (if ASN-and-LX-reuse facility is not installed, SSAIR only)
- Secondary authority (SSAR-ss and SSAIR-ss only)
- Special operation
- Subspace replacement (SSAR-ss and SSAIR-ss only)
- Trace

Programming Note: A program given control by a basic PROGRAM CALL operation can use EXTRACT SECONDARY ASN AND INSTANCE to obtain the ASTEIN to be used by PROGRAM TRANSFER WITH INSTANCE to return to the calling program or by SET SECONDARY ASN WITH INSTANCE to restore its secondary address space after a change of that space. This EXTRACT SECONDARY ASN AND INSTANCE instruction should be executed while the original secondary space remains continuously the secondary space; otherwise, depending on actions by the control program, EXTRACT SECONDARY ASN AND INSTANCE may return an ASTEIN that allows return to or use of a conceptually incorrect secondary space for which the ASTEIN has been changed.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B.1	Operation exception (SSAIR only, if the ASN-and-LX-reuse facility is not installed).
7.B.2	Special-operation exception due to DAT being off, or the ASN-translation control, bit 44 of control register 14, being zero.
8.A	Trace exceptions.
8.B.1	ASN-translation exceptions (SSAR-ss and SSAIR-ss only).
8.B.2	Special-operation exception due to ASN-and-LX reuse enabled and reusable-ASN bit in ASN-second-table entry being one (SSAR-ss only).
8.B.3	Special-operation exception due to controlled-ASN bit in ASN-second-table entry being one in the problem state (SSAIR-ss only).
8.B.4	ASTE-instance exception due to ASTEIN in general register R_1 not being equal to ASTEIN in ASN-second-table entry (SSAIR-ss only). Note: Subspace-replacement exceptions, which are not shown in detail in this figure, can occur with any priority after 8.B.4.
8.B.5	Secondary-authority exception due to authority-table entry being outside table (SSAR-ss and SSAIR-ss only).
8.B.6	Addressing exception for access to authority-table entry (SSAR-ss and SSAIR-ss only).
8.B.7	Secondary-authority exception due to S bit in authority-table entry being zero (SSAR-ss and SSAIR-ss only).

Figure 10-66. Priority of Execution: SET SECONDARY ASN (WITH INSTANCE)

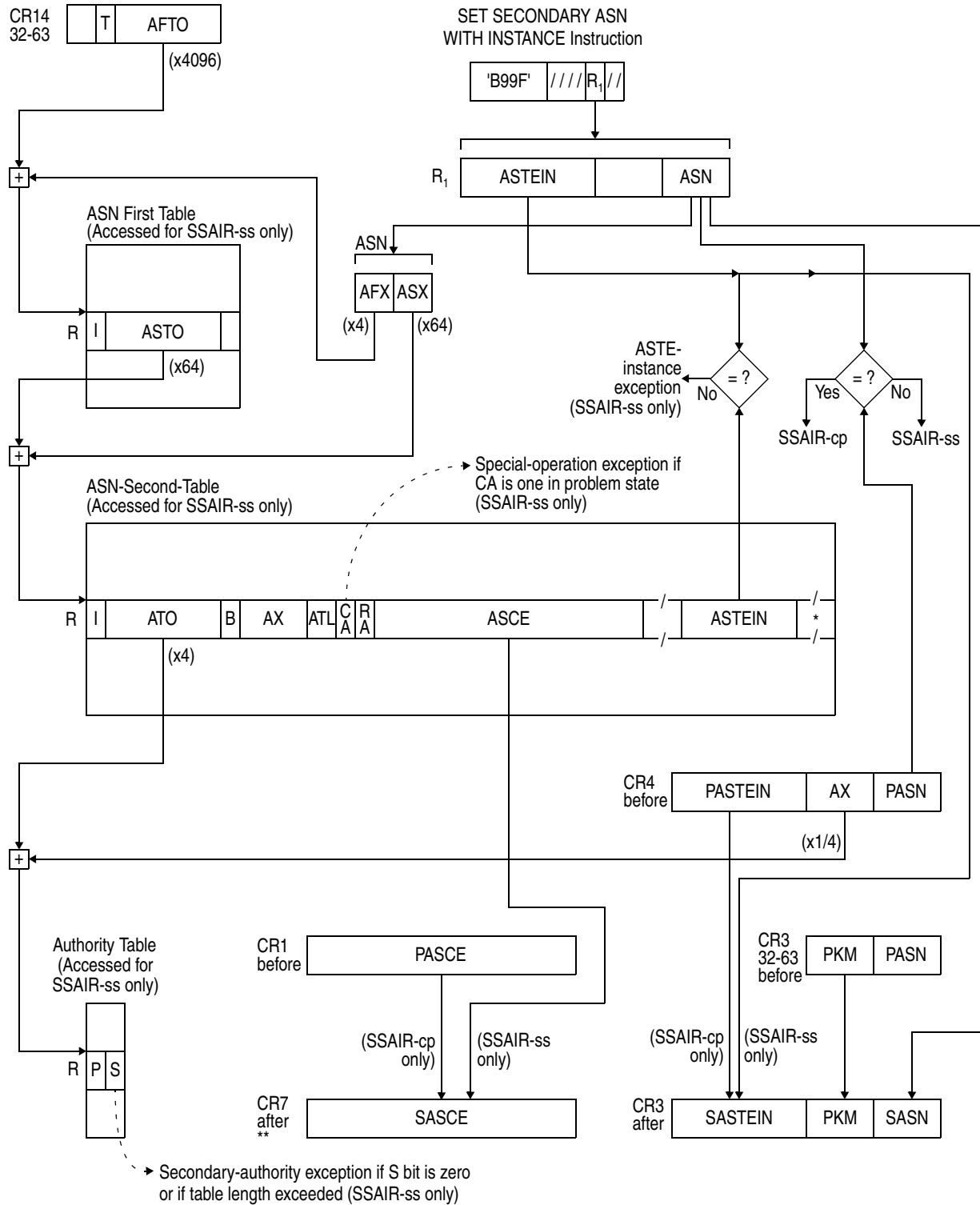


R: Address is real

*: ASTE is 64 bytes; selected fields and the last 16 bytes are not shown.

** For SSAR-ss only, bits 0-55 and 58-63 of SASCE may be replaced from a subspace ASCE

Figure 10-67. Execution of SET SECONDARY ASN



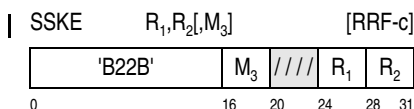
R: Address is real

*: ASTE is 64 bytes; selected fields and the last 16 bytes are not shown.

** For SSAIR-ss only, bits 0-55 and 58-63 of SASCE may be replaced from a subspace ASCE

Figure 10-68. Execution of SET SECONDARY ASN WITH INSTANCE

SET STORAGE KEY EXTENDED



The storage key for one or more 4K-byte blocks is replaced by the value in the first-operand register. When the conditional-SSKE facility is installed, certain functions of the key-setting operation may be bypassed.

When the conditional-SSKE facility is not installed, or when the conditional-SSKE facility is installed and both the MR and MC bits of the M_3 field are zero, the storage key for the 4K-byte block that is addressed by the contents of general register R_2 is replaced by bits from general register R_1 . The instruction completes without changing the condition code.

When the conditional-SSKE facility is installed and either or both of the MR and MC bits are one, the access-control bits, fetch-protection bit, and, optionally, the reference bit and change bit of the storage key that is addressed by the contents of general register R_2 are compared with corresponding bits in general register R_1 . If the compared bits are equal, then no change is made to the key; otherwise, selected bits of the key are replaced by the corresponding bits in general register R_1 . The storage key prior to any modification is inserted in general register R_1 , and the result is indicated by the condition code.

When the enhanced-DAT facility is installed, the above operations may be repeated for the storage keys of multiple 4K-byte blocks within the same 1MB block, subject to the control of the multiple-block control, described below.

The M_3 field has the following format:



The bits of the M_3 field are defined as follows:

- **Nonquiescing Control (NQ):** The NQ bit, bit 0 of the M_3 field, controls whether a quiescing operation is performed, as described below.

- **Reference-Bit-Update Mask (MR):** The MR bit, bit 1 of the M_3 field, controls whether updates to the reference bit in the storage key may be bypassed, as described below.
- **Change-Bit-Update Mask (MC):** The MC bit, bit 2 of the M_3 field, controls whether updates to the change bit in the storage key may be bypassed, as described below.
- **Multiple-Block Control (MB):** The MB bit, bit 3 of the M_3 field, controls whether the storage keys for multiple 4K-byte blocks of storage may be set, as described in "Setting Storage Keys in Multiple 4K-Byte Blocks" on page 10-123.

When the nonquiescing key-setting facility is not installed, bit 0 of the M_3 field is ignored. When the conditional-SSKE facility is not installed, bit positions 1 and 2 of the M_3 field are ignored. When the enhanced-DAT facility is not installed, bit position 3 of the M_3 field is ignored.

When the conditional-SSKE facility is installed, processing is as follows:

1. When both the MR and MC bits, bits 1 and 2 of the M_3 field, are zero, the instruction completes as though the conditional-SSKE facility was not installed. The storage key for the 4K-byte block that is addressed by the contents of general register R_2 is replaced by bits from general register R_1 , and the instruction completes without changing the condition code.
2. When either or both the MR and MC bits are one, processing is as follows:
 - a. Prior to any modification, the contents of the storage key for the 4K-byte block that is addressed by general register R_2 are placed in bit positions 48-54 of general register R_1 , and bit 55 of general register R_1 is set to zero. Bits 0-47 and 56-63 of the register remain unchanged.

If an invalid checking-block code (CBC) is detected when fetching the storage key, then (a) the entire storage key for the 4K-byte block is replaced by bits 56-62 of general register R_1 , (b) the contents of bit positions 48-55 of general register R_1 are unpredictable, and (c) the instruction completes by setting condition code 3.

- b. The access-control bits and fetch-protection bit of the storage key for the designated 4K-byte block are compared with the corresponding fields in bits 56-60 of general register R_1 . If the respective fields are not equal, the entire storage key for the 4K-byte block is replaced by bits from general register R_1 , and the instruction completes by setting condition code 1.

When the access-control and fetch-protection bits in the storage key are equal to the respective bits in general register R_1 , processing continues as described below.

- c. When both the MR and MC bits are one, the instruction completes by setting condition code 0. The storage key remains unchanged in this case.
- d. When the MR bit is zero and the MC bit is one, then the reference bit of the storage key for the designated 4K-byte block is compared with bit 61 of general register R_1 . If the bits are equal, the instruction completes by setting condition code 0. The storage key remains unchanged in this case.

If the bits are not equal, then either (a) the entire storage key for the designated 4K-byte block is replaced by the bits in general register R_1 , and the instruction completes by setting condition code 1; or (b) the reference bit for the storage key is replaced by bit 61 of general register R_1 , the change bit for the key is unpredictable, and the instruction completes by setting condition code 2. It is unpredictable whether condition code 1 or 2 is set.

- e. When the MC bit is zero and the MR bit is one, then the change bit of the storage key for the designated 4K-byte block is compared with bit 62 of general register R_1 . If the bits are equal, the instruction completes by setting condition code 0. The storage key remains unchanged in this case, except that the reference bit is unpredictable.

If the bits are not equal, then either (a) the entire storage key for the designated 4K-byte block is replaced by the bits in general register R_1 , and the instruction completes by setting condition code 1; or (b) the change bit

for the storage key is replaced by bit 62 of general register R_1 , the reference bit for the key is unpredictable, and the instruction completes by setting condition code 2. It is unpredictable whether condition code 1 or 2 is set.

When the enhanced-DAT facility is not installed, or when the facility is installed but the multiple-block control is zero, general register R_2 contains a real address. When the enhanced-DAT facility is installed and the multiple-block control is one, general register R_2 contains an absolute address.

In the 24-bit addressing mode, bits 40-51 of general register R_2 designate a 4K-byte block in real or absolute storage, and bits 0-39 and 52-63 of the register are ignored. In the 31-bit addressing mode, bits 33-51 of general register R_2 designate a 4K-byte block in real or absolute storage, and bits 0-32 and 52-63 of the register are ignored. In the 64-bit addressing mode, bits 0-51 of general register R_2 designate a 4K-byte block in real or absolute storage, and bits 52-63 of the register are ignored.

Because it is a real or absolute address, the address designating the storage block is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The new seven-bit storage-key value, or selected bits thereof, is obtained from bit positions 56-62 of general register R_1 . The contents of bit positions 0-55 and 63 of the register are ignored. When the conditional-SSKE facility is installed, and either or both the MR and MC bits are one, bit position 63 should contain a zero; otherwise, the program may not operate compatibly in the future.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed, except that when the conditional-SSKE facility is installed and the resulting condition code is 0, it is unpredictable whether a serialization and checkpoint-synchronization function is performed after the operation completes.

When the nonquiescing key-setting facility is not installed, or when the facility is installed and the non-quiescing control (NQ) is zero, the following applies:

- A quiescing operation is performed.

- For any store access, by any CPU or channel program, completed to the designated 4K-byte block either before or after the setting of the key by this instruction, the associated setting of the reference and change bits to one in the storage key for the block also is completed before or after, respectively, the execution of this instruction.

When the nonquiescing key-setting facility is installed and the NQ control is one, a quiescing operation is not necessarily performed. See “Storage-Key Accesses” on page 5-91 for a discussion the effects of quiescing on key-setting instructions, and see “Quiescing” on page 5-102 for details on the quiescing operation.

Setting Storage Keys in Multiple 4K-Byte Blocks

When the enhanced-DAT facility is not installed, or when the facility is installed, but the multiple-block control is zero, the storage key for a single 4K-byte block is set, as described above.

When the enhanced-DAT facility is installed, and the multiple-block control is one, the storage keys for multiple 4K-byte blocks within a 1M-byte block may be set, beginning with the block specified by the second-operand address, and continuing to the right with each successive block up to the next 1M-byte boundary. In this case, SET STORAGE KEY EXTENDED is interruptible, and processing is as follows:

- When an interruption occurs (other than one that follows termination), the leftmost bits of general register R_2 comprising the 4K-byte block address have been updated so the instruction, when reexecuted, resumes at the point of interruption. If either or both the MR or MC bits are one, the condition code is unpredictable; otherwise, the condition code is unchanged.
- When the instruction completes without interruption, the leftmost bits of general register R_2 comprising the 4K-byte block address have been updated to the next 1M-byte boundary. If either or both the MR or MC bits are one, condition code 3 is set; otherwise, the condition code is unchanged.

In either of the above two cases, the following applies:

- Bits 52-63 of general register R_2 remain unchanged.
- When either or both the MR or MC bits are one, bits 48-55 of general register R_1 are unpredictable.

When multiple-block processing occurs and the R_1 and R_2 fields designate the same register, the second-operand address is placed in the register. When multiple-block processing occurs in the 24-bit or 31-bit addressing modes, the leftmost bits which are not part of the address in bit positions 32-63 of general register R_2 are set to zeros; bits 0-31 of the register are unchanged.

Resulting Condition Code:

When the conditional-SSKE facility is not installed, or when both the MR and MC bits of the M_3 field are zero, the condition code remains unchanged.

When the conditional-SSKE facility is installed, and either or both of the MR and MC bits are one, the condition code is set as follows:

- 0 Storage key not set
- 1 Entire storage key set
- 2 Partial storage key set
- 3 Entire storage key set; bits 48-55 of general register R_1 are unpredictable.

Program Exceptions:

- Addressing (address specified by general register R_2)
- Privileged operation

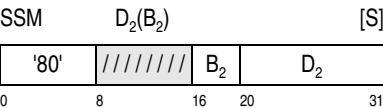
Programming Notes:

1. The M_3 field of the instruction is considered to be optional, as indicated by the field being contained within brackets [] in the assembler syntax. When the M_3 field is not specified, the assembler places zeros in that field of the instruction.
2. When setting multiple storage keys within the same 1M-byte block to the same value, use of the multiple-block control (MB, bit 3 of the M_3 field) may yield better performance than executing separate SSKE instructions for each 4K-byte block in the megabyte.
3. If the program does not rely on the setting of the reference bit, it may set the MR bit of the M_3 field

to one, regardless of whether or not the conditional-SSKE facility is installed. Similarly, if the program does not rely on the setting of the change bit, it may set the MC bit of the M_3 field to one, regardless of whether or not the conditional-SSKE facility is installed. In these cases, the program cannot rely on the condition code or the key value returned in general register R_1 . Conversely, if the program depends on accurate setting of the reference or change bit, then the MR and MC bits should be set to zero, such that reference and change bit recording are properly maintained when the conditional-SSKE facility is installed.

4. When SSKE is issued, the program must ensure that no channel subsystem is simultaneously altering the storage designated by general register R_2 . The program also must ensure that no other CPU or channel subsystem is accessing the storage designated by general register R_2 when the nonquiescing (NQ) control is one. Otherwise, unpredictable results may be observed by the other CPUs and channel subsystem, including the alteration of the block designated by general register R_2 .

SET SYSTEM MASK



Bits 0-7 of the current PSW are replaced by the byte at the location designated by the second-operand address.

Special Conditions

When the SSM-suppression-control bit, bit 33 of control register 0, is one and the CPU is in the supervisor state, a special-operation exception is recognized.

The value to be loaded into the PSW is not checked for validity before loading. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, if the contents of bit positions 0 and 2-4 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2. The specification exception, which is listed as a program exception for this instruction, is described in “Early Exception Rec-

ognition” on page 6-8. This exception may be considered as caused by execution of this instruction or as occurring early in the process of preparing to execute the subsequent instruction.

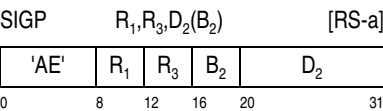
The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Privileged operation
- Special operation
- Specification

SIGNAL PROCESSOR



An eight-bit order code and, if called for, a 32-bit parameter are transmitted to the CPU designated by the CPU address contained in the third operand. The result is indicated by the condition code and may be detailed by status assembled in bit positions 32-63 of the first-operand location.

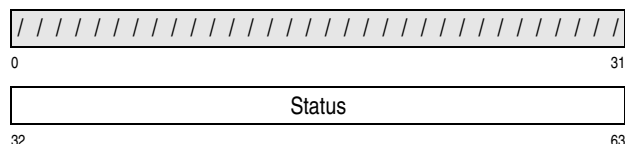
The second-operand address is not used to address data; instead, bits 56-63 of the address contain the eight-bit order code. Bits 0-55 of the second-operand address are ignored. The order code specifies the function to be performed by the addressed CPU. The assignment and definition of order codes appear in “CPU Signaling and Response” on page 4-58.

The 16-bit binary number contained in bit positions 48-63 of general register R_3 forms the CPU address. Bits 0-47 of the register are ignored. When the specified order is the set-architecture order, the CPU address is ignored; all other CPUs in the configuration are considered to be addressed.

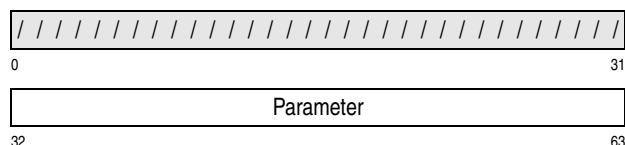
The general register containing the 32-bit parameter in bit positions 32-63 is R_1 or R_1+1 , whichever is the odd-numbered register. It depends on the order code whether a parameter is provided and for what purpose it is used.

The operands just described have the following formats:

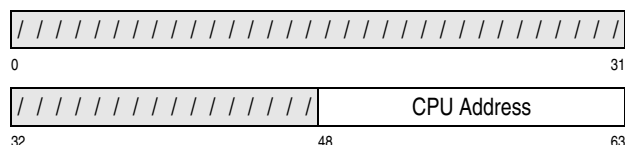
General register designated by R_1 :



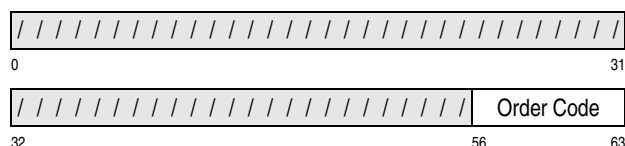
General register designated by R_1 or $R_1 + 1$, which-ever is the odd-numbered register:



General register designated by R_3 :



Second-operand address:



A serialization function is performed before the operation begins and again after the operation is completed.

When the order code is accepted and no nonzero status is returned, condition code 0 is set. When status information is generated by this CPU or returned by the addressed CPU, the status is placed in bit positions 32-63 of general register R_1 , bits 0-31 of the register remain unchanged, and condition code 1 is set.

When the access path to the addressed CPU is busy, or the addressed CPU is operational but in a state where it cannot respond to the order code, condition code 2 is set.

When the addressed CPU is not operational (that is, it is not provided in the installation, it is not in the configuration, it is in any of certain customer-engineer test modes, or its power is off), condition code 3 is set.

Resulting Condition Code:

- 0 Order code accepted
- 1 Status stored
- 2 Busy
- 3 Not operational

Program Exceptions:

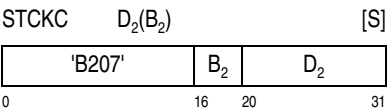
- Privileged operation

Programming Notes:

1. A more detailed discussion of the condition-code settings for SIGNAL PROCESSOR is contained in "CPU Signaling and Response" on page 4-58.
2. To ensure that presently written programs will be executed properly when new facilities using additional bits are installed, only zeros should appear in the unused bit positions of the second-operand address and in bit positions 32-47 of general register R_3 .
3. Certain SIGNAL PROCESSOR orders are provided with the expectation that they will be used primarily in special circumstances. Such orders may be implemented with the aid of an auxiliary maintenance or service processor, and, thus, the execution time may take several seconds. Unless all of the functions provided by the order are required, combinations of other orders, in conjunction with appropriate programming support, can be expected to provide a specific function more rapidly. The emergency-signal, external-call, and sense orders are the only orders which are intended for frequent use. The following orders are intended for infrequent use, and performance therefore may be much slower than for frequently used orders: restart, set prefix, store status at address, start, stop, stop and store status, set architecture, and all the reset orders. An alternative to the set-prefix order, for faster performance when the receiving CPU is not already stopped, is the use of the emergency-signal or external-call order, followed by the execution of a SET PREFIX instruction on the addressed CPU. Clearing the TLB of entries is ordinarily accom-

plished more rapidly through the use of the emergency-signal or external-call order, followed by execution of the PURGE TLB instruction on the addressed CPU, than by use of the set-prefix order.

STORE CLOCK COMPARATOR



The current value of the clock comparator is stored at the doubleword location designated by the second-operand address.

Zeros are provided for the rightmost bit positions of the clock comparator that are not compared with the TOD clock.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Privileged operation
- Specification

STORE CONTROL



Bits of the set of control registers starting with control register R₁ and ending with control register R₃ are stored at the locations designated by the second-operand address.

For STORE CONTROL (STCTL), bits 32-63 of the control registers are stored in successive words beginning at the second-operand address, and bits 0-31 of the registers are ignored. For STORE CONTROL (STCTG), bits 0-63 of the control registers are stored in successive doublewords beginning at the second-operand address.

The storage area where the contents of the control registers are placed starts at the location designated by the second-operand address and continues through as many storage words, for STCTL, or doublewords, for STCTG, as the number of control registers specified. The contents of the control registers are stored in ascending order of their register numbers, starting with control register R₁ and continuing up to and including control register R₃, with control register 0 following control register 15. The contents of the control registers remain unchanged.

The displacement for STCTL is treated as a 12-bit unsigned binary integer. The displacement for STCTG is treated as a 20-bit signed binary integer.

Special Conditions

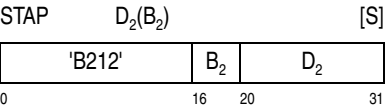
The second operand must be designated on a word boundary for STCTL or on a doubleword boundary for STCTG; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Privileged operation
- Specification

STORE CPU ADDRESS



The CPU address by which this CPU is identified in a multiprocessing configuration is stored at the half-word location designated by the second-operand address.

Special Conditions

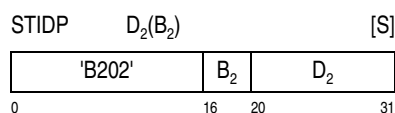
The operand must be designated on a halfword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

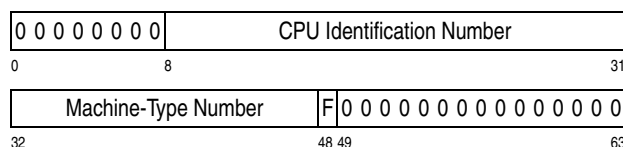
- Access (store, operand 2)
- Privileged operation
- Specification

STORE CPU ID



Information identifying either (1) the CPU or (2) the configuration and the logical partition in which the program is being executed is stored at the doubleword location designated by the second-operand address.

The information stored has the following format:



Bit positions 8-31 contain the CPU identification number, consisting of six four-bit digits. Some or all of these digits are selected from the physical serial number stamped on the CPU.

The format bit (F) in bit position 48 specifies the format of the first two digits of the CPU identification number.

When the format bit is zero, the contents of the CPU-identification-number field, in conjunction with the machine-type number, permit unique identification of the CPU. When the format bit is one, the CPU-identification number identifies the system configuration as opposed to an individual CPU within the configuration, and it identifies the logical partition in which the program is being executed.

Bit positions 32-47 contain the machine-type number of the CPU. Bit positions 0-7 and 49-63 contain zeros.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Privileged operation
- Specification

Programming Notes:

1. The program should allow for the possibility that the CPU identification number may contain the digits A-F as well as the digits 0-9.
2. When the format bit in bit position 48 of the second operand is zero, the CPU identification number, in conjunction with the machine-type number, provides a unique CPU identification that can be used in associating results with an individual machine.

When the format bit is one, the CPU-identification number identifies the system configuration as opposed to an individual CPU within the configuration, and it identifies the requesting logical partition. The CPU identity can be obtained using the STORE CPU ADDRESS or the STORE SYSTEM INFORMATION instruction.

3. In versions of *Enterprise Systems Architecture/390 Principles of Operation* prior to SA22-7201-03, the machine-type-number field was called the model-number field.
4. In ESA/390, bit positions 0-7 of the information stored contain a version code. When the version code is nonzero, it is usually indicative of the model number of the model and the number of CPUs contained in the model. In both ESA/390 and z/Architecture, the STORE SYSTEM INFORMATION instruction can be used to determine the model number and the number of CPUs in the model.

5. The CPU identification number has the hex format:
- “Annnnn” in the basic mode, or
 - LPnnnn” in the LPAR (logically-partitioned) mode, when the format bit is zero, or
 - PPnnnn” in the LPAR mode, when the format bit is one.

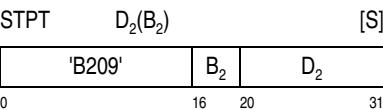
Where:

- A is the CPU address of the CPU.
- L is a logical CPU address.
- P is a logical-partition identifier.
- PP is the user partition identifier (UPID). The UPID is an eight-bit unsigned binary integer bound to a logical partition.
- n is a digit derived from the serial number of the CPU.

The terminology above that is not defined in this publication is defined in the appropriate System Library publications for the model.

6. The format bit is always stored as a zero in the basic mode.
7. Model z800 and z900 machines always store the format bit as a zero.
8. When STORE CPU ID is executed in a virtual machine, the VM operating system replaces bits 0-7 of the information stored with FF hex.

STORE CPU TIMER



The current value of the CPU timer is stored at the doubleword location designated by the second-operand address.

Zeros are provided for the rightmost bit positions that are not updated by the CPU timer.

Special Conditions

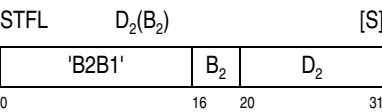
The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Privileged operation
- Specification

STORE FACILITY LIST



A list of bits providing information about facilities is stored in the word at real address 200. The meanings of the bits are identical to the first 32 bits stored by STORE FACILITY LIST EXTENDED. Figure 4-19, “Assigned Facility Bits” on page 4-67 shows the meanings of the assigned facility bits.

The second-operand address is ignored but should be zero to permit possible future extensions.

Key-controlled and low-address protection do not apply.

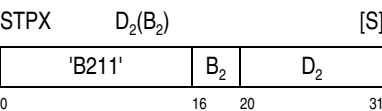
Condition Code: The code remains unchanged.

Program Exceptions:

- Privileged operation

Programming Note: STORE FACILITY LIST stores only a 32-bit indication of facilities. STORE FACILITY LIST EXTENDED may be used to store the entire list of installed facilities.

STORE PREFIX



The contents of bit positions 33-50 of the prefix register are stored in bit positions 1-18 of the word location designated by the second-operand address, and zeros are stored in bit positions 0 and 19-31 of the word.

Special Conditions

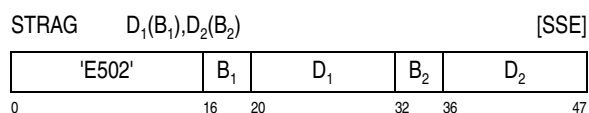
The operand must be designated on a word boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Privileged operation
- Specification

STORE REAL ADDRESS



The 64-bit real address corresponding to the second-operand virtual address is stored in the doubleword at the location designated by the first-operand address.

The virtual address specified by the B_2 and D_2 fields is translated by means of the dynamic-address-translation facility, regardless of whether DAT is on or off.

DAT is performed by using an address-space-control element that depends on the current value of the address-space-control bits, bits 16 and 17 of the PSW, as shown in the following table:

PSW Bits 16 and 17	Address-Space-Control Element Used by DAT
00	Contents of control register 1
10	Contents of control register 7
01	The address-space-control element obtained by applying the access-register-translation (ART) process to the access register designated by the B_2 field
11	Contents of control register 13

ART and DAT may be performed with the use of the ART-lookaside buffer (ALB) and translation-lookaside buffer (TLB), respectively.

The resultant 64-bit real address is stored at the first-operand location.

The translated address is not inspected for boundary alignment or for addressing or protection exceptions.

The address computations for the operands are performed according to the current addressing mode, specified by bits 31 and 32 of the current PSW.

The addresses of the region-table entry or entries, if used, and of the segment-table entry and page-table entry are treated as 64-bit addresses regardless of the current addressing mode. It is unpredictable whether the addresses of these entries are treated as real or absolute addresses.

Special Conditions

The first operand must be designated on a double-word boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing exceptions.

Condition Code: The code remains unchanged.

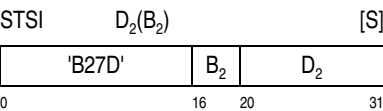
Program Exceptions:

- Access (fetch, operand 2, except for an addressing or protection exception for the designated location; store, operand 1)
- Privileged operation
- Specification

Programming Note: STORE REAL ADDRESS is contrasted to LOAD REAL ADDRESS as follows:

- In the 24-bit or 31-bit addressing mode, LOAD REAL ADDRESS (LRA) loads bits 33-63 of the real address if bits 0-32 of the address are all zeros or recognizes a special-operation exception if bits 0-32 are not all zeros. LRA in the 64-bit addressing mode, and LOAD REAL ADDRESS (LRAG) in any addressing mode, loads bits 0-63 of the real address. STORE REAL ADDRESS stores bits 0-63 of the real address in any addressing mode.
- LOAD REAL ADDRESS, for most access-exception conditions, does not recognize the conditions as exceptions but instead sets the condition code to indicate the occurrence of the conditions. STORE REAL ADDRESS recognizes all access-exception conditions as exceptions, resulting in a program interruption.

STORE SYSTEM INFORMATION



Depending on a function code in general register 0, either an identification of the level of the configuration executing the program is placed in general register 0 or information about a component or components of a configuration is stored in a system-information block (SYSIB). When information about a component or components is requested, the information is specified by further contents of general register 0 and by contents of general register 1. The SYSIB, if any, is designated by the second-operand address.

The machine is considered to provide one, two, or three levels of configuration. The levels are:

- 1. The basic machine, which is the machine as if it were operating in the basic mode.
- 2. A logical partition, which is provided if the machine is operating in the LPAR, or logically-partitioned, mode. A logical partition is provided by the LPAR hypervisor, which is a part of the machine. A basic machine exists even when the machine is operating in the LPAR mode.
- 3. A virtual machine, which is provided by a virtual-machine (VM) control program that is executed either by the basic machine or in a logical partition. A virtual machine may itself execute a VM control program that provides a higher-level (more removed from the basic machine) virtual machine, which also is considered a level-3 configuration.

The terms basic mode, LPAR mode, logical partition, hypervisor, and virtual machine, and any other terms related specifically to those terms, are not defined in this publication; they are defined in the machine manuals.

A program being executed by a level-1 configuration (the basic machine) can request information about that configuration. A program being executed by a level-2 configuration (in a logical partition) can request information about the logical partition and about the underlying basic machine. A program being executed by a level-3 configuration (a virtual

machine) can request information about the virtual machine and about the one or two underlying levels; a basic machine is always underlying, and a logical partition may or may not be between the basic machine and the virtual machine. When information about a virtual machine is requested, information is provided about the configuration executing the program and about any underlying level or levels of virtual machine. In any of these cases, information is provided about a level only if the level implements the instruction.

The function code determining the operation is an unsigned binary integer in bit positions 32-35 of general register 0 and is as follows:

Function Code	Information Requested
0	Current-configuration-level number
1	Information about level 1 (the basic machine)
2	Information about level 2 (a logical partition)
3	Information about level 3 (a virtual machine)
4-14	None; codes are reserved
15	Current-configuration-level information

Invalid Function Code

The level of the configuration executing the program is called the current level. The configuration level specified by a nonzero function code (other than 15) is called the specified level. When the specified level is numbered higher than the current level, then the function code is called invalid, the condition code is set to 3, and no other action (including checking) is performed. Function code 15 is invalid when the configuration-topology facility is not installed.

Valid Function Code

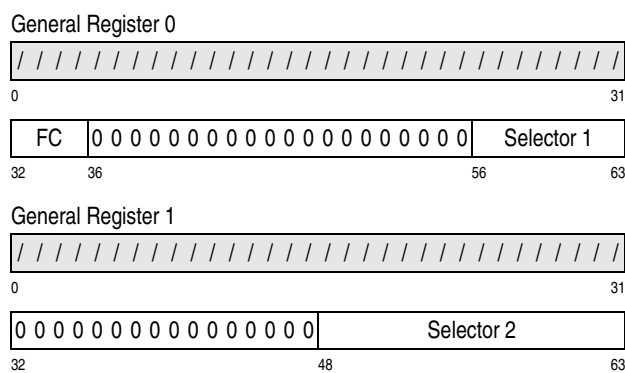
When the function code is equal to or less than the number of the current level, or 15, it is called valid. In this case, bits 36-55 of general register 0 and bits 32-47 of general register 1 must be zero; otherwise, a specification exception is recognized. Bits 0-31 of general registers 0 and 1 always are ignored.

When the function code is 0, an unsigned binary integer identifying the current configuration level (1 for basic machine, 2 for logical partition, or 3 for virtual machine) is placed in bit positions 32-35 of general register 0, the condition code is set to 0, and no further action is performed.

When the function code is valid and nonzero, general registers 0 and 1 contain additional specifications about the information requested, as follows:

- Bit positions 56-63 of general register 0 contain an unsigned binary integer, called *selector 1*, that specifies a component or components of the specified configuration.
- Bit positions 48-63 of general register 1 contain an unsigned binary integer, called *selector 2*, that specifies the type of information requested.

The contents of general registers 0 and 1 are as follows:



When the function code is valid and nonzero, information may be stored in a system-information block (SYSIB) beginning at the second-operand location. The SYSIB is 4K bytes and must begin at a 4K-byte boundary; otherwise, a specification exception may be recognized, depending on selector 1 and selector 2 and on whether access exceptions are recognized due to references to the SYSIB (see “Special Conditions”).

Selector 1 can have values as follows:

Selector 1	Information Requested
0	None; selector is reserved
1	Information about the configuration level specified by the function code.
2	Information about one or more CPUs in the specified configuration level
3-255	None; selectors are reserved

When selector 1 is 1, selector 2 can have values as follows:

Selector 2 when Selector 1 Is 1	Information Requested
0	None; selector is reserved
1	Information about the specified configuration level
2-6	Topology information about the specified configuration level
7-65,535	None; selectors are reserved

When selector 1 is 2, selector 2 can have values as follows:

Selector 2 when Selector 1 Is 2	Information Requested
0	None; selector is reserved
1	Information about the CPU executing the program in the specified configuration level
2	Information about all CPUs in the specified configuration level
3-65,535	None; selectors are reserved

Only certain combinations of the function code, selector 1, and selector 2 are valid, as shown in Figure 10-69.

Function Code	Selector 1	Selector 2	Information Requested about
0	-	-	Current-configuration-level number
1	1	1	Basic-machine configuration
1	2	1	Basic-machine CPU
1	2	2	Basic-machine CPUs
2	2	1	Logical-partition CPU
2	2	2	Logical-partition CPUs
3	2	2	Virtual-machine CPUs
15	1	2-6	Topology information of current configuration
Explanation:			
-	-	-	Ignored.

Figure 10-69. Valid Function-Code, Selector-1, and Selector-2 Combinations for STORE SYSTEM INFORMATION

When the specified function-code, selector-1, and selector-2 combination is invalid (is other than as shown in Figure 10-69), or if it is valid but the requested information is not available because the specified level does not implement or does not fully implement the instruction or because a necessary part of the level is uninstalled or not initialized, and provided that an exception is not recognized (see “Special Conditions”), the condition code is set to 3. When the function code is nonzero, the combination is valid, the requested information is available, and there is no exception, the requested information is stored in a system-information block (SYSIB) at the second-operand address.

Some or all of the SYSIB may be fetched before it is stored.

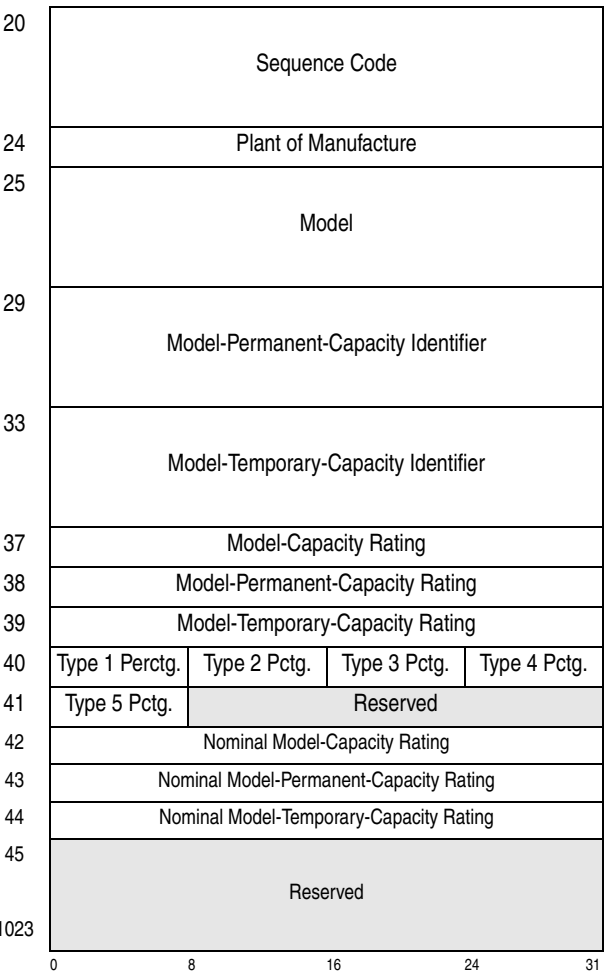
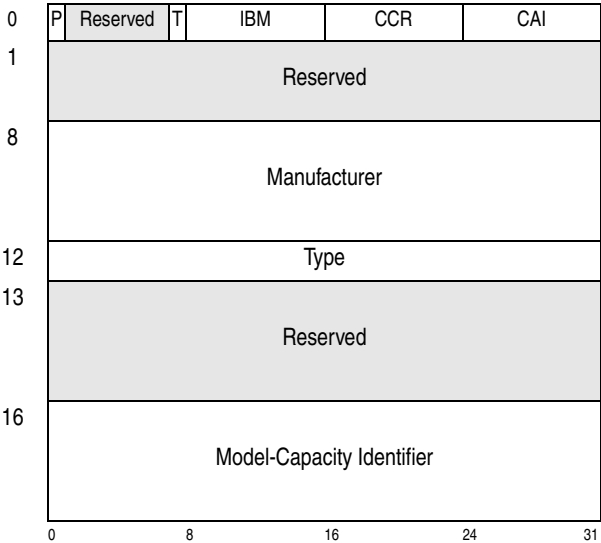
A SYSIB may be identified in references by means of “SYSIB fc.s1.s2,” where “fc,” “s1,” and “s2” are the values of a function code, selector 1, and selector 2, respectively.

Following sections describe the defined SYSIBs by means of figures and related text. In the figures, the offsets shown on the left are word values. “The configuration” refers to the configuration level specified by the function code (the configuration level about which information is requested).

SYSIB 1.1.1 (Basic-Machine Configuration)

SYSIB 1.1.1 has the following format:

SYSIB 1.1.1



Word 0, Byte 0: Byte 0 of word 0 contains the following bit definitions:

Bit Meaning

- 0 When bit 0 of byte 0 (P) is one, the type-percent-age bytes, located in words 40-41, are valid. When P is zero, the bytes are stored as zeros, but have no meaning.
- 1-6 Bits 1-6 of byte 0 are reserved and stored as zeros.
- 7 Bit 7 of byte 0 (T), when one, indicates that the condition represented by the CCR and CAI fields is relatively transient.

Reserved: The contents of bits 1-6 of word 0, words 1-7, 13-15, and 45-63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

IBM: Byte 1 of word 0 is assigned to IBM internal use.

Capacity-Change Reason (CCR): When the CAI byte is nonzero, the content of byte 2 of word 0 is an 8-bit unsigned integer whose value indicates one of the following reasons which is associated with the present values contained in SYSIB 1.1.1.

CCR Capacity-Change Reason

- | | |
|-------|---|
| 0 | Machine is running at nominal capacity. |
| 1 | The capacity change is due solely to the setting of a manual control, such as intentional power-save. |
| 2 | The capacity change is due to a machine-exception condition, such as detection of overheating. |
| 3 | The capacity change is due to a non-exception machine condition, such as concurrent service. |
| 4 | The capacity change is due to an exception condition external to the machine, such as detection of supplied power or cooling falling outside required tolerances. |
| 5-255 | Reserved. |

When the CAI byte is zero, the CCR field is undefined, and stored as zero.

When multiple capacity-change reasons exist, CCR is set according to the following priority:

- 1a. Machine-exception condition, CCR=2.
- 1b. External-exception condition, CCR=4.
2. Non-exception machine condition, CCR=3.
3. Manual control, CCR=1.

Capacity-Adjustment Indication (CAI): Byte 3 of word 0, when nonzero, is an 8-bit unsigned integer whose value is in the range 1-100 and represents the aggregate position of model-dependent controls. Temporary capacity changes that affect machine performance (for example, those due to Capacity Backup, CBU, On/Off Capacity on Demand, OOCOD, and Capacity Planned Event, CPE) are not included. When zero, the indication is not reported. When in the range 1-99, some amount of reduction is indi-

cated. When 100, the machine is operating at its normal capacity.

Primary CPUs and all secondary-type CPUs are similarly affected.

The model-capacity rating is not affected. A change in the CAI may also reflect a change in CPU-capability fields of SYSIB 1.2.2.

Manufacturer: Words 8-11 contain the 16-character (0-9 or uppercase A-Z) EBCDIC name of the manufacturer of the configuration. The name is left justified with trailing blanks if necessary.

Type: Word 12 contains the four-character (0-9) EBCDIC type number of the configuration. (This is called the machine-type number in the definition of STORE CPU ID.)

Model-Capacity Identifier (C): Words 16-19 contain the 16-character (0-9 or uppercase A-Z) EBCDIC model-capacity identifier of the configuration. The model-capacity identifier is left justified with trailing blanks if necessary.

Sequence Code: Words 20-23 contain the 16-character (0-9 or uppercase A-Z) EBCDIC sequence code of the configuration. The sequence code is right justified with leading EBCDIC zeros if necessary.

Plant of Manufacture: Word 24 contains the four-character (0-9 or uppercase A-Z) EBCDIC code that identifies the plant of manufacture for the configuration. The code is left justified with trailing blanks if necessary.

Model: When word 25 is not binary zeros, words 25-28 contain the 16-character (0-9 or uppercase A-Z) EBCDIC model identification of the configuration. The model identification is left justified with trailing blanks if necessary. (This is called the model number in programming note 4 on page 10-127 of STORE CPU ID.) When word 25 is binary zeros, the contents of words 16-19 represent both the model-capacity identifier and the model.

Model-Permanent-Capacity Identifier (P): When non-zero, words 29-32 contain the 16-character (0-9 or uppercase A-Z) EBCDIC model-permanent-capacity identifier of the configuration. The identifier is left justified with trailing blanks if necessary.

Model-Temporary-Capacity Identifier (T): When non-zero, words 33-36 contain the 16-character (0-9 or uppercase A-Z) EBCDIC model-temporary-capacity identifier of the configuration. The identifier is left justified with trailing blanks if necessary.

Model-Capacity Rating (CR): When non-zero, word 37 contains a 32-bit unsigned integer whose value is associated with the model capacity as identified by the model-capacity identifier. There is no formal description of the algorithm used to generate this integer.

Model-Permanent-Capacity Rating (PR): When non-zero, word 38 contains a 32-bit unsigned integer whose value is associated with the model-permanent capacity as identified by the model-permanent-capacity identifier. There is no formal description of the algorithm used to generate this integer.

Model-Temporary-Capacity Rating (TR): When non-zero, word 39 contains a 32-bit unsigned integer whose value is associated with the model-temporary capacity as identified by the model-temporary-capacity identifier. There is no formal description of the algorithm used to generate this integer.

Type N Pctg. Each of the byte fields in words 40-41 that is designated as a type-N percentage contains an 8-bit unsigned binary integer whose value is in the range 0-100 and represents a percentage. When non-zero, the percentage may be used to affect the use and allowed utilization of the secondary-CPU's whose CPU type corresponds to the particular byte. When a byte in this range contains a value of zero, use rules of the corresponding secondary-CPU type are not overridden. The reserved bytes in word 41 are reserved for the potential addition of new secondary CPU types.

Nominal Model-Capacity Rating (NCR): When non-zero, word 42 contains a 32-bit unsigned integer whose value is associated with the nominal model capacity as identified by the model-capacity identifier. There is no formal description of the algorithm used to generate this integer. The NCR value equals the CR value when the CAI byte contains a value of 100. When the CAI byte is less than 100, the CR value is less than the NCR value.

Nominal Model-Permanent-Capacity Rating (NPR): When non-zero, word 43 contains a 32-bit unsigned integer whose value is associated with the nominal model-permanent capacity as identified by

the model-permanent-capacity identifier. There is no formal description of the algorithm used to generate this integer. The NPR value equals the PR value when the CAI byte contains a value of 100. When the CAI byte is less than 100, the PR value is less than the NPR value.

Nominal Model-Temporary-Capacity Rating

(NTR): When non-zero, word 44 contains a 32-bit unsigned integer whose value is associated with the nominal model-temporary capacity as identified by the model-temporary-capacity identifier. There is no formal description of the algorithm used to generate this integer. The NTR value equals the TR value when the CAI byte contains a value of 100. When the CAI byte is less than 100, the TR value is less than the NTR value.

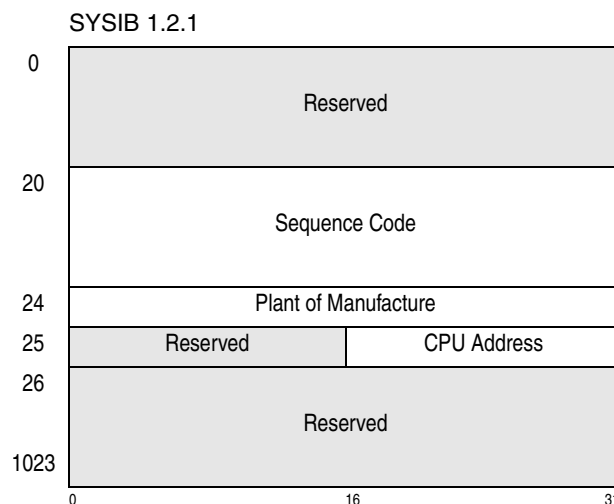
Programming Notes:

1. The fields of the SYSIB 1.1.1 are similar to those of the node descriptor described in the publication *Common I/O-Device Commands and Self Description*, SA22-7204. However, the contents of the SYSIB fields may not be identical to the contents of the corresponding node-descriptor fields because the SYSIB fields:
 - Allow more characters.
 - Are more flexible regarding the type of characters allowed.
 - Provide information that is justified differently within the field.
 - May not use the same method to determine the contents of fields such as the sequence-code field.
2. The model field in a node descriptor corresponds to the content of the STSI model field and not the STSI model-capacity-identifier field.
3. The model field specifies the model of the machine (i.e., the physical model); the model-capacity identifier field specifies a token that may be used to locate a statement of capacity or performance in the System Library publication for the model.
4. Each of the three model-capacity-identifier fields specifies a token that may be used to locate a statement of capacity or performance in the System Library publication for the model.

5. Each of the three model-capacity-rating fields, corresponding one-to-one with the three model-capacity-identifier fields, specifies a numeric value that may or may not represent the actual capacity of the machine
6. The term *capacity backup* (CBU) represents a methodology by which spare capacity in a CPC can be used to replace capacity from another CPC within an enterprise, normally for a relatively limited period of time. Typically CBU is used when another CPC of the enterprise has failed, and the CPC operating under CBU rules is making up for the missing CPC's capacity. However, CBU can also indicate a capacity that reflects a net reduction such that the current capacity would be less than the permanent capacity.
7. The term *On/Off Capacity on Demand* (On/Off CoD) represents a methodology by which spare capacity in a CPC can be made available to increase the total capacity within an enterprise, normally for a limited period of time. For example, On/Off CoD may be used to acquire additional capacity in order to handle a workload peak.
8. The content of the model-capacity identifier (C) field corresponds to the current capacity at which the central-processing complex (CPC) is operating.
9. The content of the model-permanent-capacity identifier (P) corresponds to the capacity of the CPC exclusive of temporarily-available increased capacity and temporarily-available replacement capacity.
10. The content of the model-temporary-capacity identifier (T) corresponds to the total of permanent capacity and temporarily-available increased capacity but exclusive of any temporarily-available replacement capacity.
11. When the T bit in byte 0 is one, the current capacity values and indicators are in a relatively-short period of transition and should be permitted to stabilize, that is, a result where T is zero, before any resultant action proceeds.

SYSIB 1.2.1 (Basic-Machine CPU)

SYSIB 1.2.1 has the following format:



Reserved: The contents of words 0-19, bytes 0 and 1 of word 25, and words 26-63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

Sequence Code: Words 20-23 contain the 16-character (0-9 or uppercase A-Z) EBCDIC sequence code of the configuration. The code is right justified with leading EBCDIC zeros if necessary.

Plant of Manufacture: Word 24 contains the four-character (0-9 or uppercase A-Z) EBCDIC code that identifies the plant of manufacture for the configuration. The code is left justified with trailing blanks if necessary.

CPU Address: Bytes 2 and 3 of word 25 contain the CPU address by which this CPU is identified in a multiprocessing configuration. The CPU address is a 16-bit unsigned binary integer.

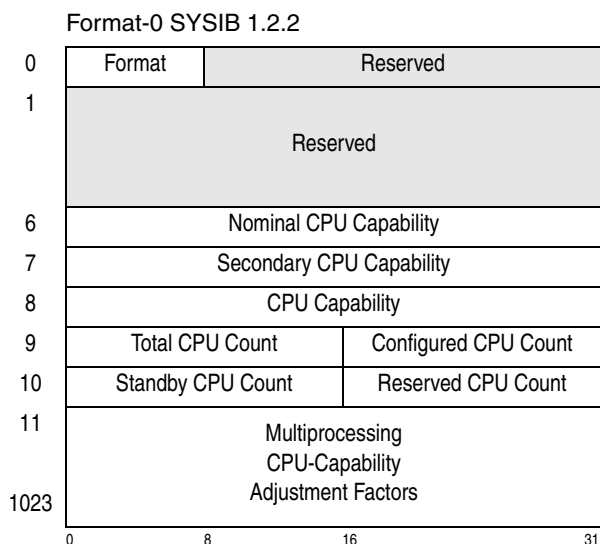
The CPU address is the same as is stored by STORE CPU ADDRESS when the program is executed by a machine operating in the basic mode.

Programming Note: Multiple CPUs in the same configuration have the same sequence code, and it is necessary to use other information, such as the CPU address, to establish a unique CPU identity. The sequence code returned for a basic-machine CPU and a logical-partition CPU are identical and have the same value as the sequence code returned for the basic-machine configuration.

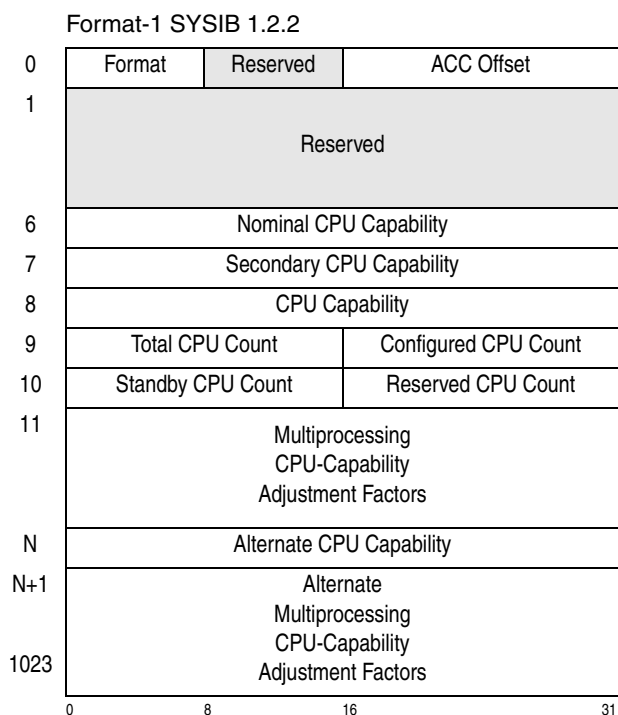
SYSIB 1.2.2 (Basic-Machine CPUs)

The format field in byte 0 of word 0 determines the format of the SYSIB.

When the format field has a value of zero, SYSIB 1.2.2 has a format-0 layout as follows:



When the format field has a value of one, SYSIB 1.2.2 has a format-1 layout as follows:



Reserved: When the format field contains a value of zero, the contents of bytes 1-3 of word 0 and words 1-5 are reserved and stored as zeros. When the format field contains a value of one, the contents of byte 1 of word 0 and words 1-5 are reserved and stored as zeros.

When fewer than 64 words are needed to contain the information for all the CPUs, the portion of the SYSIB following the adjustment-factor list in a format-0 SYSIB or the alternate-adjustment-factor list in a format-1 SYSIB, up to word 63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

When 64 or more words are needed to contain the information for all the CPUs, the portion of the SYSIB following the adjustment-factor list in a format-0 SYSIB or the alternate-adjustment-factor list in a format-1 SYSIB, up to word 1023 are reserved and may be stored as zeros or may remain unchanged.

Format: Byte 0 of word 0 contains an 8-bit unsigned binary integer that specifies the format of SYSIB 1.2.2.

Alternate-CPU-Capability Offset: When the format field has a value of one, bytes 2-3 of word 0 contain a 16-bit unsigned binary integer that specifies the offset in bytes of the alternate-CPU-capability field in the SYSIB.

Nominal CPU Capability: Word 6, when not zero, is formatted and encoded the same as word 8, CPU capability, including that a lower value indicates a proportionally higher CPU capability.

When the CAI byte of SYSIB 1.1.1 is zero, the nominal CPU capability (word 6) is stored as zero. The nonzero value equals the word-8 value when the CAI byte of SYSIB 1.1.1 contains a value of 100. When the CAI byte of SYSIB 1.1.1 is nonzero and less than 100, the word-6 value indicates a CPU speed greater than the CPU speed indicated by the word-8 value.

Secondary CPU Capability: Word 7, when not zero, is formatted and encoded the same as word 8, CPU capability, including that a lower value indicates a proportionally higher CPU capability, and specifies a secondary capability that may be applied to certain types of CPUs in the configuration.

There is no formal description of the algorithm used to generate this value, except that it is the same as the algorithm used to generate the CPU capability. The value is used as an indication of the capability of a CPU relative to the capability of other CPU models, and also relative to the capability of other CPU types within a model.

The capability value applies to each of the CPUs of one or more applicable CPU types in the configuration. That is, all CPUs in the configuration of an applicable type or types have the same capability. When the value is zero, all CPUs of any CPU type in the configuration have the same capability, as specified by the CPU capability.

The secondary CPU capability may or may not be the same value as the CPU-capability value.

The multiprocessing-CPU-capability-adjustment factors are also applicable to CPUs whose capability is specified by the secondary CPU capability.

CPU Capability: Word 8 specifies the capability of one of the CPUs in the configuration.

If bits 0-8 of word 8 are zero, the word contains a 32-bit unsigned binary integer (I) in the range $1 \leq I < 2^{23}$. If bits 0-8 of word 8 are nonzero, the word contains a 32-bit binary floating point short-format number instead of an unsigned binary integer.

Regardless of encoding, the value represents the capability of one of the CPUs in the configuration, and a lower value indicates a proportionally higher CPU capability. Beyond that, there is no formal description of the algorithm used to generate this value. The value is used as an indication of the capability of the CPU relative to the capability of other CPU models.

The capability value applies to each of the non-secondary CPUs in the configuration. That is, all non-secondary CPUs in the configuration have the same capability.

Total CPU Count: Bytes 0 and 1 of word 9 contain a 16-bit unsigned binary integer that specifies the total number of CPUs in the configuration. This number includes all CPUs in the configured state, the standby state, or the reserved state.

Configured CPU Count: Bytes 2 and 3 of word 9 contain a 16-bit unsigned binary integer that speci-

fies the number of CPUs that are in the configured state. A CPU is in the configured state when it is in the configuration and available to be used to execute programs.

Standby CPU Count: Bytes 0 and 1 of word 10 contain a 16-bit unsigned binary integer that specifies the number of CPUs that are in the standby state. A CPU is in the standby state when it is in the configuration, is not available to be used to execute programs, and can be made available by issuing instructions to place it in the configured state.

Reserved CPU Count: Bytes 2 and 3 of word 10 contain a 16-bit unsigned binary integer that specifies the number of CPUs that are in the reserved state. A CPU is in the reserved state when it is in the configuration, is not available to be used to execute programs, and cannot be made available by issuing instructions to place it in the configured state. (It may be possible to place a reserved CPU in the standby or configured state by means of manual actions.)

Multiprocessing CPU-Capability Adjustment Factors: Beginning with bytes 0 and 1 of word 11, the SYSIB contains a series of contiguous two-byte fields, each containing a 16-bit unsigned binary integer used to form an adjustment factor (fraction) for the value contained in the CPU-capability field. Such a fraction is developed by using the value (V) of the first two-byte field according to one of the following methods:

- If V is in the range $0 < V \leq 100$, a denominator of 100 is indicated which produces a fraction of $V/100$.
- If V is in the range $100 < V \leq 255$, a denominator of 255 is indicated which produces a fraction of $V/255$.
- If V is in the range $255 < V \leq 65,535$, a denominator of 65,536 is indicated which produces a fraction of $V/65,536$.

Thus, the fraction represented by each two-byte field is then developed by dividing the contents of a two-byte field by the indicated denominator.

The number of adjustment-factor fields is one less than the number of CPUs specified in the total-CPU-count field. The adjustment-factor fields correspond to configurations with increasing numbers of CPUs in the configured state. The first adjustment-factor field

corresponds to a configuration with two CPUs in the configured state. Each successive adjustment-factor field corresponds to a configuration with a number of CPUs in the configured state that is one more than that for the preceding field.

Alternate CPU Capability: When the format field has a value of one, if bits 0-8 of word N are zero, the word contains a 32-bit unsigned binary integer (I) in the range $0 \leq I < 2^{23}$ that specifies the announced capability of one of the CPUs in the configuration. If bits 0-8 of word N are nonzero, the word contains a 32-bit binary floating point short-format number instead of a 32-bit unsigned binary integer.

Regardless of encoding, a lower value indicates a proportionally higher CPU capacity. Beyond that, there is no formal description of the algorithm used to generate this value. The value is used as an indication of the announced capability of the CPU relative to the announced capability of other CPU models.

The alternate-capability value applies to each of the CPUs in the configuration. That is, all CPUs in the configuration have the same alternate capability.

Alternate Multiprocessing CPU-Capability Adjustment Factors: Beginning with bytes 0 and 1 of word N+1, the SYSIB contains a series of contiguous two-byte fields, each containing a 16-bit unsigned binary integer used to form an adjustment factor (fraction) for the value contained in the alternate-CPU-capability field. Such a fraction is developed by using the value (V) of the first two-byte field according to one of the following methods:

- If V is in the range $0 < V \leq 100$, a denominator of 100 is indicated which produces a fraction of $V/100$.
- If V is in the range $100 < V \leq 255$, a denominator of 255 is indicated which produces a fraction of $V/255$.
- If V is in the range $255 < V \leq 65,535$, a denominator of 65,536 is indicated which produces a fraction of $V/65,536$.

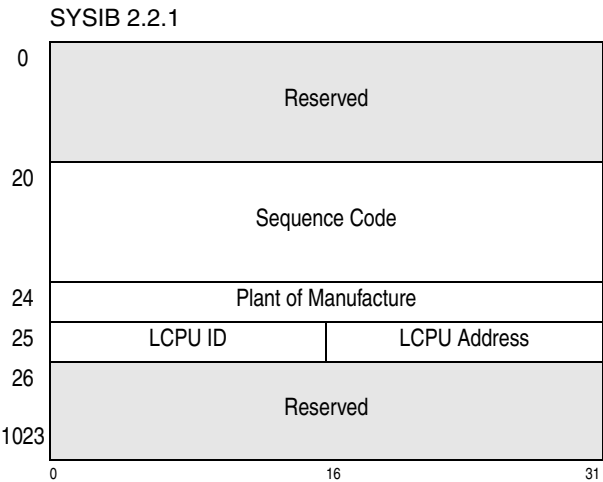
Thus, the fraction represented by each two-byte field is then developed by dividing the contents of a two-byte field by the indicated denominator.

The number of alternate-adjustment-factor fields is one less than the number of CPUs specified in the

total-CPU-count field. The alternate-adjustment-factor fields correspond to configurations with increasing numbers of CPUs in the configured state. The first alternate-adjustment-factor field corresponds to a configuration with two CPUs in the configured state. Each successive alternate-adjustment-factor field corresponds to a configuration with a number of CPUs in the configured state that is one more than that for the preceding field.

SYSIB 2.2.1 (Logical-Partition CPU)

SYSIB 2.2.1 has the following format:



Reserved: The contents of words 0-19 and 26-63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

Sequence Code: Words 20-23 contain the 16-character (0-9 or uppercase A-Z) EBCDIC sequence code of the configuration. The code is right justified with leading EBCDIC zeros if necessary.

Plant of Manufacture: Word 24 contains the four-character (0-9 or uppercase A-Z) EBCDIC code that identifies the plant of manufacture for the configuration. The code is left justified with trailing blanks if necessary.

Logical-CPU ID: Bytes 0 and 1 of word 25 contain a 16-bit unsigned binary integer that can be used in conjunction with the logical-CPU address to distinguish the logical CPU from the other logical CPUs provided by the same LPAR hypervisor.

Logical-CPU Address: Bytes 2 and 3 of word 25 contain the logical-CPU address by which this logical CPU is identified within the level-2 configuration. The logical-CPU address is a 16-bit unsigned binary integer.

The logical-CPU-address field contains the same information as is stored by STORE CPU ADDRESS when the machine is operating in the LPAR mode.

Programming Note: Multiple logical CPUs in the same level-2 configuration have the same logical-CPU sequence code, and it is necessary to use other information, such as the logical-CPU address, to establish a unique logical-CPU identity. The sequence code returned for a basic-machine CPU and a logical-partition CPU are identical and have the same value as the sequence code returned for the basic-machine configuration.

SYSIB 2.2.2 (Logical-Partition CPUs)

SYSIB 2.2.2 has the following format:

SYSIB 2.2.2			
0	Reserved		
8	LPAR Number	Reserved	LCPU
9	Total LCPU Count	Configured LCPU Count	
10	Standby LCPU Count	Reserved LCPU Count	
11	Logical-Partition Name		
12	Logical-Partition CAF		
13	Model-Dependent Data		
14	Reserved		
15	Reserved		
16	Reserved		
17	Reserved		
18	Dedicated LCPU Count	Shared LCPU Count	
19	Reserved		
1023	Reserved		
	0	16	24 31

Reserved: The contents of words 0-7, byte 2 of word 8, words 16-17, and words 19-63 are reserved and are stored as zeros. The contents of words 64-1023 are reserved and may be stored as zeros or may remain unchanged.

Logical-Partition Number: Bytes 0 and 1 of word 8 contain a 16-bit unsigned binary integer which is the number of the level-2 configuration. This number distinguishes the configuration from all other level-2 configurations provided by the same LPAR hypervisor.

Logical-CPU Characteristics (LCPU): The contents of byte 3 of word 8 describe the characteristics of the logical CPUs that are provided for the level-2 configuration. The bits and their meanings are as follows:

Bit Meaning

- 0 Dedicated: When one, bit 0 indicates that one or more of the logical CPUs for this level-2 configuration are provided using level-1 CPUs that are dedicated to this level-2 configuration and are not used to provide logical CPUs for any other level-2 configuration. The number of logical CPUs that are provided using dedicated level-1 CPUs is specified by the dedicated-LCPU-count value in bytes 0 and 1 of word 18.

When zero, bit 0 indicates that none of the logical CPUs for this level-2 configuration are provided using level-1 CPUs that are dedicated to this level-2 configuration.

- 1 Shared: When one, bit 1 indicates that one or more of the logical CPUs for this level-2 configuration are provided using level-1 CPUs that can be used to provide logical CPUs for other level-2 configurations. The number of logical CPUs that are provided using shared level-1 CPUs is specified by the shared-LCPU-count value in bytes 2 and 3 of word 18.

When zero, bit 1 indicates that none of the logical CPUs for this level-2 configuration are provided using shared level-1 CPUs.

- 2 Utilization Limit: When one, bit 2 indicates that the amount of use of the level-1 CPUs that are used to provide the logical CPUs for this level-2 configuration is limited.

When zero, bit 2 indicates that the amount of use of the level-1 CPUs that are used to provide the logical CPUs for this level-2 configuration is unlimited.

- 3-7 Reserved.

Total Logical-CPU Count: Bytes 0 and 1 of word 9 contain a 16-bit unsigned binary integer that specifies the total number of logical CPUs that are provided for this level-2 configuration. This number includes all of the logical CPUs that are in the configured state, the standby state, or the reserved state.

Configured Logical-CPU Count: Bytes 2 and 3 of word 9 contain a 16-bit unsigned binary integer that specifies the number of logical CPUs for this level-2 configuration that are in the configured state.

A logical CPU is in the configured state when it is in the level-2 configuration and is available to be used to execute programs.

Standby Logical-CPU Count: Bytes 0 and 1 of word 10 contain a 16-bit unsigned binary integer that specifies the number of logical CPUs for this level-2 configuration that are in the standby state.

A logical CPU is in the standby state when it is in the level-2 configuration, is not available to be used to execute programs, and can be made available by issuing instructions to place it in the configured state.

Reserved Logical-CPU Count: Bytes 2 and 3 of word 10 contain a 16-bit unsigned binary integer that specifies the number of CPUs for this level-2 configuration that are in the reserved state.

A logical CPU is in the reserved state when it is in the level-2 configuration, is not available to be used to execute programs, and cannot be made available by issuing instructions to place it in the configured state. (It may be possible to place the reserved CPU in the standby or configured state through manual actions.)

Logical-Partition Name: Words 11-12 contain the 8-character EBCDIC name of this level-2 configuration. The name is left justified with trailing blanks if necessary.

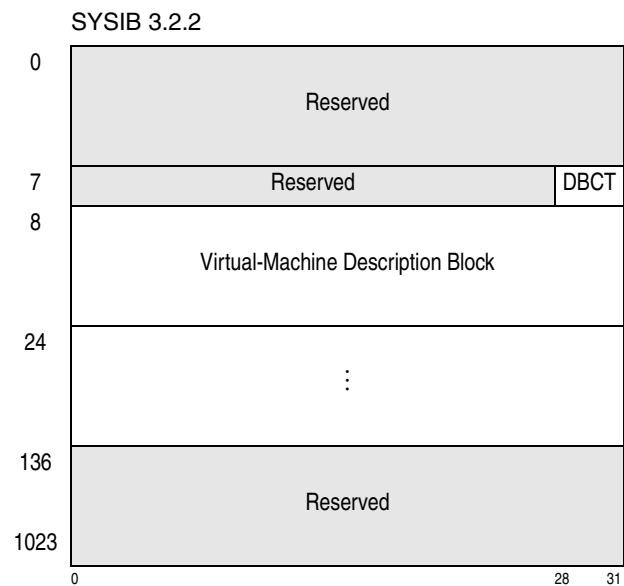
Logical-Partition Capability Adjustment Factor (CAF): Word 13 contains a 32-bit unsigned binary integer, called an adjustment factor, with a value of 1000 or less. The adjustment factor specifies the amount of the underlying level-1-configuration capability that is allowed to be used for this level-2 configuration by the LPAR hypervisor. The fraction of level-1-configuration capability is determined by dividing the CAF value by 1000.

Dedicated Logical-CPU Count: Bytes 0 and 1 of word 18 contain a 16-bit unsigned binary integer that specifies the number of configured-state logical CPUs for this level-2 configuration that are provided using dedicated level-1 CPUs. (See the description of bit 0 of the logical-CPU-characteristics field.)

Shared Logical-CPU Count: Bytes 2 and 3 of word 18 contain a 16-bit unsigned binary integer that specifies the number of configured-state logical CPUs for this level-2 configuration that are provided using shared level-1 CPUs. (See the description of bit 1 of the logical-CPU-characteristics field.)

SYSIB 3.2.2 (Virtual-Machine CPUs)

SYSIB 3.2.2 has the following format:



Reserved: The contents of words 0-6, bits 0-27 of word 7, and words 136-1023 are reserved and are stored as zeros.

Description-Block Count (DBCT): Bits 28-31 of word 7 contain a four-bit unsigned binary integer that specifies the number (up to eight) of virtual-machine description blocks that are stored in the SYSIB beginning at word 8.

Virtual-Machine Description Blocks: Words 8-135 contain from one to eight 64-byte virtual-machine description blocks, depending on the number of nested level-3 configurations, if any, and their processing characteristics.

When a level-3 configuration is provided by a virtual-machine control program and the control program is being executed by a level-3 configuration provided by another virtual-machine control program, the level-3 configurations are said to be “nested.” Level-3 configurations can be nested in this way for several levels.

The collection of nested level-3 configurations that is in the path between a program being executed by a level-3 configuration and the basic machine is called a “level-3-configuration stack.” The level-3 configurations in a stack are consecutively numbered. The level-3 configuration provided by a virtual-machine control program being executed by either a level-2 configuration or a level-1 configuration is the lowest-numbered (0) level-3 configuration in the stack. The level-3 configuration that is executing the program containing this instruction is the highest numbered (N) level-3 configuration in the stack.

If more than one virtual-machine description block is stored in words 8-135 of the SYSIB, the blocks are stored according to the following rules:

- The collection of level-3 configurations described is a contiguous subset of the total collection of level-3 configurations in the level-3-configuration stack. The subset always includes the highest-numbered level-3 configuration in the stack. One or more level-3 configurations at the bottom of the stack may not be described because STORE SYSTEM INFORMATION is not implemented by the highest of those configurations or the limit of eight description blocks would be exceeded.
- The highest-numbered level-3 configuration in the level-3-configuration stack is always described by the first description block in the SYSIB. The lowest-numbered level-3 configuration in the stack, of those that are included in the subset that is described, is described by the last description block in the SYSIB.

The contents of the SYSIB subsequent to the virtual-machine description blocks and prior to word 136 are reserved and are stored as zeros.

The virtual-machine description block has the following format:

	Virtual-Machine Description Block		
0	Reserved		
1	Total LCPU Count	Configured LCPU Count	
2	Standby LCPU Count	Reserved LCPU Count	
3	Virtual-Machine Name		
4			
5	Virtual-Machine CAF		
6	Control-Program Identifier		
10	Reserved		
15			
	0	16	31

Reserved: The contents of words 0 and 10-15 are reserved and are stored as zeros.

Total Logical-CPU Count: Bytes 0 and 1 of word 1 contain a 16-bit unsigned binary integer that specifies the total number of logical CPUs that are provided for this level-3 configuration. This number includes all of the logical CPUs that are in the configured state, the standby state, and the reserved state.

Configured Logical-CPU Count: Bytes 2 and 3 of word 1 contain a 16-bit unsigned binary integer that specifies the number of logical CPUs for this level-3 configuration that are in the configured state.

A logical CPU is in the configured state when it is in the level-3 configuration and is available to be used to execute programs.

Standby Logical-CPU Count: Bytes 0 and 1 of word 2 contain a 16-bit unsigned binary integer that specifies the number of logical CPUs for this level-3 configuration that are in the standby state.

A logical CPU is in the standby state when it is in the level-3 configuration, is not available to be used to execute programs, and can be made available by issuing instructions to place it in the configured state.

Reserved Logical-CPU Count: Bytes 2 and 3 of word 2 contain a 16-bit unsigned binary integer that

specifies the number of CPUs for this level-3 configuration that are in the reserved state.

A logical CPU is in the reserved state when it is in the level-3 configuration, is not available to be used to execute programs, and cannot be made available by issuing instructions to place it in the configured state. (It may be possible to place the logical CPU in the standby or configured state through manual actions.)

Virtual-Machine Name: Words 3-4 contain the eight-character EBCDIC name of this level-3 configuration. The name is left justified with trailing blanks if necessary.

Virtual-Machine Capability Adjustment Factor (CAF): Word 5 contains a 32-bit unsigned binary integer, called an adjustment factor, with a value of 1000 or less. The adjustment factor specifies the amount of the underlying level-1-, level-2-, or level-3-configuration capability that is allowed to be used for this level-3 configuration by the virtual-machine control program. The fraction of the underlying capability is determined by dividing the CAF value by 1000.

Control-Program Identifier: Words 6-9 contain the 16-character EBCDIC identifier of the virtual-machine control program that provides this level-3 configuration. This identifier may include qualifiers such as version number and release level. The identifier is left justified with trailing blanks if necessary.

SYSIB 15.1.2 - 15.1.6 (Configuration Topology)

SYSIBs 15.1.2 through 15.1.6 each have the following format:

SYSIB 15.1.2 - 15.1.6				
0	Reserved		Length	
1	Mag6	Mag5	Mag4	Mag3
2	Mag2	Mag1	Reserved	MNest
3	Reserved			
4	Topology List			
:				
N-1				
N	Reserved			
:				
1023				
	0	8	16	24 31

Selector 2 specifies the MNest value to which the response is limited. STSI completes with condition code 3 for the following cases:

- 1. If the maximum-MNest facility is installed and selector 2 exceeds the nonzero model-dependent maximum-selector-2 value.
- 2. If the maximum-MNest facility is not installed and selector 2 is not specified as two.

Reserved: The contents of bytes 0-1 of word 0, byte 2 of word 2, and word 3 are reserved and are stored as zeros. The contents of words N-1023 are reserved and may be stored as zeros or may remain unchanged.

Length: Bytes 2-3 of word 0 contain a 16-bit unsigned binary integer whose value is the count of bytes of the entire SYSIB 15.1.2. The length of just the topology list is determined by subtracting 16 from the length value in bytes 2-3 of word 0. N in Figure 10-70 is determined by evaluating the formula $N=Length/4$.

Mag1-6: Word 1 and bytes 0-1 of word 2 constitute six one-byte fields where the content of each byte indicates the maximum number of container-type topology-list entries (TLE) or CPU-type TLEs at the corresponding nesting level. CPU-type TLEs are always found only at the Mag1 level. Additionally, the Mag1 value also specifies the maximum number of CPUs that may be represented by a container-type TLE of the Mag2 level. When the value of the nesting level is greater than one, containing nesting levels above the Mag1 level are occupied only by container-type TLEs. A dynamic change to the topology may alter the number of TLEs and the number of CPUs at the Mag1 level, but the limits represented by the values of the Mag1-6 fields do not change within a model family of machines.

The topology is a structured list of entries where an entry defines one or more CPUs or else is involved with the nesting structure. The following illustrates the meaning of the magnitude fields:

- When all CPUs of the machine are peers and no containment organization exists, other than the entirety of the central-processing complex itself, the value of the nesting level is 1, Mag1 is the only non-zero magnitude field, and the number of CPU-type TLEs stored does not exceed the Mag1 value.

- When all CPUs of the machine are subdivided into peer groups such that one level of containment exists, the value of the nesting level is 2, Mag1 and Mag2 are the only non-zero magnitude fields, the number of container-type TLEs stored does not exceed the Mag2 value, and the number of CPU-type TLEs stored within each container does not exceed the Mag1 value.
- The Mag3-6 bytes similarly become used (proceeding in a right-to-left direction) when the value of the nesting level falls in the range 3-6.

MNest: When the maximum-MNest facility is not installed, byte 3 of word 2 specifies the nesting level of the topology to which the configuration may be extended without requiring a re-IPL. The maximum MNest value is model dependent in the range 2-6; the minimum is one. If MNest is one, there is no actual TLE nesting structure, Mag1 is the only non-zero field in the Mag1-6 range, and only CPU-type TLEs are represented in the topology list. The MNest value indicates the number of non-zero magnitude values beginning with the magnitude field at byte 1 of word 2 (Mag1), proceeding left when MNest is greater than one, and with the remaining magnitude fields stored as zeros.

The value of MNest is the maximum possible nesting. No dynamic configuration change exceeds this limit.

When the maximum-MNest facility is installed, the maximum possible nesting is indicated by other means and the MNest value in byte 3 of word 2 reflects both the requested selector-2 value and the number of non-zero magnitude values beginning with the magnitude field at byte 1 of word 2 (Mag1).

Topology List: Words of Figure 10-70 in the range 4 through N-1 specify a list of one or more topology-list entries (TLE). Each TLE is an eight-byte or sixteen-byte field; thus N is an even number of words, and a corollary is that a TLE always starts on a doubleword boundary.

Topology-List Entries: The first TLE in the topology list begins at a nesting level equal to MNest-1. The entire topology list represents the configuration of the issuer of the STSI instruction specifying SYSIB 15.1.2; no outermost container TLE entry is used as it would be redundant with the entire list, and the entire configuration. Therefore, the highest nesting level may have more than a single peer container.

Figure 10-70 illustrates the types of TLEs:

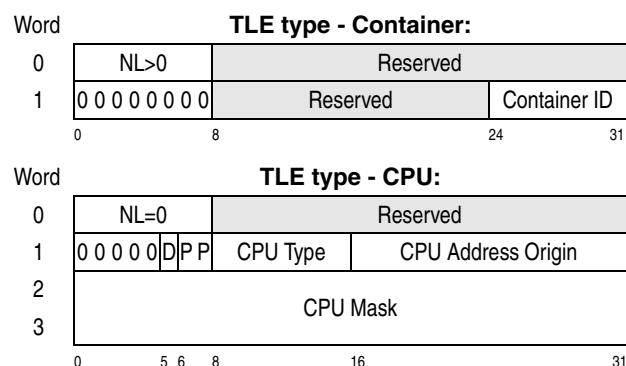


Figure 10-70. Topology-List Entries

Nesting Level (NL): Byte 0 of word 0 specifies the TLE nesting level.

NL Meaning

- 00 The TLE is a CPU-type TLE.
- 01-05 The TLE is a container-type TLE. The first container-type TLE stored in a topology list or a parent container has a container-ID in the range 1-255. If sibling containers exist within the same parent, they proceed in an ascending order of container IDs, that may or may not be consecutive, to a maximum value of 255.

06-FF Reserved

Sibling TLEs have the same value of nesting level which is equivalent to either the value of the nesting level minus one of the immediate parent TLE, or the value of MNest minus one, because the immediate parent is the topology list rather than a TLE.

Reserved, 0: For a container-type TLE, bytes 1-3 of word 0 and bytes 0-2 of word 1 are reserved and stored as zeros. For a CPU-type TLE, bytes 1-3 of word 0 and bits 0-4 of word 1 are reserved and stored as zeros.

Container ID: Byte 3 of word 1 of a container-type TLE specifies an 8-bit unsigned non-zero binary integer whose value is the identifier of the container. The container ID for a TLE is unique within the same parent container.

Dedicated (D): Bit 5 of word 1 of a CPU-type TLE, when one, indicates that the one or more CPUs represented by the TLE are dedicated. When D is zero, the one or more CPUs of the TLE are not dedicated.

Polarization (PP): Bits 6-7 of word 1 of a CPU-type TLE specify the polarization value and, when polarization is vertical, the degree of vertical polarization also called entitlement (high, medium, low) of the corresponding CPU(s) represented by the TLE. The following values are used:

PP	Meaning
0	The one or more CPUs represented by the TLE are horizontally polarized.
1	The one or more CPUs represented by the TLE are vertically polarized. Entitlement is low.
2	The one or more CPUs represented by the TLE are vertically polarized. Entitlement is medium.
3	The one or more CPUs represented by the TLE are vertically polarized. Entitlement is high.

Polarization is only significant in a logical and virtual multiprocessing configuration that uses shared host processors and addresses how the resource assigned to a configuration is applied across the CPUs of the configuration. When horizontal polarization is in effect, each CPU of a configuration is guaranteed approximately the same amount of resource. When vertical polarization is in effect, CPUs of a configuration are classified into three levels of resource entitlement: high, medium, and low.

Both subsystem reset and successful execution of the SIGP set-architecture order specifying ESA/390 mode place a configuration and all of its CPUs into horizontal polarization. The CPUs immediately affected are those that are in the configured state. When a CPU in the standby state is configured, it acquires the current polarization of the configuration and causes a topology change of that configuration to be recognized.

A dedicated CPU is either horizontally or vertically polarized. When a dedicated CPU is vertically polarized, entitlement is always high. Thus, when D is one, PP is either 00 binary or 11 binary.

CPU Type: Byte 1 of word 1 of a CPU-type TLE specifies an 8-bit unsigned binary integer whose value is the CPU type of the one or more CPUs represented by the TLE. The CPU-type value specifies either a primary-CPU type or any one of the possible secondary-CPU types.

CPU-Address Origin: Bytes 2-3 of word 1 of a CPU-type TLE specify a 16-bit unsigned binary integer whose value is the CPU address of the first CPU

in the range of CPUs represented by the CPU mask, and whose presence is represented by the value of bit position 0 in the CPU mask. A CPU-address origin is evenly divisible by 64. The value of a CPU-address origin is the same as that stored by the STORE CPU ADDRESS (STAP) instruction when executed on the CPU represented by bit position 0 in the CPU mask.

CPU Mask: Words 2-3 of a CPU-type TLE specify a 64-bit mask where each bit position represents a CPU. The value of the CPU-address origin field plus a bit position in the CPU mask equals the CPU address for the corresponding CPU.

When a CPU mask bit is zero, the corresponding CPU is not represented by the TLE. The CPU is either not in the configuration or else must be represented by another CPU-type TLE.

When a CPU mask bit is one, the corresponding CPU has the modifier-attribute values specified by the TLE, is in the topology of the configuration, and is not present in any other TLE of the topology.

Thus, for example, if the CPU-address origin is a value of 64, and bit position 15 of the CPU mask is one, CPU 79 is in the configuration and has the CPU type, polarization, entitlement, and dedication as specified by the TLE.

TLE Ordering: The modifier attributes that apply to a CPU-type TLE are CPU type, polarization, entitlement, and dedication. Polarization and entitlement (for vertical polarization) are taken as a single attribute, albeit with four possible values (horizontal, vertical-high, vertical-medium, and vertical-low).

A single CPU TLE is sufficient to represent as many as 64 CPUs that all have the same modifier-attribute values.

When more than 64 CPUs exist, or the entire range of CPU addresses are not covered by a single CPU-address origin, and the modifier attributes are constant, a separate sibling CPU TLE is stored for each CPU-address origin, as necessary, in ascending order of CPU-address origin. Each such TLE stored has at least one CPU represented. The collection of one or more such CPU TLEs is called a CPU-TLE set.

When multiple CPU types exist, a separate CPU-TLE set is stored for each, in ascending order of CPU type.

When multiple polarization-and-entitlement values exist, a separate CPU-TLE set is stored for each, in descending order of polarization value and degree (vertical high, medium, low, then horizontal). When present, all polarization CPU-TLE sets of a given CPU type are stored before the next CPU-TLE set of the next CPU type.

When both dedicated and not-dedicated CPUs exist, a separate CPU-TLE set is stored for each, dedicated appearing before not-dedicated.

All TLEs are ordered assuming a depth-first traversal where the sort order from major to minor is as follows:

1. CPU type
 - a. Lowest CPU-type value
 - b. Highest CPU-type value
2. Polarization-Entitlement
 - a. Vertical high
 - b. Vertical medium
 - c. Vertical low
 - d. Horizontal
3. Dedication (when applicable)
 - a. Dedicated
 - b. Not dedicated

The ordering by CPU-address origin and modifier attributes of sibling CPU TLEs within a parent container is done according to the following list, which proceeds from highest to lowest.

1. CPU-TLE set of lowest CPU-type value, vertical high, dedicated
2. CPU-TLE set of lowest CPU-type value, vertical high, not-dedicated
3. CPU-TLE set of lowest CPU-type value, vertical medium, not-dedicated
4. CPU-TLE set of lowest CPU-type value, vertical low, not-dedicated
5. CPU-TLE set of lowest CPU-type value, horizontal, dedicated

6. CPU-TLE set of lowest CPU-type value, horizontal, not-dedicated
7. CPU-TLE set of highest CPU-type value, vertical high, dedicated
8. CPU-TLE set of highest CPU-type value, vertical high, not-dedicated
9. CPU-TLE set of highest CPU-type value, vertical medium, not-dedicated
10. CPU-TLE set of highest CPU-type value, vertical low, not-dedicated
11. CPU-TLE set of highest CPU-type value, horizontal, dedicated
12. CPU-TLE set of highest CPU-type value, horizontal, not-dedicated

Other TLE Rules :

1. A container-type TLE is located at nesting levels in the range 1-5.
2. A CPU-type TLE is located at nesting level 0.
3. The number of sibling container-type TLEs in a topology list or a given parent container does not exceed the value of the magnitude byte (Mag2-6) of the nesting level corresponding to the siblings.
4. The number of CPUs represented by the one or more CPU-type TLEs of the parent container does not exceed the value of the Mag1 magnitude byte.
5. The content of a TLE is defined as follows:
 - If a TLE is a container-type TLE, the content is a list that immediately follows the parent TLE, comprised of one or more child TLEs, and each child TLE has a nesting level of one less than the nesting level of the parent TLE or topology-list end.
 - If a TLE is a CPU-type TLE, the content is one or more CPUs, as identified by the other fields of a CPU TLE.
6. When the first TLE at a nesting level is a CPU entry, the maximum nesting level 0 has been reached.

CPU Topology Overview

With the advent of the IBM System z9 (2094 z9 EC) and subsequent models, and even previously,

machine organization into nodal structures has resulted in a non-uniform memory access (NUMA) behavior, sometimes also called “lumpiness”. The purpose of the SYSIB 15.1.2 and the PERFORM TOPOLOGY FUNCTION (PTF) instruction is to provide additional machine topology awareness to the program so that certain optimizations can be performed to improve cache-hit ratios and thereby improve overall performance.

The amount of host-CPU resource assigned to a multiprocessing (MP) guest configuration has generally been spread evenly across the number of configured guest CPUs. Such an even spread implies that no particular guest CPU or CPUs are entitled to any extra host-CPU provisioning than any other, arbitrarily-determined guest CPUs. This condition of the guest configuration, affecting all CPUs of the configuration, is called *horizontal polarization*.

Under horizontal polarization, assignment of a host CPU to a guest CPU is approximately the same amount of provisioning for each guest CPU. When the provisioning is not dedicated, the same host CPUs provisioning the guest CPUs also may be used to provision guest CPUs of another guest, or even other guest CPUs of the same guest configuration. When the other guest configuration is a different logical partition, a host CPU, when active in each partition, typically must access main storage more because the cache-hit ratio is reduced by having to share the caches across multiple relocation zones. If host-CPU provisioning can alter the balance such that some host CPUs are mostly, or even exclusively, assigned to a given guest configuration, and that becomes the normal behavior, then cache-hit ratios improve, as does performance. Such an uneven spread implies that one or more guest CPUs are entitled to extra host-CPU provisioning versus other, arbitrarily-determined guest CPUs that are entitled to less host-CPU provisioning. This condition of the guest configuration, affecting all CPUs of the configuration, is called *vertical polarization*.

The architecture categorizes vertical polarization into three levels of entitlement of provisioning, high, medium, and low:

- High entitlement guarantees approximately 100% of a host CPU being assigned to a logical/virtual CPU, and the affinity is maintained as a strong correspondence between the two. With respect to provisioning of a logical partition,

when vertical polarization is in effect, the entitlement of a dedicated CPU is defined to be high.

- Medium entitlement guarantees an unspecified amount of host CPU resource (one or more host CPUs) being assigned to a logical/virtual CPU, and any remaining capacity of the host CPU is considered to be slack that may be assigned elsewhere. The best case for the available slack would be to assign it as local slack if that is possible. A less-beneficial result occurs if that available slack is assigned as remote slack. (See “CPU Slack” on page 10-147 for descriptions of the two slack terms.) It is also the case that the resource percentage assigned to a logical CPU of medium entitlement is a much softer approximation as compared to the 100% approximation of a high-entitlement setting.
- Low entitlement guarantees approximately 0% of a host CPU being assigned to a logical/virtual CPU. However, if slack is available, such a logical/virtual CPU may still receive some CPU resource.

A model of nested containers using polarization is intended to provide a level of intelligence about the machine's nodal structure as it applies to the requesting configuration, so that, generally, clusters of host CPUs can be assigned to clusters of guest CPUs, thereby improving as much as possible the sharing of storage and the minimizing of different configurations essentially colliding on the same host CPUs.

Polarization and entitlement indicate the relationship of physical CPUs to logical CPUs or logical CPUs to virtual CPUs in a guest configuration, and how the capacity assigned to the guest configuration is apportioned across the CPUs that comprise the configuration. Historically, a guest configuration has been horizontally polarized. For however many guest CPUs were defined to the configuration, the host-CPU resource assigned was spread evenly across all of the guest CPUs in an equitable, non-entitled manner. It can be said that the weight of a single logical CPU in a logical partition when horizontal polarization is in effect is approximately equal to the total configuration weight divided by the number of CPUs. However, with the introduction of the 2097 and family models, it becomes imperative to be able to spread the host-CPU resource in a different manner, which is called vertical polarization of a configuration, and then the degree of provisioning of guest CPUs with

host CPUs being indicated as high, medium, or low entitlement. High entitlement is in effect when a logical/virtual CPU of a vertically-polarized configuration is entirely backed by the same host CPU. Medium entitlement is in effect when a logical/virtual CPU of a vertically-polarized configuration is partially backed by a host CPU. Low entitlement is in effect when a logical/virtual CPU of a vertically-polarized configuration is not guaranteed any host-CPU resource, other than what might become available due to slack resource becoming available.

CPU Slack

Regarding slack CPU resource, there are two kinds:

- Local slack becomes available when a logical/virtual CPU of a configuration is not using all the resource to which it is entitled and such slack is then used within the configuration of that CPU. Local slack is preferred over remote slack as better hit ratios on caches are expected when the slack is used within the configuration.
- Remote slack becomes available when a logical/virtual CPU of a configuration is not using all the resource to which it is entitled and such slack is then used outside the configuration of that CPU. Remote slack is expected to exhibit lower hit ratios on caches, but it is still better than not running a logical/virtual CPU at all.

The goal is to maximize the CPU cache hit ratios.

For a logical partition, the amount of physical-CPU resource is determined by the overall system weightings that determine the CPU resource assigned to each logical partition. For example, in a logical 3-way MP that is assigned physical-CPU resource equivalent to a single CPU, and is horizontally polarized, each logical CPU would be dispatched independently and thus receive approximately 33% physical-CPU resource. If the same configuration were to be vertically polarized, only a single logical CPU would be run and would receive approximately 100% of the assigned physical-CPU resource (high entitlement) while the remaining two logical CPUs would not normally be dispatched (low entitlement). Such resource assignment is normally an approximation. Even a low-entitlement CPU may receive some amount of resource if only to help ensure that a program does not get stuck on such a CPU.

By providing a means for a control program to indicate that it understands polarization, and to receive

an indication for each CPU of its polarization and, if vertical polarization, the degree of entitlement, the control program can make more-intelligent use of data structures that are generally thought to be local to a CPU vs. available to all CPUs of a configuration. Also, such a control program can avoid directing work to any low-entitlement CPU.

The actual physical-CPU resource assigned might not constitute an integral number of CPUs, so there is also the possibility of one or more CPUs in an MP vertically-polarized configuration being entitled but not to a high degree, thereby resulting in such CPUs having either medium or low vertical entitlement.

It is possible for any remaining low-entitlement CPUs to receive some amount of host-CPU resource. For example, this may occur when such a CPU is targeted, such as via a SIGP order and slack host-CPU resource is available. Otherwise, such a logical/virtual CPU might remain in an undispached state, even if it is otherwise capable of being dispatched.

A 2-bit polarization field is defined for the CPU-type topology-list entry (TLE) of the STORE SYSTEM INFORMATION (STSI) instruction. The degree of vertical-polarization entitlement for each CPU is indicated as high, medium, or low. The assignment is not a precise percentage but rather is somewhat fuzzy and heuristic.

In addition to vertical polarization as a means of reassigning weighting to guest CPUs, another concept exists, which is the creation and management of slack capacity (also called “whitespace”). Slack capacity is created under the following circumstances:

- A high-vertical CPU contributes to slack when its average utilization (AU) falls below 100 percent (100-AU).
- A medium-vertical CPU that has an assigned provisioning of M percent of a host CPU contributes to slack when its average utilization (AU) falls below M percent ($M - AU > 0$).
- A low-vertical CPU does not contribute to slack.
- A high-vertical CPU is not a consumer of slack.
- A medium-vertical CPU may or may not be a consumer of slack.
- A low-vertical CPU is only a consumer of slack.

Depending upon its utilization and pattern of being dispatched, a horizontally-polarized CPU can be either a contributor to slack or a consumer of it.

Programming Note: A possible examination process of a topology list is described. Before an examination of a topology list is begun, the current-TLE pointer is initialized to reference the first or top TLE in the topology list, the prior-TLE pointer is initialized to null, and then TLEs are examined in a top-to-bottom order.

As a topology-list examination proceeds, the current-TLE pointer is advanced by incrementing the current-TLE pointer by the size of the current TLE to which it points. A container-type TLE is advanced by adding eight to the current-TLE pointer. A CPU-type TLE is advanced by adding sixteen to the current-TLE pointer. The process of advancing the current-TLE pointer includes saving its value as the prior-TLE pointer just before it is incremented. TLE examination is not performed if the topology list has no TLEs.

The examination process is outlined in the following steps:

1. If the current-TLE nesting level is zero, and the prior-TLE nesting level is null or one, the current TLE represents the first CPU-type TLE of a group of one or more CPU-type TLEs. The program should perform whatever action is appropriate for when a new group of one or more CPUs is first observed. Go to step 5.
2. If the current-TLE nesting level is zero, and the prior-TLE nesting level is zero, the current TLE represents a subsequent CPU-type TLE of a group of CPU-type TLEs that represent siblings of the CPUs previously observed in steps 1 or 2. The program should perform whatever action is appropriate for when the size of an existing sibling group of one or more CPUs is increased. Go to step 5.
3. If the current-TLE nesting level is not zero, and the prior-TLE nesting level is zero, the prior TLE represents a last or only CPU-type TLE of a group of one or more CPU-type TLEs. The program should perform whatever action is appropriate for when an existing group of one or more CPUs is completed. Go to step 5.
4. Go to step 5.

The priority of the recognition of exceptions and con-

By elimination, this would be the case where the current-TLE nesting level is not zero, and the prior-TLE nesting level is not zero. If the current-TLE nesting level is less than the prior-TLE nesting level, the direction of topology-list traversal is toward a CPU-type TLE. If the current-TLE nesting level is greater than the prior-TLE nesting level, the direction of topology-list traversal is away from a CPU-type TLE. Container-type TLEs are being traversed leading to either (1) another group of CPU-type TLEs that are a separate group in the overall topology, or (2) the end of the topology list. In either case, no particular processing is required beyond advancing to the next TLE.

5. Advance to the next TLE position based upon the type of the current TLE. If the advanced current-TLE pointer is equivalent to the end of the topology list:
 - a. No more TLEs of any type exist.
 - b. If the prior-TLE nesting level is zero, the program should perform whatever action is appropriate for when an existing group of one or more CPUs is completed.
 - c. The examination is complete.

Otherwise go to step 1.

Special Conditions

The condition code is set to 3 if the function code in bit positions 32-35 of general register 0 is greater than the current-level number.

Bits 36-55 of general register 0 and 32-47 of general register 1 must be zero; otherwise, a specification exception is recognized.

When the function code is valid and nonzero, the following special conditions apply in an unpredictable order:

- The second operand must be designated on a 4K-byte boundary; otherwise, a specification exception is recognized.
- If the function-code, selector-1, and selector-2 combination is invalid, or if it is valid but the requested information is not available, the condition code is set to 3.

dition codes is shown in Figure 10-71.

Resulting Condition Code:

- 0 Requested configuration-level number placed in general register 0 or requested SYSIB information stored
- 1 --
- 2 --
- 3 Requested SYSIB information not available

Program Exceptions:

- Access (store, operand 2, only if function code nonzero)
- Privileged operation
- Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Privileged-operation exception for privileged instruction.
8.	Condition code 3 due to function code greater than current-level number.
9.	Specification exception due to bits 36-55 of general register 0 or bits 32-47 of general register 1 not zero.
10.	Condition code 0 due to function code 0.
11.A	Specification exception due to second-operand address not designating a 4K-byte boundary.
11.B	Condition code 3 due to invalid function-code, selector-1, and selector-2 combination or requested information not available.
12.	Access exceptions (store) for system-information block.
13.	Condition code 0 due to information stored in system-information block.

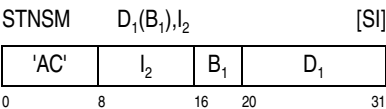
Figure 10-71. Priority of Execution: STORE SYSTEM INFORMATION

Programming Note: The storage-operand references for STORE SYSTEM INFORMATION may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

Program Exceptions:

- Access (store, operand 1)
- Privileged operation

STORE THEN AND SYSTEM MASK



Bits 0-7 of the current PSW are stored at the first-operand location. Then the contents of bit positions 0-7 of the current PSW are replaced by the logical AND of their original contents and the second operand.

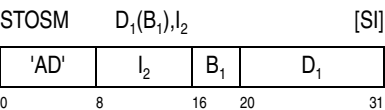
Special Conditions

The operation is suppressed on addressing and protection exceptions.

Condition Code: The code remains unchanged.

Programming Note: STORE THEN AND SYSTEM MASK permits the program to set selected bits in the system mask to zeros while retaining the original contents for later restoration. For example, it may be necessary that a program, which has no record of the present status, disable program-event recording for a few instructions.

STORE THEN OR SYSTEM MASK



Bits 0-7 of the current PSW are stored at the first-operand location. Then the contents of bit positions 0-7 of the current PSW are replaced by the logical OR of their original contents and the second operand.

Special Conditions

The value to be loaded into the PSW is not checked for validity before loading. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, if the contents of bit positions 0 and 2-4 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2. The specification exception, which is listed as a program exception for this instruction, is described in “Early Exception Recognition” on page 6-8. It may be considered as occurring early in the process of preparing to execute the following instruction.

The operation is suppressed on addressing and protection exceptions.

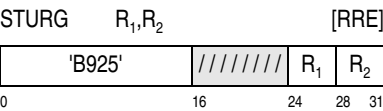
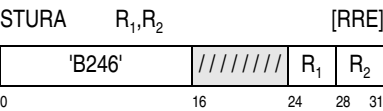
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 1)
- Privileged operation
- Specification

Programming Note: STORE THEN OR SYSTEM MASK permits the program to set selected bits in the system mask to ones while retaining the original contents for later restoration. For example, the program may enable the CPU for I/O interruptions without having available the current status of the external-mask bit.

STORE USING REAL ADDRESS



For STORE USING REAL ADDRESS (STURA), bits 32-63 of general register R₁ are stored in the word at the real-storage location addressed by the contents of general register R₂. For STORE USING REAL ADDRESS (STURG), bits 0-63 of general register R₁ are stored in the doubleword at that real-storage location.

In the 24-bit addressing mode, bits 40-63 of general register R₂ designate the real-storage location, and bits 0-39 of the register are ignored. In the 31-bit addressing mode, bits 33-63 of general register R₂ designate the real-storage location, and bits 0-32 of the register are ignored. In the 64-bit addressing mode, bits 0-63 of general register R₂ designate the real-storage location.

Because it is a real address, the address designating the storage word or doubleword is not subject to dynamic address translation.

Special Conditions

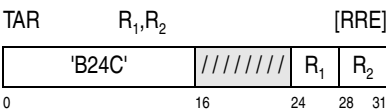
The contents of general register R₂ must designate a location on a word boundary for STURA or on a doubleword boundary for STURG; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (address specified by general register R₂)
- Privileged operation
- Protection (store, operand 2, key-controlled protection and low-address protection)
- Specification

TEST ACCESS



The access-list-entry token (ALET) in access register R₁ is tested for exceptions recognized during access-register translation (ART). The extended authorization index (EAX) used is bits 32-47 of general register R₂. The ALET is also tested for whether it designates the dispatchable-unit access list or the primary-space access list and for whether it is 00000000 or 00000001 hex.

When R₁ is 0, the actual contents of access register 0 are used in ART, instead of the 00000000 hex that is usually used.

Bits 0-31 and 48-63 of general register R₂ are ignored.

The operation does not depend on the translation mode — bits 5, 16, and 17 of the PSW are ignored.

When the ALET specified by means of the R_1 field is other than 00000000 or 00000001 hex, the ART process is applied to the ALET. The EAX specified by means of the R_2 field is called the effective EAX, and it is the EAX which is used by ART. When a condition exists that would normally cause one of the exceptions shown in the following table, the instruction is completed by setting condition code 3.

Exception Name	Cause
ALET specification	ALET bits 0-6 not all zeros
ALEN translation	Access-list entry (ALE) outside list or invalid (bit 0 is one)
ALE sequence	ALE sequence number (ALESN) in ALET not equal to ALESN in ALE
ASTE validity	ASN-second-table entry (ASTE) invalid (bit 0 is one)
ASTE sequence	ASTE sequence number (ASTESN) in ALE not equal to ASTESN in ASTE
Extended authority	ALE private bit not zero, ALE authorization index (ALEAX) not equal to effective EAX, and secondary bit selected by effective EAX either outside authority table or zero

When ART is completed without one of the above conditions being present, the instruction is completed by setting condition code 1 or 2, depending on whether the effective access list is the dispatchable-unit access list or the primary-space access list, respectively. The effective access list is the dispatchable-unit access list if bit 7 of the ALET is zero, or it is the primary-space access list if bit 7 is one. ART, including the obtaining of the effective access-list designation, is described in “Access-Register-Translation Process” on page 5-55.

When the ALET is 00000000 hex, the instruction is completed by setting condition code 0. When the ALET is 00000001 hex, the instruction is completed by setting condition code 3.

Special Conditions

An addressing exception is recognized when the address used by ART to fetch the effective access-

list designation or the ALE, ASTE, or authority-table entry designates a location which is not available in the configuration.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-72 on page 10-152.

Resulting Condition Code:

- 0 Access-list-entry token (ALET) is 00000000 hex
- 1 ALET designates the dispatchable-unit access list and does not cause exceptions in access-register translation (ART)
- 2 ALET designates the primary-space access list and does not cause exceptions in ART
- 3 ALET is 00000001 hex or causes exceptions in ART

Program Exceptions:

- Addressing (effective access-list designation, access-list entry, ASN-second-table entry, or authority-table entry)

Programming Notes:

1. TEST ACCESS permits a called program to check whether an ALET passed from the calling program is authorized for use by means of the calling program's EAX. The calling program's EAX can be obtained from the last linkage-stack state entry by means of EXTRACT STACKED STATE. The called program can thus avoid performing an operation for the calling program, through the use of the called program's EAX, which the calling program is not authorized to perform by means of its own EAX.
2. When an ALET equal to 00000000 hex is passed during a program linkage performed by PROGRAM CALL with space switching (PC-ss), and the ALET conceptually designates the calling program's primary address space and the called program's secondary address space, the ALET must be changed to 00000001 hex before it is used by the called program. Condition code 0 of TEST ACCESS indicates a 00000000 hex ALET so that the ALET can be changed to 00000001 hex by the called program.

3. PROGRAM CALL to current primary (PC-cp) sets the secondary address space equal to the primary address space. PC-ss sets the secondary address space equal to the calling program's primary address space, except that stacking PC-ss sets it equal to the called program's primary address space when the secondary-ASN control in the entry-table entry used is one. In all these cases, a passed 00000001 hex ALET that conceptually designates the calling program's secondary address space is not usable by the called program, even after any transformation (unless the operation was PC-cp and the calling program's PASN and SASN are equal). This is why TEST ACCESS sets condition code 3 when the tested ALET is 00000001 hex.

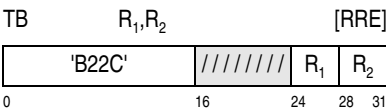
4. After a PC-ss, a passed ALET that conceptually designates an entry in the primary-space access list of the calling program is not usable by the called program. This is why TEST ACCESS sets
- condition code 2, instead of condition code 1, when the tested ALET designates the primary-space access list.

5. The control program may manage the ASN-second-table entry in a way that causes a correctable ASTE-validity or ASTE-sequence exception situation to exist; that is, a situation which, if it were to cause a program interruption during access-register translation, would be corrected by the control program so that access-register translation could be completed successfully. In this case, the program should not use TEST ACCESS directly but should instead use a control-program service that uses TEST ACCESS and that corrects the situation, if possible, when condition code 3 is set. MVS/ESA provides the TESTART macro instruction for use instead of the direct use of TEST ACCESS.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.	Access exceptions for second instruction halfword.
8.	Condition code 0 due to access-list-entry-token (ALET) being 00000000 hex.
9.	Condition code 3 due to ALET being 00000001 hex or ALET bits 0-6 not being all zeros.
10.	Addressing exception for access to effective access-list designation.
11.	Condition code 3 due to access-list entry (ALE) being outside the list.
12.	Addressing exception for access to ALE.
13.	Condition code 3 due to ALE being invalid (bit 0 is 1) or access-list-entry sequence number (ALESN) in the ALET not being equal to the ALESN in the ALE.
14.	Addressing exception for access to ASN-second-table entry (ASTE).
15.	Condition code 3 due to ASTE being invalid (bit 0 is one) or ASTE sequence number (ASTESN) in the ALE not being equal to the ASTESN in the ASTE.
16.	Condition code 3 due to authority-table entry being outside table.
17.	Addressing exception for access to authority-table entry.
18.	Condition code 3 due to ALE private bit not being zero, ALE authorization index (ALEAX) not being equal to effective extended authorization index (EAX), and secondary bit selected by effective EAX being zero.
19.	Condition code 1 if ALET bit 7 is zero; otherwise, condition code 2.

Figure 10-72. Priority of Execution: TEST ACCESS

TEST BLOCK



The storage locations and storage key of a 4K-byte block are tested for usability, and the result of the test is indicated in the condition code. The test for usability is based on the susceptibility of the block to the occurrence of invalid checking-block code.

The block tested is addressed by the contents of general register R_2 . The contents of general register R_1 are ignored.

A complete testing operation is necessarily performed only when the initial contents of bit positions 32-63 of general register 0 are zero in the 24-bit or 31-bit addressing mode, or the initial contents of bit positions 0-63 of that register are zero in the 64-bit addressing mode. In the 24-bit or 31-bit addressing mode, the contents of bit positions 32-63 of general register 0 are set to zero at the completion of the operation, and bits 0-31 of the register always are ignored and remain unchanged. In the 64-bit addressing mode, the contents of bit positions 0-63 of the register are set to zero at the completion of the operation.

If the block is found to be usable, the 4K bytes of the block are cleared to zeros, the contents of the storage key are unpredictable, and condition code 0 is set. If the block is found to be unusable, the data and the storage key are set, as far as is possible by the model, to a value such that subsequent fetches to the area do not cause a machine-check condition, and condition code 1 is set.

In the 24-bit addressing mode, bits 40-51 of general register R_2 designate a 4K-byte block in real storage, and bits 0-39 and 52-63 of the register are ignored. In the 31-bit addressing mode, bits 33-51 of the register designate the block, and bits 0-32 and 52-63 are ignored. In the 64-bit addressing mode, bits 0-51 of the register designate the block, and bits 52-63 are ignored.

The address of the block is a real address, and the accesses to the block designated by the second-operand address are not subject to key-controlled, access-list-controlled, and DAT protection. Low-address protection does apply. The operation is terminated on addressing and protection exceptions. If termination occurs, the condition code and the contents of bit positions 32-63 of general register 0 are unpredictable in the 24-bit or 31-bit addressing mode, or the condition code and bits 0-63 of the register are unpredictable in the 64-bit addressing mode. The contents of the storage block and its associated

storage key are not changed when these exceptions occur.

Depending on the model, the test for usability may be performed (1) by alternately storing and reading out test patterns to the data and storage key in the block or (2) by reference to an internal record of the usability of the blocks which are available in the configuration, or (3) by using a combination of both mechanisms.

In models in which an internal record is used, the block is indicated as unusable if a solid failure has been previously detected, or if intermittent failures in the block have exceeded the threshold implemented by the model. In such models, depending on the criteria, attempts to store may or may not occur. Thus, if block 0 is not usable, and no store occurs, low-address protection may or may not be indicated.

In models in which test patterns are used, TEST BLOCK may be interruptible. When an interruption occurs after a unit of operation, other than the last one, the condition code is unpredictable, and the contents of bit positions 32-63 of general register 0 may contain a record of the state of intermediate steps in the 24-bit or 31-bit addressing mode, or the contents of bit positions 0-63 may contain that record in the 64-bit addressing mode. When execution is resumed after an interruption, the condition code is ignored, but the record in general register 0 may be used to determine the resumption point.

If (1) TEST BLOCK is executed with an initial value other than zero in bit positions 32-63 of general register 0 in the 24-bit or 31-bit addressing mode or bit positions 0-63 in the 64-bit addressing mode, or (2) the interrupted instruction is resumed after an interruption with a value in bit positions 32-63 or 0-63 (depending on the addressing mode) of general register 0 or a value in the storage block or its associated storage key other than the corresponding value which was present at the time of the interruption, or (3) the block or its associated storage key is accessed by another CPU or by the channel subsystem during the execution of the instruction, then the contents of the storage block, its associated storage key, and bit positions 32-63 or 0-63 of general register 0 are unpredictable, along with the resultant condition-code setting.

Invalid checking-block-code errors initially found in the block or encountered during the test do not normally result in machine-check conditions. The test-

block function is implemented in such a way that the frequency of machine-check interruptions due to the instruction execution is not significant. However, if, during the execution of TEST BLOCK for an unusable block, that block is accessed by another CPU (or by the channel subsystem), error conditions may be reported both to this CPU and to the other CPU (or to the channel subsystem).

A serialization function is performed before the block is accessed and again after the operation is completed (or partially completed).

The priority of the recognition of exceptions and condition codes is shown in Figure 10-73 on page 10-154.

Resulting Condition Code:

- 0 Block usable
- 1 Block not usable
- 2 --
- 3 --

Program Exceptions:

- Addressing (fetch and store, operand 2)
- Privileged operation
- Protection (store, operand 2, low-address protection only)

Programming Notes:

1. The execution of TEST BLOCK on most models is significantly slower than that of the MOVE LONG instruction with padding; therefore, the instruction should not be used for the normal case of clearing storage.
2. The program should use TEST BLOCK at initial program loading and as part of the vary-storage-online procedure to determine if blocks of storage exist which should not be used.
3. The program should use TEST BLOCK when an uncorrected error is reported in either the data or storage key of a block. This is because in the execution of TEST BLOCK the attempt is made, as far as is possible on the model, to leave the contents of a block in a state such that subsequent prefetches or unintended references to the block do not cause machine-check conditions. The program may use the resulting condition

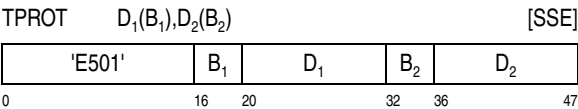
1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Privileged-operation exception.
8.	Addressing exception due to block not being available in the configuration.*
9.A	Condition code 1, block not usable.
9.B	Protection exception due to low-address protection.*
10.	Condition code 0, block usable and set to zeros.
Explanation:	
*	The operation is terminated on addressing and protection exceptions, and the condition code may be unpredictable.

Figure 10-73. Priority of Execution: TEST BLOCK

code in this case to determine if the block can be reused. (The block could be indicated as usable if, for example, the error were an externally generated error or an indirect storage error.) This procedure should be followed regardless of whether the indirect-storage-error indication is reported.

4. The model may or may not be successful in removing the errors from a block when TEST BLOCK is executed. The program therefore should take every reasonable precaution to avoid referencing an unusable block. For example, the program should not place the page-frame real address of an unusable block in an attached and valid page-table entry.
5. On some models, machine checks may be reported for a block even though the block is not referenced by the program. When a machine check is reported for a storage-key error in a block which has been marked as unusable by the program, it is possible that SET STORAGE KEY EXTENDED may be more effective than TEST BLOCK in validating the storage key.
6. The storage-operand references for TEST BLOCK may be multiple-access references. (See "Storage-Operand Consistency" on page 5-95.)

TEST PROTECTION



The location designated by the first-operand address is tested for protection exceptions by using the access key specified by bits 56-59 of the second-operand address.

The second-operand address is not used to address data; instead, bits 56-59 of the address form the access key to be used in testing. Bits 0-55 and 60-63 of the second-operand address are ignored.

The first-operand address is a logical address. When the CPU is in the access-register mode (when DAT is on and PSW bits 16 and 17 are 01 binary), the first-operand address is subject to translation by means of both the access-register-translation (ART) and the dynamic-address-translation (DAT) processes. ART applies to the access register designated by the B₁ field, and it obtains the address-space-control element to be used by DAT. When DAT is on but the CPU is not in the access-register mode, the first-operand address is subject to translation by DAT. In this case, DAT uses the address-space-control element contained in control register 1, 7, or 13 when the CPU is in the primary-space, secondary-space, or home-space mode, respectively. When DAT is off, the first-operand address is a real address not subject to translation by either ART or DAT.

When the CPU is in the access-register mode and an address-space-control element cannot be obtained by ART because of a condition that would normally

cause one of the exceptions shown in the following table, the instruction is completed by setting condition code 3.

Exception Name	Cause
ALET specification	Access-list-entry-token (ALET) bits 0-6 not all zeros
ALEN translation	Access-list entry (ALE) outside list or invalid (bit 0 is one)
ALE sequence	ALE sequence number (ALESN) in ALET not equal to ALESN in ALE
ASTE validity	ASN-second-table entry (ASTE) invalid (bit 0 is one)
ASTE sequence	ASTE sequence number (ASTESN) in ALE not equal to ASTESN in ASTE
Extended authority	ALE private bit not zero, ALE authorization index (ALEAX) not equal to extended authorization index (EAX), and secondary bit selected by EAX either outside authority table or zero

When the access register contains 00000000 hex or 00000001 hex, ART obtains the address-space-control element from control register 1 or 7, respectively, without accessing the access list. When the B₁ field designates access register 0, ART treats the access register as containing 00000000 hex and does not examine the actual contents of the access register.

When ART is completed successfully, the operation is continued through the performance of DAT.

When DAT is on and the first-operand address cannot be translated because of a condition that would normally cause one of the exceptions shown in the

following table, the instruction is completed by setting condition code 3.

Exception Name	Cause
ASCE type	Address-space-control element (ASCE) being used is a region-second-table designation, and bits 0-10 of first-operand address not all zeros; ASCE is a region-third-table designation, and bits 0-21 of first-operand address not all zeros; or ASCE is a segment-table designation, and bits 0-32 of first-operand address not all zeros.
Region first translation	Region-first-table entry outside table or invalid.
Region second translation	Region-second-table entry outside table or invalid.
Region third translation	Region-third-table entry outside table or invalid.
Segment translation	Segment-table entry outside table or invalid
Page translation	Page-table entry invalid

When translation of the first-operand address can be completed, or when DAT is off, the storage key for the block designated by the first-operand address is tested against the access key specified in bit positions 56-59 of the second-operand address, and the condition code is set to indicate whether store and fetch accesses are permitted, taking into consideration all applicable protection mechanisms. Thus, for example, if low-address protection is active and the first-operand effective address is in the range 0-511 or 4096-4607, then a store access is not permitted. Access-list-controlled protection, DAT protection, storage-protection override, and fetch-protection override also are taken into account.

When enhanced DAT does not apply, or when enhanced DAT applies but the STE-format control is zero, bits 0-4 of the storage key for the 4K-byte block corresponding to the virtual address in the first operand are examined to determine the condition code. When enhanced DAT applies and the STE-format control is one, it is unpredictable whether bits 0-4 of the storage key or bits 48-52 of the segment-table entry used in the translation of the first operand are examined to determine the condition code.

The contents of storage, including the change bit, are not affected. Depending on the model, the reference bit for the first-operand address may be set to one, even for the case in which the location is protected against fetching.

Special Conditions

When the CPU is in the access-register mode, an addressing exception is recognized when the address used by ART to fetch the effective access-list designation or the ALE, ASTE, or authority-table entry designates a location which is not available in the configuration.

When DAT is on, an addressing exception is recognized when the address of the region-table entry or entries, segment-table entry, or page-table entry or the operand real address after translation designates a location which is not available in the configuration. Also, a translation-specification exception is recognized when a region-table entry or the segment-table entry or page-table entry has a format error, that is, when any of the reasons listed in "Translation-Specification Exception" on page 6-40 applies. When DAT is off, only the addressing exception due to the operand and real address applies.

For all of the above cases, the operation is suppressed.

Resulting Condition Code:

- 0 Fetching permitted; storing permitted
- 1 Fetching permitted; storing not permitted
- 2 Fetching not permitted; storing not permitted
- 3 Translation not available

Program Exceptions:

- Addressing (effective access-list designation, access-list entry, ASN-second-table entry, authority-table entry, region-table entry, segment-table entry, page-table entry, or operand 1)
- Privileged operation
- Translation specification

Programming Notes:

1. TEST PROTECTION permits a program to determine the protection attributes of an address passed from a calling program without incurring program exceptions. The instruction sets a condition code to indicate whether fetching or storing

is permitted at the location designated by the first-operand address of the instruction. The instruction takes into consideration all of the protection mechanisms in the machine: access-list controlled, DAT, key-controlled, and low-address protection, storage-protection override, and fetch-protection override. Additionally, since ASCE-type, region-translation, segment-translation, and page-translation-exception conditions may be a program substitute for a protection violation, these conditions are used to set the condition code rather than cause a program exception.

When the CPU is in the access-register mode, TEST PROTECTION additionally permits the program to check the usability of an access-list-entry token (ALET) in an access register without incurring program exceptions. The ALET is checked for validity (absence of an ALET-specification, ALEN-translation, and ALE-sequence-exception condition) and for being authorized for use by the program (absence of an ASTE-validity, ASTE-sequence, and extended-authority-exception condition).

2. See the programming notes under SET PSW KEY FROM ADDRESS for more details and for an alternative approach to testing the key-controlled protection attributes of addresses passed by a calling program. The approach using TEST PROTECTION has the advantage of a test which does not result in interruptions; however, the test and use are separated in time and may not be accurate if the possibility exists that the storage key of the location in question can change between the time it is tested and the time it is used.
3. In the handling of dynamic address translation, TEST PROTECTION is similar to LOAD REAL ADDRESS in that the instructions do not cause ASCE-type, region-translation, segment-translation, and page-translation exceptions. Instead, these exception conditions are indicated by means of a condition-code setting. Similarly, access-register translation sets a condition code for certain exception conditions when performed during either of the two instructions. Conditions which result in condition codes 1, 2, and 3 for LOAD REAL ADDRESS result in condition code 3 for TEST PROTECTION. The instructions also differ in several other respects. The first-operand address of TEST PROTECTION is a logical address and thus is not subject to dynamic address translation when DAT is off. The second-

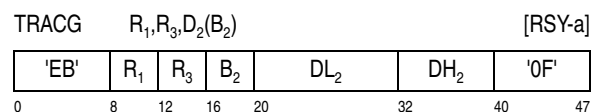
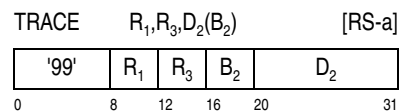
operand address of LOAD REAL ADDRESS is a virtual address which is always translated.

Access-register translation applies to TEST PROTECTION only when the CPU is in the access-register mode (DAT is on), whereas it applies to LOAD REAL ADDRESS when PSW bits 16 and 17 are 01 binary regardless of whether DAT is on or off. When condition code 3 is set because of an exception condition in access-register translation, LOAD REAL ADDRESS, but not TEST PROTECTION, returns in a general register the program-interruption code assigned to the exception.

4. Condition code 3 does not necessarily indicate that the first-operand location will always be inaccessible to the program; rather it merely indicates that the current conditions prevent the instruction from determining the protection attributes of the operand. For example, in a virtual storage environment, condition code 3 may be set if the storage location has been paged out by the operating system. If the program attempts to access the location, the operating system may resolve the page-translation exception and subsequently make the location accessible to the program.

Similarly, condition code 1 does not necessarily indicate that the address cannot ever be stored into. In an operating system that implements a Posix fork function, DAT protection is used to alert the operating system of a copy-on-write event (as described in Programming Note 1 on page 3-15). Following the operating-system resolution of the copy-on-write event, the program may be given store access to the location.

TRACE



When explicit tracing is on (bit 63 of control register 12 is one), the second operand, which is a 32-bit word in storage, is fetched, and bit 0 of the word is

examined. If bit 0 of the second operand is zero, a trace entry is formed at the real-storage location designated by control register 12.

If explicit tracing is off (bit 63 of control register 12 is zero), or if bit 0 of the second operand is one, no trace entry is formed, and no trace exceptions are recognized.

The displacement for TRACE is treated as a 12-bit unsigned binary integer. The displacement for TRACG is treated as a 20-bit signed binary integer.

The trace entry is composed of an entry-type identifier, a count of the number of general registers whose partial or entire contents are placed in the entry, a field whose contents indicate whether the entry was formed by TRACE (TRACE) or TRACE (TRACG), selected bits of the TOD clock, the second operand, and the partial or entire contents of a range of general registers. For TRACE (TRACE), bits 16-63 of the TOD clock and bits 32-63 of the general registers are placed in the trace entry. For TRACE (TRACG), bits 0-79 of the clock and bits 0-63 of the registers are placed in the entry.

The general registers are stored in ascending order of their register numbers, starting with general register R_1 and continuing up to and including general register R_3 , with general register 0 following general register 15. The trace table and the trace-entry formats are described in "Tracing" on page 4-10.

When a trace entry is made, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed. However, it is unpredictable whether or not a store into a trace-table entry from which a subsequent instruction is fetched will be observed by the CPU that performed the store. Additionally, when the store-clock-fast facility is installed and the TRACE TOD-clock control in bit 32 of control register 0 is one, it is unpredictable whether explicit tracing causes serialization to be performed.

Special Conditions

A privileged-operation exception is recognized in the problem state, even when explicit tracing is off or bit 0 of the second operand is one.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized. It is unpredictable whether the specification exception is recognized when explicit tracing is off.

It is unpredictable whether access exceptions are recognized for the second operand when explicit tracing is off.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Privileged operation
- Specification
- Trace

Programming Notes:

1. Bits 1-15 of the second operand are reserved for model-dependent functions and should therefore be set to zeros.
2. When the store-clock-fast facility is not installed, or when the TOD-clock-control in bit 32 of control register 0 is zero, the entire 32-bit TRACE operand is stored in bits 64-95 of the trace entry for TRACE (TRACE) and in bits 96-127 of the trace entry for TRACE (TRACG).
3. When the store-clock-fast facility is installed and the TRACE TOD-clock control in bit 32 of control register 0 is one, the following conditions apply:
 - Two executions of the instruction do not necessarily store different values in the TOD-clock bits of the trace entry.
 - The contents of bit positions 0-7 and 16-31 of the TRACE operand are stored in bits 64-71 and 80-95, respectively, of the trace entry for TRACE (TRACE) and in bits 96-103 and 112-127, respectively, of the trace entry for TRACE (TRACG).
 - Bits 72-79 of the trace entry for TRACE (TRACE) and bits 104-111 of the trace entry for TRACE (TRACG) are set to a model-dependent value.

TRAP



A trap operation is performed if the CPU is in the primary-space or access-register mode and the TRAP-enabled bit in byte 47 of the dispatchable-unit control table (DUCT) is one. Otherwise, a special-operation exception is recognized.

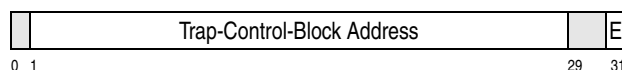
The trap operation obtains a trap-control-block address from the DUCT and then a trap-save-area address and a trap-program address from the trap control block. State information is stored in the trap save area. Then the trap-control-block address is loaded into general register 15. Finally, the current PSW is updated by setting the basic-addressing-mode bit to one (which will leave the addressing mode as either the 31-bit mode or the 64-bit mode or will change the addressing mode from the 24-bit mode to the 31-bit mode) and the address-space-control bits to zeros (primary-space mode) and by replacing the instruction address with the trap-program address. Compatibility with the ESA/390 operation of TRAP optionally is provided.

For TRAP4, the second-operand address is not used to address data; instead, bits 33-63 of the address are stored in the trap save area.

Dispatchable-Unit Control Table

Bytes 44-47 (word 10) of the dispatchable-unit-control table (DUCT) are used by this instruction. The contents of those bytes are as follows:

DUCT Bytes 44-47



The fields in bytes 44-47 of the DUCT are allocated as follows:

Trap-Control-Block Address: Bits 1-28, with three zeros appended on the right, form the 31-bit home

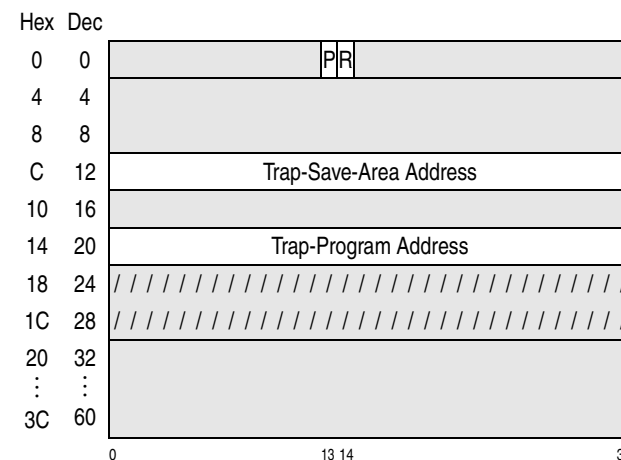
virtual address of the trap control block. This address is treated as a 31-bit home virtual address regardless of the current addressing mode and regardless of the current value of the address-space-control bits. This address, with a zero appended on the left, is placed in bit positions 32-63 of general register 15 after the contents of that register have been saved in the trap save area. If the current addressing mode is the 64-bit mode, bits 0-31 of general register 15 are set to zeros.

TRAP-Enabled Bit (E): Bit 31 specifies, when one, that the trap operation is to be performed. TRAP recognizes a special-operation exception if bit 31 is zero.

Bits 0, 29, and 30 of bytes 44-47 are ignored, but they should be zeros to permit possible future extensions.

Trap Control Block

The trap control block is 64 bytes aligned on a doubleword boundary. The format of the trap control block is:



The fields in the trap control block are allocated as follows:

PSW Control (P): Bit 13 of bytes 0-3 controls the allowed value of bit 31 of the current PSW and how bits 12 and 33-127 of the current PSW are stored in the PSW-values field in the trap save area. When bit 13 is zero.

- Bit 31 of the current PSW, the extended-addressing-mode bit, must be zero; otherwise, a special-operation exception is recognized.

- A one is stored in bit position 12 of the PSW-values field even though bit 12 of the current PSW is zero.
- Bits 97-127 of the current PSW are stored in bit positions 33-63 of the PSW-values field, bits 33-96 of the current PSW are not stored, and zeros are stored in bit positions 64-127 of the PSW-values field.

When bit 13 is one:

- Bit 31 of the current PSW may be zero or one.
- Bit 12 of the current PSW is stored in bit position 12 of the PSW-values field.
- Bits 64-127 of the current PSW are stored in bit positions 64-127 of the PSW-values field.

General-Registers Control (R): Bit 14 of bytes 0-3 controls how the contents of the general registers are stored in the general-registers 0-15 field in the trap save area. When R is zero, bits 32-63 of the general registers are stored in consecutive four-byte locations beginning at the beginning of the general-registers 0-15 field, bits 0-31 of the registers are not stored, and the last 64 bytes of the general-registers 0-15 field remain unchanged. When R is one, bits 0-63 of the general registers are stored in consecutive eight-byte locations in the general-registers 0-15 field.

Trap-Save-Area Address: Bits 1-28 of bytes 12-15, with three zeros appended on the right, form the 31-bit home virtual address of the trap save area. This address is treated as a 31-bit home virtual address regardless of the current addressing mode and regardless of the current value of the address-space-control bits. Bits 0 and 29-31 of bytes 12-15 are ignored.

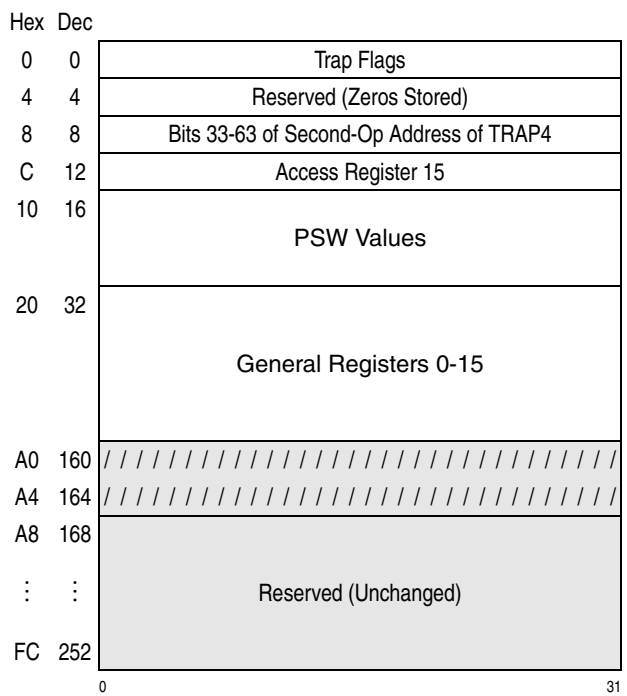
Trap-Program Address: Bits 1-31 of bytes 20-23 form the 31-bit primary virtual address of the trap program. This address is treated as a 31-bit primary virtual address regardless of the current addressing mode.

Bit positions 0-12 and 15-31 of bytes 0-3 and bytes 4-11, 16-19, and 32-63 of the trap control block are reserved and should contain zeros. Bytes 24-31 are available for use by programming.

Trap Save Area

The trap save area is 256 bytes aligned on a double-word boundary.

The trap operation stores information into the trap save area as follows:



The fields in the trap save area are allocated as follows:

Trap Flags: Information identifying the instruction(s) causing the trap operation is stored in byte positions 0-3. The detailed format of bytes 0-3 is as follows:

Flag	Meaning
Bits	
0	TRAP was target of an execute-type instruction
1	TRAP is TRAP4 (not TRAP2)
2-12	Reserved, zeros stored
13-14	Instruction-length code (ILC)
15-31	Reserved, zeros stored

Bit 0 of bytes 0-3 is set to one if TRAP was the target of an execute-type instruction (EXECUTE or EXECUTE RELATIVE LONG).

Bit 1 of bytes 0-3 is set to one if TRAP is TRAP4 (not TRAP2).

Bits 13 and 14 are the instruction-length code (ILC) that specifies the length of the TRAP instruction, or the length of the execute-type instruction if TRAP was the target of an execute-type instruction.

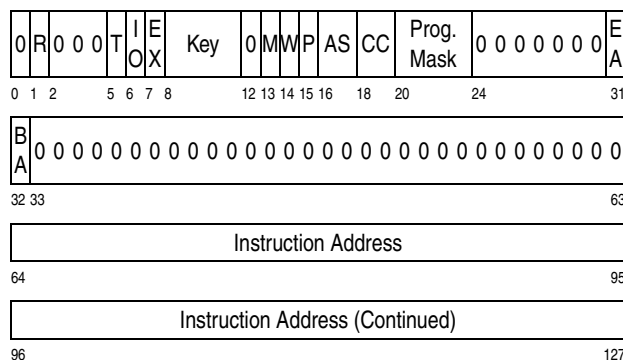
Bits 2-12 and 15-31 are reserved and are stored as zeros.

Bits 33-63 of Second-Operand Address of TRAP4: For TRAP4, bits 33-63 of the second-operand address, generated under the control of the current addressing mode and with a zero appended on the left, are stored in byte positions 8-11. Only bits 33-63 of the second-operand address are stored even when the current addressing mode is the 64-bit mode. For TRAP2, all zeros are stored in byte positions 8-11.

Access Register 15: The contents of access register 15 are stored in byte positions 12-15.

PSW Values: The following description applies when the PSW control, bit 13 of bytes 0-3 of the trap control block, is one.

Certain information from the current PSW is stored in byte positions 16-31. The PSW has the following format:



Bits 0-127 of bytes 16-31 correspond one-to-one with bits 0-127 of the PSW. For some bit positions of bytes 16-31, the corresponding PSW bits are stored. For the other bit positions of bytes 16-31, unpredict-

able values are stored. Information is stored in bytes 16-31 as follows:

Bits	Value
0	Zero
1	Unpredictable
2-4	Zero
5-11	Unpredictable
12	Zero
13	Unpredictable
14	Wait state (W)
15	Problem state (P)
16-17	Address-space control (AS)
18-19	Condition code (CC)
20-23	Program mask
24-30	Zero
31	Extended addressing mode (EA)
32	Basic addressing mode (BA)
33-63	Zero
64-127	Instruction address

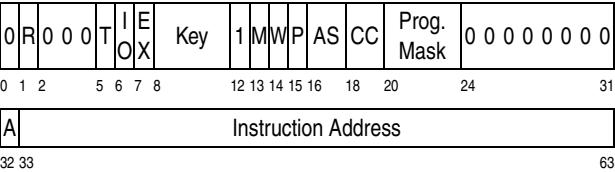
In summary, bits 0, 2-4, 12, 24-30, and 33-63 are zero, bits 1, 5-11, and 13 are unpredictable, and the other bits are set with variable information from the PSW.

The wait-state, problem-state, address-space-control, condition-code, program-mask, extended-addressing-mode, and basic-addressing-mode values specify the state of the CPU before the TRAP instruction was executed. The instruction-address value is the updated instruction address, which is the address of the instruction following TRAP, or the address of the instruction following the execute-type instruction if TRAP was the target of an execute-type instruction.

When the PSW control in the trap control block is zero, the operation is as described above except as follows:

- Bit 31 of the current PSW must be zero; otherwise, a special-operation exception is recognized.
- A one is stored in bit position 12 of bytes 16-31.
- Bits 97-127 of the current PSW are stored in bit positions 33-63 of bytes 16-31, bits 33-96 of the current PSW are not stored, and zeros are stored in bit positions 64-127 of bytes 16-31 (bytes 24-31).

In this case, bytes 16-23 have the format of an ESA/390 PSW, which is as follows:



General Registers 0-15: Contents of general registers 0-15 are stored in byte positions 32-159 as described in “General-Registers Control (R)” on page 10-160. When bits 32-63 or 0-63 of the general registers are stored, they are stored in ascending order of register numbers, starting with register 0 and continuing up to and including register 15.

Bytes 160-255 always remain unchanged. Bytes 168-255 are reserved. Bytes 160-167 are available for use by programming.

Special Conditions

The CPU must be in the primary-space mode or access-register mode, and bit 31 in bytes 44-47 of the dispatchable-unit control table must be one; otherwise, a special-operation exception is recognized. A special-operation exception is also recognized if the PSW control, bit 13 of bytes 0-3 of the trap control block is zero and bit 31 of the current PSW, the extended-addressing-mode bit, is one.

All protection mechanisms apply in the usual way to the accesses to the trap control block and trap save area. Access exceptions may or may not be recognized for sections of the trap control block and trap save area that are not referenced by the TRAP instruction.

The trap-program address in the trap control block is not tested before it replaces the instruction address in the PSW. An odd address will cause a specification exception to be recognized as part of the execution of the next instruction.

The operation is suppressed on all addressing and protection exceptions.

The priority of recognition of program exceptions for the instruction is shown in Figure 10-74 on page 10-163.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, trap control block; store, trap save area)
- Addressing (dispatchable-unit control table)
- Special operation
- Trace

Programming Notes:

1. It is intended that TRAP instructions will overlay instructions in an application program in order to give control to a trap program, which might be a program for performing fix-ups of data used by the application program, such as dates that may be a “Year-2000” problem. TRAP2 can overlay a two-byte instruction, and TRAP4 can overlay a four-byte instructions or the first four bytes of a six-byte instruction. The trap program is to simulate the overlaid instruction and perform fix-ups as appropriate, and it is then to return control to the application program.
2. The trap program can use the RESUME PROGRAM instruction to return control to the application program. For example, the trap program can restore the contents of all registers except access and general registers 15, and then, using those registers (or at least the general register) to address the trap save area, can restore the contents of those registers and also PSW fields from the trap save area. RESUME PROGRAM has control bits in its parameter list that allow it to restore PSW fields from a field having either the ESA/390 PSW format or the z/Architecture PSW format and to restore either bits 32-63 or 0-63 of a general register.
3. The trap control block and trap save area are in the home address space, and the trap program is in the primary address space. The trap-control-block address placed in general register 15 by TRAP can be useful to the trap program if (1) the primary address space and home address space are the same address space, (2) the trap control block and trap save area are at the same locations in the primary address space as in the home address space, or (3) the trap program can use access registers to access the home address space.

4. The storage-operand references for TRAP may be multiple-access references. (See “Storage-Operand Consistency” on page 5-95.)

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword (TRAP4 only).
7.B	Special-operation exception due to the CPU not being in the primary-space mode or access-register mode.
7.C.1	Addressing exception for access to dispatchable-unit control table.
7.C.2	Special-operation exception due to bit 31 in bytes 44-47 of dispatchable-unit control table being zero.
8.A	Trace exceptions.
8.B.1	Access exceptions (fetch) for trap control block.
8.B.2	Special-operation exception due to PSW control in trap control block being zero and PSW bit 31 being one.
8.B.3	Access exceptions (store) for trap save area.

Figure 10-74. Priority of Execution: TRAP

Chapter 11. Machine-Check Handling

Machine-Check Detection	11-2	Processing Damage	11-17
Correction of Machine Malfunctions	11-2	Storage Errors	11-18
Error Checking and Correction	11-2	Storage Error Uncorrected	11-18
CPU Retry	11-2	Storage Error Corrected	11-18
Effects of CPU Retry	11-3	Storage-Key Error Uncorrected	11-18
Checkpoint Synchronization	11-3	Storage Degradation	11-18
Handling of Machine Checks during		Indirect Storage Error	11-19
Checkpoint Synchronization	11-3	Machine-Check Interruption-Code Validity	
Checkpoint-Synchronization Operations ..	11-3	Bits	11-19
Checkpoint-Synchronization Action	11-4	PSW-MWP Validity	11-19
Channel-Subsystem Recovery	11-4	PSW Mask and Key Validity	11-19
Unit Deletion	11-4	PSW Program-Mask and Condition-Code	
Handling of Machine Checks	11-4	Validity	11-19
Validation	11-5	PSW-Instruction-Address Validity	11-20
Invalid CBC in Storage	11-6	Failing-Storage-Address Validity	11-20
Programmed Validation of Storage	11-7	External-Damage-Code Validity	11-20
Invalid CBC in Storage Keys	11-7	Floating-Point-Register Validity	11-20
Invalid CBC in Registers	11-8	General-Register Validity	11-20
Check-Stop State	11-9	Control-Register Validity	11-20
System Check Stop	11-10	Storage Logical Validity	11-20
Machine-Check Interruption	11-10	Access-Register Validity	11-20
Exigent Conditions	11-10	TOD-Programmable-Register Validity ..	11-20
Repressible Conditions	11-10	Floating-Point-Control-Register Validity ..	11-20
Interruption Action	11-11	CPU-Timer Validity	11-20
Point of Interruption	11-13	Clock-Comparator Validity	11-20
Machine-Check-Interruption Code	11-13	Machine-Check Extended Interruption	
Subclass	11-14	Information	11-21
System Damage	11-14	Register-Save Areas	11-21
Instruction-Processing Damage	11-14	External-Damage Code	11-21
System Recovery	11-14	Failing-Storage Address	11-22
Timing-Facility Damage	11-15	Handling of Machine-Check Conditions	11-22
External Damage	11-15	Floating Interruption Conditions	11-22
Degradation	11-15	Floating Machine-Check-Interruption	
Warning	11-16	Conditions	11-23
Channel Report Pending	11-16	Floating I/O Interruptions	11-23
Service-Processor Damage	11-16	Machine-Check Masking	11-23
Channel-Subsystem Damage	11-16	Channel-Report-Pending Subclass Mask ..	11-23
Subclass Modifiers	11-16	Recovery Subclass Mask	11-24
Backed Up	11-16	Degradation Subclass Mask	11-24
Delayed Access Exception	11-16	External-Damage Subclass Mask	11-24
Ancillary Report	11-17	Warning Subclass Mask	11-24
Synchronous Machine-Check-Interruption		Machine-Check Logout	11-24
Conditions	11-17	Summary of Machine-Check Masking	11-24
Processing Backup	11-17		

The machine-check-handling mechanism provides extensive equipment-malfunction detection to ensure the integrity of system operation and to permit automatic recovery from some malfunctions. Equipment

malfunctions and certain external disturbances are reported by means of a machine-check interruption to assist in program-damage assessment and recovery. The interruption supplies the program with infor-

mation about the extent of the damage and the location and nature of the cause. Equipment malfunctions, errors, and other situations which can cause machine-check interruptions are referred to as machine checks.

Machine-Check Detection

Machine-check-detection mechanisms may take many forms, especially in control functions for arithmetic and logical processing, addressing, sequencing, and execution. For program-addressable information, detection is normally accomplished by encoding redundancy into the information in such a manner that most failures in the retention or transmission of the information result in an invalid code. The encoding normally takes the form of one or more redundant bits, called check bits, appended to a group of data bits. Such a group of data bits and the associated check bits are called a checking block. The size of the checking block depends on the model.

The inclusion of a single check bit in the checking block allows the detection of any single-bit failure within the checking block. In this arrangement, the check bit is sometimes referred to as a “parity bit.” In other arrangements, a group of check bits is included to permit detection of multiple errors, to permit error correction, or both.

For checking purposes, the contents of the entire checking block, including the redundancy, are called the checking-block code (CBC). When a CBC completely meets the checking requirements (that is, no failure is detected), it is said to be valid. When both detection and correction are provided and a CBC is not valid but satisfies the checking requirements for correction (the failure is correctable), it is said to be near-valid. When a CBC does not satisfy the checking requirements (the failure is uncorrectable), it is said to be invalid.

Correction of Machine Malfunctions

Four mechanisms may be used to provide recovery from machine-detected malfunctions: error checking

and correction, CPU retry, channel-subsystem recovery, and unit deletion.

Machine failures which are corrected successfully may or may not be reported as machine-check interruptions. If reported, they are system-recovery conditions, which permit the program to note the cause of CPU delay and to keep a log of such incidents.

Error Checking and Correction

When sufficient redundancy is included in circuitry or in a checking block, failures can be corrected. For example, circuitry can be triplicated, with a voting circuit to determine the correct value by selecting two matching results out of three, thus correcting a single failure. An arrangement for correction of failures of one order and for detection of failures of a higher order is called error checking and correction (ECC). Commonly, ECC allows correction of single-bit failures and detection of double-bit failures.

Depending on the model and the portion of the machine in which ECC is applied, correction may be reported as system recovery, or no report may be given.

Uncorrected errors in storage and in the storage key may be reported, along with a failing-storage address, to indicate where the error occurred. Depending on the situation, these errors may be reported along with system recovery or with the damage or backup condition resulting from the error.

CPU Retry

In some models, information about some portion of the state of the machine is saved periodically. The point in the processing at which this information is saved is called a checkpoint. The information saved is referred to as the checkpoint information. The action of saving the information is referred to as establishing a checkpoint. The action of discarding previously saved information is called invalidation of the checkpoint information. The length of the interval between establishing checkpoints is model-dependent. Checkpoints may be established at the beginning of each instruction or several times within a single instruction, or checkpoints may be established less frequently.

Subsequently, this saved information may be used to restore the machine to the state that existed at the time when the checkpoint was established. After restoring the appropriate portion of the machine state, processing continues from the checkpoint. The process of restoring to a checkpoint and then continuing is called CPU retry.

CPU retry may be used for machine-check recovery, to effect nullification and suppression of instruction execution when certain program interruptions occur, and in other model-dependent situations.

Effects of CPU Retry

CPU retry is, in general, performed so that there is no effect on the program. However, change bits which have been changed from zeros to ones are not necessarily set back to zeros. As a result, change bits may appear to be set to ones for blocks which would have been accessed if restoring to the checkpoint had not occurred. If the path taken by the program is dependent on information that may be changed by another CPU or by a channel program or if an interruption occurs, then the final path taken by the program may be different from the earlier path; therefore, change bits may be ones because of stores along a path apparently never taken.

Checkpoint Synchronization

Checkpoint synchronization consists in the following steps.

1. The CPU operation is delayed until all conceptually previous accesses by this CPU to storage have been completed, both for purposes of machine-check detection and as observed by other CPUs and by channel programs.
2. All previous checkpoints, if any, are invalidated.
3. Optionally, a new checkpoint is established.

The CPU operation is delayed until all of these actions appear to be completed, as observed by other CPUs and by channel programs.

Handling of Machine Checks during Checkpoint Synchronization

When, in the process of completing all previous stores as part of the checkpoint-synchronization action, the machine is unable to complete all stores successfully but can successfully restore the machine to a previous checkpoint, processing backup is reported.

When, in the process of completing all stores as part of the checkpoint-synchronization action, the machine is unable to complete all stores successfully and cannot successfully restore the machine to a previous checkpoint, the type of machine-check-interruption condition reported depends on the origin of the store. Failure to successfully complete stores associated with instruction execution may be reported as instruction-processing damage, or some less critical machine-check-interruption condition may be reported with the storage-logical-validity bit set to zero. A failure to successfully complete stores associated with the execution of an interruption, other than program or supervisor call, is reported as system damage.

When the machine check occurs as part of a checkpoint-synchronization action before the execution of an instruction, the execution of the instruction is nullified. When it occurs before the execution of an interruption, the interruption condition, if the interruption is external, I/O, or restart, is held pending. If the checkpoint-synchronization operation was a machine-check interruption, then along with the originating condition, either the storage-logical-validity bit is set to zero or instruction-processing damage is also reported. Program interruptions, if any, are lost.

Checkpoint-Synchronization Operations

All interruptions and the execution of certain instructions cause a checkpoint-synchronization action to be performed. The operations which cause a checkpoint-synchronization action are called checkpoint-synchronization operations and include:

- CPU reset
- All interruptions: external, I/O, machine check, program, restart, and supervisor call
- The BRANCH ON CONDITION (BCR) instruction with the M_1 and R_2 fields containing all ones and all zeros, respectively
- The instructions LOAD PSW, LOAD PSW EXTENDED, SET STORAGE KEY EXTENDED, and SUPERVISOR CALL
- All I/O instructions
- The instructions MOVE TO PRIMARY, MOVE TO SECONDARY, PROGRAM CALL, PROGRAM TRANSFER, SET ADDRESS SPACE CONTROL, and SET SECONDARY ASN, and PROGRAM RETURN when the state entry to be unstacked is a program-call state entry
- The four trace functions: branch tracing, ASN tracing, mode tracing, and explicit tracing, except that when the store-clock-fast facility is installed

and the TRACE TOD-clock control in bit 32 of control register 0 is one, then it is unpredictable whether explicit tracing causes a checkpoint-synchronization action.

- PAGE IN and PAGE OUT

Programming Note: The instructions which are defined to cause the checkpoint-synchronization action invalidate checkpoint information but do not necessarily establish a new checkpoint. Additionally, the CPU may establish a checkpoint between any two instructions or units of operation, or within a single unit of operation. Thus, the point of interruption for the machine check is not necessarily at an instruction defined to cause a checkpoint-synchronization action.

Checkpoint-Synchronization Action

For all interruptions except I/O interruptions, a checkpoint-synchronization action is performed at the completion of the interruption. For I/O interruptions, a checkpoint-synchronization action may or may not be performed at the completion of the interruption. For all interruptions except program, supervisor-call, and exigent machine-check interruptions, a checkpoint-synchronization action is also performed before the interruption. The fetch access to the new PSW may be performed either before or after the first checkpoint-synchronization action. The store accesses and the changing of the current PSW associated with the interruption are performed after the first checkpoint-synchronization action and before the second.

For all checkpoint-synchronization instructions except BRANCH ON CONDITION (BCR), I/O instructions, and SUPERVISOR CALL, checkpoint-synchronization actions are performed before and after the execution of the instruction. For BCR, only one checkpoint-synchronization action is necessarily performed, and it may be performed either before or after the instruction address is updated. For SUPERVISOR CALL, a checkpoint-synchronization action is performed before the instruction is executed, including the updating of the instruction address in the PSW. The checkpoint-synchronization action taken after the supervisor-call interruption is considered to be part of the interruption action and not part of the instruction execution. For I/O instructions, a checkpoint-synchronization action is always performed

before the instruction is executed and may or may not be performed after the instruction is executed.

The four trace functions — branch tracing, ASN tracing, mode tracing, and explicit tracing — cause checkpoint-synchronization actions to be performed before the trace action and after completion of the trace action, except that when the store-clock-fast facility is installed and the TRACE TOD-clock control in bit 32 of control register 0 is one, then it is unpredictable whether checkpoint-synchronization actions are performed for explicit tracing.

Channel-Subsystem Recovery

When errors are detected in the channel subsystem, the channel subsystem attempts to analyze and recover the internal state associated with the various channel-subsystem functions and the state of the channel subsystem and various subchannels. This process, which is called channel-subsystem recovery, may result in a complete recovery or may result in the termination of one or more I/O operations and the clearing of the affected subchannels. Special channel-report-pending machine-check-interruption conditions may be generated to indicate to the program the status of the channel-subsystem recovery.

Malfunctions associated with the I/O operations, depending on the severity of the malfunction, may be reported by means of the I/O-interruption mechanism or by means of the channel-report-pending and channel-subsystem-damage machine-check-interruption conditions.

Unit Deletion

In some models, malfunctions in certain units of the system can be circumvented by discontinuing the use of the unit. Examples of cases where unit deletion may occur include the disabling of all or a portion of a cache or of a translation-lookaside buffer (TLB). Unit deletion may be reported as a degradation machine-check-interruption condition.

Handling of Machine Checks

A machine check is caused by a machine malfunction and not by data or instructions. This is ensured during the power-on sequence by initializing the

machine controls to a valid state and by placing valid CBC in the CPU registers, in the storage keys, and in main storage.

Designation of an unavailable component, such as a storage location, subchannel, or I/O device, does not cause a machine-check indication. Instead, such a condition is indicated by the appropriate program or I/O interruption or condition-code setting. In particular, an attempt to access a storage location which is not in the configuration, or which has power off at the storage unit, results in an addressing exception when detected by the CPU and does not generate a machine-check condition, even though the storage location or its associated storage key has invalid CBC. Similarly, if the channel subsystem attempts to access such a location, an I/O-interruption condition indicating program check is generated rather than a machine-check condition.

A machine check is indicated whenever the result of an operation could be affected by information with invalid CBC or when any other malfunction makes it impossible to establish reliably that an operation can be, or has been, performed correctly. When information with invalid CBC is fetched but not used, the condition may or may not be indicated, and the invalid CBC is preserved.

When a machine malfunction is detected, the action taken depends on the model, the nature of the malfunction, and the situation in which the malfunction occurs. Malfunctions affecting operator-facility actions may result in machine checks or may be indicated to the operator. Malfunctions affecting certain other operations such as SIGNAL PROCESSOR may be indicated by means of a condition code or may result in a machine-check-interruption condition.

A malfunction detected as part of an I/O operation may cause a machine-check-interruption condition, an I/O-error condition, or both. I/O-error conditions are indicated by an I/O interruption or by the appropriate condition-code setting during the execution of an I/O instruction. When the machine reports a failing-storage location detected during an I/O operation, both I/O-error and machine-check conditions may be indicated. The I/O-error condition is the primary indication to the program. The machine-check condition is a secondary indication, which is presented as system recovery together with a failing-storage address.

Certain malfunctions detected as part of I/O instructions and I/O operations are reported by means of special machine-check conditions called I/O machine-check conditions. Thus, malfunctions detected as part of an operation which is I/O related may be reported, depending on the error, in any of three ways: I/O-error condition, I/O machine-check condition, or non-I/O machine-check condition. In some cases, the definition requires the error to be reported by only one of these mechanisms; in other cases, any one, or in some cases, more than one, may be indicated.

Programming Note: Although the definition for machine-check conditions is that they are caused by machine malfunctions and not by data and instructions, there are certain unusual situations in which machine-check conditions are caused by events which are not machine malfunctions. Two examples follow:

1. In some cases, the channel-report-pending machine-check-interruption condition indicates a non-error situation. For example, this condition is generated at the completion of the function specified by RESET CHANNEL PATH.
2. Improper use of DIAGNOSE may result in machine-check conditions.

Validation

Machine errors can be generally classified as solid or intermittent, according to the persistence of the malfunction. A persistent machine error is said to be solid, and one that is not persistent is said to be intermittent. In the case of a register or storage location, a third type of error must be considered, called externally generated. An externally generated error is one where no failure exists in the register or storage location but invalid CBC has been introduced into the location by actions external to the location. For example, the value could be affected by a power transient, or an incorrect value may have been introduced when the information was placed at the location.

Invalid CBC is preserved as invalid when information with invalid CBC is fetched or when an attempt is made to update only a portion of the checking block. When an attempt is made to replace the contents of the entire checking block and the block contains invalid CBC, it depends on the operation and the model whether the block remains with invalid CBC or

is replaced. An operation which replaces the contents of a checking block with valid CBC, while ignoring the current contents, is called a validation operation. Validation is used to place a valid CBC in a register or at a location which has an intermittent or externally generated error.

Validating a checking block does not ensure that a valid CBC will be observed the next time the checking block is accessed. If the failure is solid, validation is effective only if the information placed in the checking block is such that the failing bits are set to the value to which they fail. If an attempt is made to set the bits to the state opposite to that in which they fail, then the validation will not be effective. Thus, for a solid failure, validation is only useful to eliminate the error condition, even though the underlying failure remains, thereby reducing the exposure to additional reports. The locations, however, cannot be used, since invalid CBC will result from attempts to store other values at the location. For an intermittent failure, however, validation is useful to restore a valid CBC such that a subsequent partial store into the checking block will be permitted. (A partial store is a store into a checking block without replacing the entire checking block.)

When a checking block consists of multiple bytes in storage, or multiple bits in CPU registers, the invalid CBC can be made valid only when all of the bytes or bits are replaced simultaneously.

For each type of field in the system, certain instructions are defined to validate the field. Depending on the model, additional instructions may also perform validation; or, in some models, a register is automatically validated as part of the machine-check-interruption sequence after the original contents of the register are placed in the appropriate save area.

When an error occurs in a checking block, the original information contained in the checking block should be considered lost even after validation. Automatic register validation leaves the contents unpredictable. Programmed and manual validation of checking blocks causes the contents to be changed explicitly.

Programming Note: The machine-check-interruption handler must assume that the registers require validation. Thus, each register should be loaded, using an instruction defined to validate, before the register is used or stored.

Invalid CBC in Storage

The size of the checking block in storage depends on the model but is never more than 4K bytes.

When invalid CBC is detected in storage, a machine-check condition may occur; depending on the circumstances, the machine-check condition may be system damage, instruction-processing damage, or system recovery. If the invalid CBC is detected as part of the execution of a channel program, the error is reported as an I/O-error condition. When a CCW, indirect-data-address word, modified-indirect-data-address word, or data is prefetched from storage, is found to have invalid CBC, but is not used in the channel program, the condition is normally not reported as an I/O-error condition. The condition may or may not be reported as a machine-check-interruption condition. Invalid CBC detected during accesses to storage for other than CPU-related accesses may be reported as system recovery with storage error uncorrected indicated, since the primary error indication is reported by some other means.

When the storage checking block consists of multiple bytes and contains invalid CBC, special storage-validation procedures are generally necessary to restore or place new information in the checking block. Validation of storage is provided with the manual load-clear and system-reset-clear operations and is also provided as a program function. Programmed storage validation is done a block at a time, by executing the privileged instruction TEST BLOCK. Manual storage validation by clear reset validates all blocks which are available in the configuration.

A checking block with invalid CBC is never validated unless the entire contents of the checking block are replaced. An attempt to store into a checking block having invalid CBC, without replacing the entire checking block, leaves the data in the checking block (including the check bits) unchanged. Even when an instruction or a channel-program-input operation specifies that the entire contents of a checking block are to be replaced, validation may or may not occur, depending on the operation and the model.

Programming Note: Machine-check conditions may be reported for prefetched and unused data. Depending on the model, such situations may or may not be successfully retried. For example, a BRANCH AND LINK (BALR) instruction which specifies an R₂ field of zero will never branch, but on some models a

prefetch of the location designated by register 0 may occur. Access exceptions associated with this prefetch will not be reported. However, if an invalid checking-block code is detected, CPU retry may be attempted. Depending on the model, the prefetch may recur as part of the retry, and thus the retry will not be successful. Even when the CPU retry is successful, the performance degradation of such a retry is significant, and system recovery may be presented, normally with a failing-storage address. To avoid continued degradation, the program should initiate proceedings to eliminate use of the location and to validate the location.

Programmed Validation of Storage

Provided that an invalid CBC does not exist in the storage key associated with a 4K-byte block, the instruction TEST BLOCK causes the entire 4K-byte block to be set to zeros with a valid CBC, regardless of the current contents of the storage. TEST BLOCK thus removes an invalid CBC from a location in storage which has an intermittent, or one-time, failure. However, if a permanent failure exists in a portion of the storage, a subsequent fetch may find an invalid CBC.

Invalid CBC in Storage Keys

Depending on the model, each storage key may be contained in a single checking block, or the access-control and fetch-protection bits and the reference and change bits may be in separate checking blocks.

Figure 11-1 on page 11-7 describes the action taken when the storage key has invalid CBC. The figure indicates the action taken for the case when the access-control and fetch-protection bits are in one checking block and the reference and change bits are in a separate checking block. In machines where both fields are included in a single checking block, the action taken is the combination of the actions for each field in error, except that completion is permitted only if an error in all affected fields permits completion. References to main storage to which key-controlled protection does not apply are treated as if an access key of zero is used for the reference. This includes such references as channel-program references during initial program loading and implicit references, such as interruption action and DAT-table accesses.

Type of Reference	Action Taken on Invalid CBC	
	For Access-Control and Fetch-Protection Bits	For Reference and Change Bits
SET STORAGE KEY EXTENDED	Complete; validate.	Complete; validate.
INSERT STORAGE KEY EXTENDED	PD; preserve.	PD; preserve.
RESET REFERENCE BIT EXTENDED	PD or complete; preserve.	PD; preserve.
INSERT VIRTUAL STORAGE KEY or TEST PROTECTION	PD; preserve.	CPF; preserve.
CPU prefetch (information not used)	CPF; preserve.	CPF; preserve.
Channel-program prefetch (information not used)	IPF; preserve.	IPF; preserve.
Fetch, nonzero access key	MC; preserve.	MC or complete; preserve.
Store ¹ , nonzero access key	MC ² ; preserve.	MC and preserve; or complete ³ and correct.
Fetch, zero access key ⁴	MC or complete; preserve.	MC or complete; preserve.
Store ¹ , zero access key ²	MC or complete; preserve.	MC and preserve; or complete ³ and correct.

Figure 11-1. Invalid CBC in Storage Keys (Part 1 of 2)

Explanation:	
1	CPU virtual- and logical-address store accesses are subject to DAT protection. When the DAT-protection bit is one, the location will not be changed; however, the machine may indicate a machine-check condition if the storage key or the data itself has invalid CBC.
2	The contents of the main-storage location are not changed.
3	The contents of the reference and change bits are set to ones if the "complete" action is taken.
4	The action shown for an access key of zero is also applicable to references to which key-controlled protection does not apply.
Complete	The condition does not cause termination of the execution of the instruction, and, unless an unrelated condition prohibits it, the execution of the instruction is completed, ignoring the error condition. No machine-check-damage conditions are reported, but system recovery may be reported.
Correct	The reference and change bits are set to ones with valid CBC.
Preserve	The contents of the entire checking block having invalid CBC are left unchanged.
Validate	The entire key is set to the new value with valid CBC. When the conditional-SSKE facility is installed and either or both the MR and MC bits of the M_3 field are one, condition code 3 is set when the instruction completes.
CPF	Invalid CBC in the storage key for a CPU prefetch which is unused, or for instructions which do not examine the reference and change bits, may result in any of the following situations: <ul style="list-style-type: none"> • The operation is completed; no machine-check condition is reported. • The operation is completed; system recovery, with storage-key error uncorrected, is reported. • Instruction-processing damage, with or without backup and with storage-key error uncorrected, is reported.
IPF	Invalid CBC in the storage key for a channel-program prefetch which is unused may result in any of the following: <ul style="list-style-type: none"> • The I/O operation is completed; no machine-check condition is reported. • The I/O operation is completed; system recovery, with storage-key error uncorrected, is reported.
MC	Same as PD for CPU references, but a channel-subsystem reference may result in the following combinations of I/O-error conditions and machine-check conditions: <ul style="list-style-type: none"> • An I/O-error condition is reported; no machine-check condition is reported. • An I/O-error condition is reported; system recovery, with or without storage-key error uncorrected, is reported.
PD	Instruction-processing damage, with or without backup and with or without storage-key error uncorrected, is reported.
Note:	When storage-key error uncorrected is reported, a failing storage address may or may not also be reported.

Figure 11-1. Invalid CBC in Storage Keys (Part 2 of 2)

Invalid CBC in Registers

When invalid CBC is detected in a CPU register, a machine-check condition may be recognized. CPU registers include the general, floating-point, floating-point-control, access, control, and TOD programmable registers, the current PSW, the prefix register, the TOD clock, the CPU timer, and the clock comparator.

When a machine-check interruption occurs, whether or not it is due to invalid CBC in a CPU register, the following actions affecting the CPU registers, other than the prefix register and the TOD clock, are taken as part of the interruption.

1. The contents of the registers are saved in assigned storage locations. Any register which is in error is identified by a corresponding validity bit of zero in the machine-check-interruption code. Malfunctions detected during register saving do not result in additional machine-check-interruption conditions; instead, the correctness

of all the information stored is indicated by the appropriate setting of the validity bits.

2. On some models, registers with invalid CBC are then validated, their actual contents being unpredictable. On other models, programmed validation is required.

The prefix register and the TOD clock are not stored during a machine-check interruption, have no corresponding validity bit, and are not validated.

On those models in which registers are not automatically validated as part of the machine-check interruption, a register with invalid CBC will not cause a machine-check-interruption condition unless the contents of the register are actually used. In these models, each register may consist of one or more checking blocks, but multiple registers are not included in a single checking block. When only a portion of a register is accessed, invalid CBC in the unused portion of the same register may cause a

machine-check-interruption condition. For example, invalid CBC in the right half of a floating-point register may cause a machine-check-interruption condition if a LOAD (LE) operation attempts to replace the left half, or short form, of the register.

Invalid CBC associated with the prefix register cannot safely be reported by the machine-check interruption, since the interruption itself requires that the prefix value be applied to convert real addresses to the corresponding absolute addresses. Invalid CBC in the prefix register causes the CPU to enter the check-stop state immediately.

On those models which do not validate registers during a machine-check interruption, the following instructions will cause validation of a register, provided the information in the register is not used before the register is validated. Other instructions, although they replace the entire contents of a register, do not necessarily cause validation.

General registers are validated by BRANCH AND SAVE and LOAD ADDRESS executed in the 64-bit addressing mode, by LOAD (LGR), and by LOAD (LG) and LOAD MULTIPLE (LMG) if the operand is on a doubleword boundary.

Floating-point registers are validated by LOAD (LDR) and, if the operand is on a doubleword boundary, by LOAD (LD).

The floating-point-control register is validated by LOAD FLOATING POINT CONTROL REGISTER.

Access registers are validated by LOAD ACCESS MULTIPLE. Only the even-odd access-register pairs that are included in the set of access registers specified for LOAD ACCESS MULTIPLE are validated. Thus, when a single access register is specified, or when a pair of access registers starting with an odd-numbered register is specified, no register is validated.

Control registers may be validated either singly or in groups by using the instruction LOAD CONTROL (LCTLG).

The TOD programmable register, CPU timer, clock comparator, and prefix register are validated by SET CLOCK PROGRAMMABLE REGISTER, SET CPU TIMER, SET CLOCK COMPARATOR, and SET PREFIX, respectively.

The TOD clock is validated by a SET CLOCK instruction that sets the clock.

Programming Note: Depending on the register and the model, the contents of a register may be validated by the machine-check interruption, or the model may require that a program execute a validating instruction after the machine-check interruption has occurred. In the case of the CPU timer, depending on the model, both the machine-check interruption and validating instructions may be required to restore the CPU timer to full working order.

Check-Stop State

In certain situations, it is impossible or undesirable to continue operation when a machine error occurs. In these cases, the CPU may enter the check-stop state, which is indicated by the check-stop indicator.

In general, the CPU may enter the check-stop state whenever an uncorrectable error or other malfunction occurs and the machine is unable to recognize a specific machine-check-interruption condition.

The CPU always enters the check-stop state if any of the following conditions exists:

- PSW bit 13 is zero, and an exigent machine-check condition is generated.
- During the execution of an interruption due to one exigent machine-check condition, another exigent machine-check condition is detected.
- During a machine-check interruption, the machine-check-interruption code cannot be stored successfully, or the new PSW cannot be fetched successfully.
- Invalid CBC is detected in the prefix register.
- A malfunction in the receiving CPU, which is detected after accepting the order, prevents the successful completion of a SIGNAL PROCESSOR order and the order was a reset, or the receiving CPU cannot determine what the order was. The receiving CPU enters the check-stop state.

There may be many other conditions for particular models when an error may cause check stop.

When the CPU is in the check-stop state, instructions and interruptions are not executed. The TOD clock is normally not affected by the check-stop state. The CPU timer may or may not run in the check-stop state, depending on the error and the model. The start key and stop key are not effective in this state.

The CPU may be removed from the check-stop state by CPU reset.

In a multiprocessing configuration, a CPU entering the check-stop state generates a request for a malfunction-alert external interruption to all CPUs in the configuration. Except for the reception of a malfunction alert, other CPUs and the I/O system are normally unaffected by the check-stop state in a CPU. However, depending on the nature of the condition causing the check stop, other CPUs may also be delayed or stopped, and channel subsystem and I/O activity may be affected.

System Check Stop

In a multiprocessing configuration, some errors, malfunctions, and damage conditions are of such severity that the condition causes all CPUs in the configuration to enter the check-stop state. This condition is called a system check stop. The state of the channel subsystem and I/O activity is unpredictable.

Machine-Check Interruption

A request for a machine-check interruption, which is made pending as the result of a machine check, is called a machine-check-interruption condition. There are two types of machine-check-interruption conditions: exigent conditions and repressible conditions.

Exigent Conditions

Exigent machine-check-interruption conditions are those in which damage has or would have occurred such that execution of the current instruction or interruption sequence cannot safely continue. Exigent conditions include two subclasses: instruction-processing damage and system damage. In addition to indicating specific exigent conditions, system damage is used to report any malfunction or error which cannot be isolated to a less severe report.

Exigent conditions for instruction sequences can be either nullifying exigent conditions or terminating exigent conditions, according to whether the instructions affected are nullified or terminated. Exigent conditions for interruption sequences are terminating exigent conditions. The terms “nullification” and “termination” have the same meanings as those used in Chapter 5, “Program Execution”, except that more than one instruction may be involved. Thus, a nullifying exigent condition indicates that the CPU has returned to the beginning of a unit of operation prior to the error. A terminating exigent condition means that the results of one or more instructions may have unpredictable values.

Repressible Conditions

Repressible machine-check-interruption conditions are those in which the results of the instruction-processing sequence have not been affected. Repressible conditions can be delayed, until the completion of the current instruction or even longer, without affecting the integrity of CPU operation. Repressible conditions are of three groups: recovery, alert, and repressible damage. Each group includes one or more subclasses.

A malfunction in the CPU, storage, or operator facilities which has been successfully corrected or circumvented internally without logical damage is called a recovery condition. Depending on the model and the type of malfunction, some or all recovery conditions may be discarded and not reported. Recovery conditions that are reported are grouped in one subclass, system recovery.

A machine-check-interruption condition not directly related to a machine malfunction is called an alert condition. The alert conditions are grouped in two subclasses: degradation and warning.

A malfunction resulting in an incorrect state of a portion of the system not directly affecting sequential CPU operation is called a repressible-damage condition. Repressible-damage conditions are grouped in five subclasses, according to the function affected: timing-facility damage, external damage, channel report pending, channel-subsystem damage, and service-processor damage.

Programming Notes:

1. Even though repressible conditions are usually reported only at normal points of interruption, they may also be reported with exigent machine-check conditions. Thus, if an exigent machine-check condition causes an instruction to be abnormally terminated and a machine-check interruption occurs to report the exigent condition, any pending repressible conditions may also be reported. The meaningfulness of the validity bits depends on what exigent condition is reported.
2. Classification of damage as either exigent or repressible does not imply the severity of the damage. The distinction is whether action must be taken as soon as the damage is detected (exigent) or whether the CPU can continue processing (repressible). For a repressible condition, the current instruction can be completed before taking the machine-check interruption if the CPU is enabled for machine checks; if the CPU is disabled for machine checks, the condition can safely be kept pending until the CPU is again enabled for machine checks.

For example, the CPU may be disabled for machine-check interruptions because it is handling an earlier instruction-processing-damage interruption. If, during that time, an I/O operation encounters a storage error, that condition can be kept pending because it is not expected to interfere with the current machine-check processing. If, however, the CPU also makes a reference to the area of storage containing the error before re-enabling machine-check interruptions, another instruction-processing-damage condition is created, which is treated as an exigent condition and causes the CPU to enter the check-stop state.

3. A repressible condition may be a floating condition. A floating repressible condition is eligible to cause an interruption on any CPU in the configuration. At the point when a CPU performs an interruption for a floating repressible condition, the condition is no longer eligible to cause an interruption on the remaining CPUs in the configuration.

Interruption Action

A machine-check interruption causes the following actions to be taken. An architectural-mode identification with the value 01 hex is stored at real location 163. The PSW reflecting the point of interruption is stored as the machine-check old PSW in the quadword at real location 352. The contents of other registers are stored in register-save areas at real locations 4608-4863, 4892-4895, 4900-4911, 4913-4919, and 4928-5119. After the contents of the registers are stored in register-save areas, depending on the model, the registers may be validated with the contents being unpredictable. A machine-check-interruption code (MCIC) of eight bytes is stored at real locations 232-239. An external-damage code may be stored at real locations 244-247, and a failing-storage address may be stored at real locations 248-255. The new PSW is fetched from real locations 480-495. In addition, a machine-check logout may have occurred.

The machine-generated addresses used to access the old and new PSW, the MCIC, extended interruption information, and the fixed-logout area are all real addresses.

The fields in assigned storage locations that are accessed during the machine-check interruption are summarized in Figure 11-2.

Information Stored (Fetched)	Starting Location*	Length in Bytes
Architectural-mode identification	163	1
Old PSW	352	16
New PSW (fetched)	480	16
Machine-check-interruption code	232	8
Register-save areas		
Floating-point registers 0-15	4608	128
General registers 0-15	4736	128
Floating-point control register	4892	4
TOD programmable register	4900	4
CPU timer	4904	8
Clock comparator	4913	7
Access registers 0-15	4928	64
Control registers 0-15	4992	128
Extended interruption information		

Figure 11-2. Machine-Check-Interruption Locations

Information Stored (Fetched)	Starting Location*	Length in Bytes
External-damage code	244	4
Failing-storage address	248	8
Fixed-logout area	4864	16
Explanation:		
* All locations are in real storage.		

Figure 11-2. Machine-Check-Interrupt Locations

If the machine-check-interruption code cannot be stored successfully or the new PSW cannot be fetched successfully, the CPU enters the check-stop state.

A repressible machine-check condition can initiate a machine-check interruption only if both PSW bit 13 is one and the associated subclass mask bit, if any, in control register 14 is also one. When it occurs, the interruption does not terminate the execution of the current instruction; the interruption is taken at a normal point of interruption, and no program or supervisor-call interruptions are eliminated. If the machine check occurs during the execution of a machine function, such as a CPU-timer update, the machine-check interruption takes place after the machine function has been completed.

When the CPU is disabled for a particular repressible machine-check condition, the condition remains pending. Depending on the model and the condition, multiple repressible conditions may be held pending for a particular subclass, or only one condition may be held pending for a particular subclass, regardless of the number of conditions that may have been detected for that subclass.

When a repressible machine-check interruption occurs because the interruption condition is in a subclass for which the CPU is enabled, pending conditions in other subclasses may also be indicated by the same interruption code, even though the CPU is disabled for those subclasses. All indicated conditions are then cleared.

If a machine check which is to be reported as a system-recovery condition is detected during the execution of the interruption procedure due to a previous machine-check condition, the system-recovery condition may be combined with the other conditions, discarded, or held pending.

An exigent machine-check condition can cause a machine-check interruption only when PSW bit 13 is one. When a nullifying exigent condition causes a machine-check interruption, the interruption is taken at a normal point of interruption. When a terminating exigent condition causes a machine-check interruption, the interruption terminates the execution of the current instruction and may eliminate the program and supervisor-call interruptions, if any, that would have occurred if execution had continued. Proper execution of the interruption sequence, including the storing of the old PSW and other information, depends on the nature of the malfunction. When an exigent machine-check condition occurs during the execution of a machine function, such as a CPU-timer update, the sequence is not necessarily completed.

If, during the execution of an interruption due to one exigent machine-check condition, another exigent machine check is detected, the CPU enters the check-stop state. If an exigent machine check is detected during an interruption due to a repressible machine-check condition, system damage is reported.

When PSW bit 13 is zero, an exigent machine-check condition causes the CPU to enter the check-stop state.

Machine-check-interruption conditions are handled in the same manner regardless of whether the wait-state bit in the PSW is one or zero: a machine-check condition causes an interruption if the CPU is enabled for that condition.

Machine checks which occur while the rate control is set to the instruction-step position are handled in the same manner as when the control is set to the process position; that is, recovery mechanisms are active, and machine-check interruptions occur when allowed. Machine checks occurring during a manual operation may be indicated to the operator, may generate a system-recovery condition, may result in system damage, or may cause a check stop, depending on the model.

Every reasonable attempt is made to limit the side effects of any machine check and the associated interruption. Normally, interruptions, as well as the progress of I/O operations, remain unaffected. The malfunction, however, may affect these activities, and, if the currently active PSW has bit 13 set to one, the machine-check interruption will indicate the total

extent of the damage caused, and not just the damage which originated the condition.

Point of Interruption

The point in the processing which is indicated by the interruption and used as a reference point by the machine to determine and indicate the validity of the status stored is referred to as the point of interruption.

Because of the checkpoint capability in models with CPU retry, the interruption resulting from an exigent machine-check-interruption condition may indicate a point in the CPU processing sequence which is logically prior to the error. Additionally, the model may have some choice as to which point in the CPU processing sequence the interruption is indicated, and, in some cases, the status which can be indicated as valid depends on the point chosen.

Only certain points in the processing may be used as a point of interruption. For repressible machine-check interruptions, the point of interruption must be after one unit of operation is completed and any associated program or supervisor-call interruption is taken, and before the next unit of operation is begun.

Exigent machine-check conditions for instruction sequences are those in which damage has or would have occurred to the instruction stream. Thus, the damage can normally be associated with a point part way through an instruction, and this point is called the point of damage. In some cases, there may be one or more instructions separating the point of damage and the point of interruption, and the processing associated with one or more instructions may be damaged. When the point of interruption is a point prior to the point of damage due to a nullifiable exigent machine-check condition, the point of interruption can be only at the same points as for repressible machine-check conditions.

In addition to the point of interruption permitted for repressible machine-check conditions, the point of interruption for a terminating exigent machine-check

condition may also be after the unit of operation is completed but before any associated program or supervisor-call interruption occurs. In this case, a valid PSW instruction address is defined as that which would have been stored in the old PSW for the program or supervisor-call interruption. Since the operation has been terminated, the values in the result fields, other than the instruction address, are unpredictable. Thus, the validity bits associated with fields which are due to be changed by the instruction stream are meaningless when a terminating exigent machine-check condition is reported.

When the point of interruption and the point of damage due to an exigent machine-check condition are separated by a checkpoint-synchronization function, the damage has not been isolated to a particular program, and system damage is indicated.

When an exigent machine-check-interruption condition occurs, the point of interruption which is chosen affects the amount of damage which must be indicated. An attempt is made, when possible, to choose a point of interruption which permits the minimum indication of damage. In general, the preference is the interruption point immediately preceding the error.

When all the status information stored as a result of an exigent machine-check-interruption condition does not reflect the same point, an attempt is made, when possible, to choose the point of interruption so that the instruction address which is stored in the machine-check old PSW is valid.

Machine-Check-Interruption Code

On all machine-check interruptions, a machine-check-interruption code (MCIC) is stored in the doubleword starting at real location 232. The code has the format shown in Figure 11-3.

Bits in the MCIC which are not assigned or not implemented by a particular model are stored as zeros.

Programming Notes:

1. System recovery may be used to report a failing-storage address detected by a CPU prefetch or by an I/O operation.
2. Unless the corresponding validity bits are ones, the indication of system recovery does not imply storage logical validity or that the fields stored as a result of the machine-check interruption are valid.

Timing-Facility Damage

Bit 4 (CD), when one, indicates that damage has occurred to the TOD clock, CPU timer, clock comparator, or TOD programmable register, or to the CPU-timer or clock-comparator external-interruption conditions. The timing-facility-damage machine-check condition is set whenever any of the following occurs:

1. The TOD clock enters the error or not-operational state.
2. The CPU timer is damaged, and the CPU is enabled for CPU-timer external interruptions. On some models, this condition may be recognized even when the CPU is not enabled for CPU-timer interruptions. Depending on the model, the machine-check condition may be generated only as the CPU timer enters an error state. Or, the machine-check condition may be continuously generated whenever the CPU is enabled for CPU-timer interruptions, until the CPU timer is validated.
3. The clock comparator is damaged, and the CPU is enabled for clock-comparator external interruptions. On some models, this condition may be recognized even when the CPU is not enabled for clock-comparator interruptions.

Timing-facility damage may also be set along with instruction-processing damage when an instruction which accesses the TOD clock, CPU timer, or clock comparator produces incorrect results. Depending on the model, the TOD programmable register, CPU timer, or clock comparator may be validated by the interruption which reports the TOD programmable register, CPU timer, or clock comparator as invalid.

Timing-facility damage is a repressible condition. It is masked by the external-damage subclass-mask bit, which is in bit position 38 of control register 14.

Timing-facility-damage conditions for the CPU timer and the clock comparator are not recognized on most models when these facilities are not in use. The facilities are considered not in use when the CPU is disabled for the corresponding external interruptions (PSW bit 7, or the subclass-mask bits, bits 52 and 53 of control register 0, are zeros), and when the corresponding set and store instructions are not executed. Timing-facility-damage conditions that are already pending remain pending, however, when the CPU is disabled for the corresponding external interruption.

Timing-facility-damage conditions due to damage to the TOD clock are always recognized.

With TOD-clock steering, damage to the TOD clock is considered to be of such severity that it is undesirable to continue operation. When the TOD-clock-steering facility is installed, and damage has occurred to the TOD-clock-steering registers or the TOD clock and the damage is limited to a single CPU, then that CPU enters the check-stop state. If the TOD clock is in the not-operational state or error state, or when damage has occurred to the TOD-clock-steering registers or the TOD clock, and the damage affects all CPUs, then a system check stop occurs.

External Damage

Bit 5 (ED), when one, indicates that damage has occurred during operations not directly associated with processing the current instruction.

When bit 5, external damage, is one and bit 26, external-damage-code validity, is also one, the external-damage code has been stored to indicate, in more detail, the cause of the external-damage machine-check interruption. When the external damage cannot be isolated to one or more of the conditions as defined in the external-damage code, or when the detailed indication for the condition is not implemented by the model, external damage is indicated with bit 26 set to zero. The presence and extent of reporting external damage depend on the model.

External damage is a repressible condition. It is masked by the external-damage subclass-mask bit, which is in bit position 38 of control register 14.

Degradation

Bit 7 (DG), when one, indicates that continuous degradation of system performance, more serious than

that indicated by system recovery, has occurred. Degradation may be reported when system-recovery conditions exceed a machine-preestablished threshold or when unit deletion has occurred. The presence and extent of the degradation-report capability depend on the model.

Degradation is a repressible condition. It is masked by the degradation subclass-mask bit, which is in bit position 37 of control register 14.

Warning

Bit 8 (W), when one, indicates that damage is imminent in some part of the system (for example, that power is about to fail, or that a loss of cooling is occurring). Whether warning conditions are recognized depends on the model.

If the condition responsible for the imminent damage is removed before the interruption request is honored (for example, if power is restored), the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from the same condition.

Warning is a repressible condition. It is masked by the warning subclass-mask bit, which is in bit position 39 of control register 14.

Channel Report Pending

Bit 9 (CP), when one, indicates that a channel report, consisting of one or more channel-report words, has been made pending, and the contents of the channel-report words describe, in further detail, the effect of the malfunction and the results of analysis or the action performed. A channel report becomes pending when one of the following conditions has occurred:

1. Channel-subsystem recovery has been completed. The channel-subsystem recovery may have been initiated with no prior notice to the program or may have been a result of a condition previously reported to the program.
2. The function specified by RESET CHANNEL PATH has been completed.

A channel report may also become pending under other conditions.

The channel-report words which make up the channel report may be cleared, one at a time, by execu-

tion of the instruction STORE CHANNEL REPORT WORD, which is described in "I/O Instructions" on page 14-1.

Bit 9 is meaningless when channel-subsystem damage is reported.

Channel report pending is a floating repressible condition. It is masked by the channel-report-pending subclass-mask bit, which is in bit position 35 of control register 14.

Service-Processor Damage

Bit 10 (SP), when one, indicates that damage has occurred to the service processor. Service-processor damage may be made pending at all CPUs in the configuration, or it may be detected independently by each CPU. The presence and extent of reporting service-processor damage depend on the model.

Service-processor damage is a repressible condition and has no subclass-mask bit.

Channel-Subsystem Damage

Bit 11 (CK), when one, indicates that an error or malfunction has occurred in the channel subsystem, or that the channel subsystem is in the check-stop state. The channel subsystem enters the check-stop state when a malfunction occurs which is so severe that the channel subsystem cannot continue, or if power is lost in the channel subsystem.

Channel-subsystem damage is a floating repressible condition and has no subclass-mask bit.

Subclass Modifiers

Bits 14 (B), 34 (DA), and 44 (AP) of the machine-check-interruption code act as modifiers to the subclass bits.

Backed Up

Bit 14 (B), when one, indicates that the point of interruption is at a checkpoint before the point of error. This bit is meaningful only when the instruction-processing-damage bit, bit 1, is also set to one. The presence and extent of the capability to indicate a backed-up condition depend on the model.

Delayed Access Exception

Bit 34 (DA), when one, indicates that an access exception was detected during a storage access

using DAT when no such exception was detected by an earlier test for access exceptions.

Bit 34 is a modifier to instruction-processing damage (bit 1) and is meaningful only when bit 1 of the machine-check-interruption code is one. When bit 1 is zero, bit 34 has no meaning. The presence and extent of reporting delayed access exception depend on the model.

Programming Note: The occurrence of a delayed access exception normally indicates that the program is using an improper procedure to update the DAT tables.

Ancillary Report

Bit 44 (AP), when one, indicates that a malfunction of a system component has occurred which has been recognized previously or which has affected the activities of multiple system elements such as CPUs and subchannels. When the malfunction affects the activities of multiple elements, an ancillary-report condition is recognized for all of the affected elements except one. This bit, when zero, indicates that this malfunction of a system component has not been recognized previously. This bit is meaningful for all conditions indicated by either the machine-check-interruption code or the external-damage code.

Depending on the model, recognition of an ancillary-report condition may not be provided, or it may not be provided for all system malfunctions. When ancillary-report recognition is not provided, bit 44 is set to zero.

Synchronous Machine-Check-Interruption Conditions

The instruction-processing damage and backed-up bits, bits 1 and 14 of the machine-check-interruption code, identify, in combination, two conditions.

Bit 1	Bit 14	Name of Condition
1	0	Processing damage
1	1	Processing backup

Processing Backup

The processing-backup condition indicates that the point of interruption is prior to the point, or points, of error. This is a nullifying exigent condition. When all of the other CPU-related-damage subclasses and modifiers of the machine-check-interruption code are

zero, and certain validity bits associated with CPU status are indicated as valid, then the machine has successfully returned to a checkpoint prior to the malfunction, and no damage has yet occurred to the CPU.

The subclass bits which must be zero for no damage to have occurred are as follows:

MCIC

Bit	Name
0	System damage
4	Timing-facility damage

The delayed-access-exception subclass-modifier bit, MCIC bit 34, must be zero for no damage to have occurred.

The validity bits in the machine-check-interruption code which must be one for no damage to have occurred are as follows:

MCIC

Bit	Name
20	PSW MWP bits
21	PSW mask and key
22	PSW program mask and condition code
23	PSW instruction address
27	Floating-point registers
28	General registers
29	Control registers
31	Storage logical validity (result fields within current checkpoint interval)
33	Access registers
42	TOD programmable register
43	Floating-point-control register
46	CPU timer
47	Clock comparator

Programming Note: The processing-backup condition is reported rather than system recovery to indicate that a malfunction or failure stands in the way of continued operation of the CPU. The malfunction has not been circumvented, and damage would have occurred if instruction processing had continued.

Processing Damage

The processing-damage condition indicates that damage has occurred to the instruction processing of the CPU. The point of interruption is a point beyond some or all of the points of damage. Processing damage is a terminating exigent condition; therefore, the

contents of result fields may be unpredictable and still indicated as valid.

Processing damage may include malfunctions in program-event recording, monitor call, tracing, access-register translation, and dynamic address translation. Processing damage causes any supervisor-call-interruption condition and program-interruption condition to be discarded. However, the contents of the old PSW and interruption-code locations for these interruptions may be set to unpredictable values.

Storage Errors

Bits 16-18 of the machine-check-interruption code are used to indicate an invalid CBC or a near-valid CBC detected in main storage or an invalid CBC in a storage key. Bit 19, storage degradation, may be indicated concurrently with bit 17. The failing-storage-address field, when indicated as valid, identifies a location within the storage checking block containing the error, or, for storage-key error uncorrected, within the block associated with the storage key. Bit 32, indirect storage error, may be set to one to indicate that the location designated by the failing-storage address is not the original source of the error.

The storage-error-uncorrected and storage-key-error-uncorrected bits do not in themselves indicate the occurrence of damage because the error detected may not have affected a result. The portion of the configuration affected by an invalid CBC is indicated in the subclass field of the machine-check-interruption code.

Storage errors detected for a channel program, when indicated as I/O-error conditions, may also be reported as system recovery. CBC errors that occur in storage or in the storage key and that are detected on prefetched or unused data for a CPU program may or may not be reported, depending on the model.

Storage Error Uncorrected

Bit 16 (SE), when one, indicates that a checking block in main storage contained invalid CBC and that the information could not be corrected. The contents of the checking block in main storage have not been changed. The location reported may have been accessed or prefetched for this CPU or another CPU or a channel program, or it may have been accessed as the result of a model-dependent storage access.

Storage Error Corrected

Bit 17 (SC), when one, indicates that a checking block in main storage contained near-valid CBC and that the information has been corrected before being used. Depending on the model, the contents of the checking block in main storage may or may not have been restored to valid CBC. The location reported may have been accessed or prefetched for this CPU or for another CPU or for a channel program, or it may have been accessed as the result of a model-dependent storage access. The presence and extent of the storage-error-correction capability depend on the model. This indication may or may not be accompanied by an indication of storage degradation, bit 19 (DS).

Storage-Key Error Uncorrected

Bit 18 (KE), when one, indicates that a storage key contained invalid CBC and that the information could not be corrected. The contents of the checking block in the storage key have not been changed. The storage key may have been accessed or prefetched for this CPU or for another CPU or for a channel program, or it may have been accessed as the result of a model-dependent storage access.

Storage Degradation

Bit 19 (DS), when one, indicates that degradation of the recovery characteristics has occurred for the 4K-byte block reported by the failing-storage address.

Storage degradation indicates that although the associated storage error has been corrected, there are solid failures associated with the storage block (or with its associated key) that cause the correction process to take a substantial amount of time, and that if an additional error occurs in the block, the error may not be correctable or may go undetected. Thus, this bit indicates that use of the indicated block of storage should be avoided, if possible.

The indication of storage degradation has meaning only when failing-storage-address validity, MCIC bit 24, is also one. The presence and extent of reporting storage degradation depend on the model.

Because storage degradation is normally reported with system recovery, the recovery subclass mask, bit 36 of control register 14, should be set to one in order for storage degradation to be indicated.

Indirect Storage Error

Bit 32 (IE), when one, indicates that the physical main-storage location identified by the failing-storage address is not the original source of the error. Instead, the error originated in another level of the storage hierarchy and has been propagated to the current physical-storage portion of the storage hierarchy. Bit 32 is meaningful only when bit 16 or 18 (storage error uncorrected or storage-key error uncorrected) of the machine-check-interruption code is one. When bits 16 and 18 are both zeros, bit 32 has no meaning.

For errors originating outside the storage hierarchy, the attempt to store is rejected, and the appropriate error indication is presented. When an error is detected during implicit movement of information inside the storage hierarchy, the action is not rejected and reported in this manner because the movement may be asynchronous and may be initiated as the result of an attempt to access completely unrelated information. Instead, errors in the contents of the source during implicit moving of information from one portion of the storage hierarchy to another may be preserved in the target area by placing a special invalid CBC in the checking block associated with the target location. These propagated errors, when detected later, are reported as indirect storage errors. The original source of such an error may have been in a cache associated with an I/O processor or a CPU, or the error may have been the result of a data-path failure in transmitting data from one portion of the storage hierarchy to another. Additionally, a propagated error may be generated during the movement of data from one physical portion of storage to another as the result of a storage-reconfiguration action.

The presence and extent of reporting indirect storage error depend on the model.

Programming Note: See the programming notes under TEST BLOCK in Chapter 10, “Control Instructions” for the action which should be taken after storage errors are reported.

Machine-Check Interruption-Code Validity Bits

Bits 20-24, 26-29, 31, 33, 42, 43, 46, and 47 of the machine-check-interruption code are validity bits. Each bit indicates the validity of a particular field in

storage. With the exception of the storage-logical-validity bit (bit 31), each bit is associated with a field stored during the machine-check interruption. When a validity bit is one, it indicates that the saved value placed in the corresponding storage field is valid with respect to the indicated point of interruption and that no error was detected when the data was stored.

When a validity bit is zero, one or more of the following conditions may have occurred: the original information was incorrect, the original information had invalid CBC, additional malfunctions were detected while storing the information, or none or only part of the information was stored. Even though the information is unpredictable, the machine attempts, when possible, to place valid CBC in the storage field and thus reduce the possibility of additional machine checks being caused.

The validity bits for the floating-point registers, general registers, control registers, access registers, TOD programmable register, floating-point control register, CPU timer, and clock comparator indicate the validity of the saved value placed in the corresponding save area. The information in these registers after the machine-check interruption is not necessarily correct even when the correct value has been placed in the save area and the validity bit set to one. The use of the registers and the operation of the facility associated with the control registers, floating-point control register, TOD programmable register, CPU timer, and clock comparator are unpredictable until these registers are validated. (See “Invalid CBC in Registers” on page 8.)

PSW-MWP Validity

Bit 20 (WP), when one, indicates that bits 12-15 of the machine-check old PSW are correct.

PSW Mask and Key Validity

Bit 21 (MS), when one, indicates that the system mask, PSW key, and miscellaneous bits of the machine-check old PSW are correct. Specifically, this bit covers bits 0-11, 16, 17, 24-30, and 33-63 of the PSW.

PSW Program-Mask and Condition-Code Validity

Bit 22 (PM), when one, indicates that the program mask and condition code of the machine-check old PSW are correct.

PSW-Instruction-Address Validity

Bit 23 (IA), when one, indicates that the addressing-mode and instruction-address bits, bits 31, 32, and 64-127, of the machine-check old PSW are correct.

Failing-Storage-Address Validity

Bit 24 (FA), when one, indicates that a correct failing-storage address has been stored at real location 248-255 after a storage-error-uncorrected, storage-key-error-uncorrected, or storage-error-corrected condition has occurred. The presence and extent of the capability to identify the failing-storage location depend on the model. When no such errors are reported, that is, bits 16-18 of the machine-check-interruption code are zeros, the failing-storage address is meaningless, even though it may be indicated as valid.

External-Damage-Code Validity

Bit 26 (EC), when one, and provided that bit 5, external damage, is also one, indicates that a valid external-damage code has been stored in the word at real location 244. When bit 5 is zero, bit 26 has no meaning.

Floating-Point-Register Validity

Bit 27 (FP), when one, indicates that the contents of the floating-point-register save area at real locations 4608-4735 reflect the correct state of the floating-point registers at the point of interruption.

General-Register Validity

Bit 28 (GR), when one, indicates that the contents of the general-register save area at real locations 4736-4863 reflect the correct state of the general registers at the point of interruption.

Control-Register Validity

Bit 29 (CR), when one, indicates that the contents of the control-register save area at real locations 4992-5119 reflect the correct state of the control registers at the point of interruption.

Storage Logical Validity

Bit 31 (ST), when one, indicates that the storage locations, the contents of which are modified by the instructions being executed, contain the correct information relative to the point of interruption. That is, all stores before the point of interruption are completed, and all stores, if any, after the point of interruption are suppressed. When a store before the point of interruption is suppressed because of an invalid CBC, the

storage-logical-validity bit may be indicated as one, provided that the invalid CBC has been preserved as invalid.

When instruction-processing damage is indicated but processing backup is not indicated, the storage-logical-validity bit has no meaning.

Storage logical validity reflects only the instruction-processing activity and does not reflect errors in the state of storage as the result of either I/O operations or the storing of the old PSW and other interruption information.

Access-Register Validity

Bit 33 (AR), when one, indicates that the contents of the access-register save area at real locations 4928-4991 reflect the correct state of the access registers at the point of interruption.

TOD-Programmable-Register Validity

Bit 42 (PR), when one, indicates that the contents of the TOD-programmable-register save area at real locations 4900-4903 reflect the correct state of the TOD programmable register at the point of interruption.

Floating-Point-Control-Register Validity

Bit 43 (FC), when one, indicates that the contents of the floating-point-control-register save area at real locations 4892-4895 reflect the correct state of the floating-point-control register at the point of interruption.

CPU-Timer Validity

Bit 46 (CT), when one, indicates that the CPU timer is not in error and that the contents of the CPU-timer save area at real locations 4904-4911 reflect the correct state of the CPU timer at the time the interruption occurred.

Clock-Comparator Validity

Bit 47 (CC), when one, indicates that the clock comparator is not in error, that the contents of the clock-comparator save area at real locations 4913-4919 reflect the correct state of the clock comparator at the time the interruption occurred, and that zeros have been stored at real location 4912.

Programming Note: The validity bits must be used in conjunction with the subclass bits and the backed-up bit in order to determine the extent of the damage

- The four PSW-validity bits, the six register-validity bits, the two timing-facility-validity bits, and the storage-logical-validity bit are all ones.
- Subclass bits 0, 4, 5, 10, and 11 are zeros.
- The instruction-processing-damage bit is zero or, if one, the backed-up bit is also one.
- The delayed-access-exception bit is zero.

As part of the machine-check interruption, in some cases, extended interruption information is placed in fixed areas assigned in storage. The contents of registers associated with the CPU are placed in register-save areas. For external damage, additional information is provided for some models by storing an external-damage code. When storage error uncorrected, storage error corrected, or storage-key error uncorrected is indicated, the failing-storage address is saved.

Register-Save Areas

The following are the eight sets of registers and the real locations in storage where their contents are saved during a machine-check interruption.

4608-4735	Floating-point registers 0-15
4736-4863	General registers 0-15
4892-4895	Floating-point-control register
4900-4903	TOD programmable register
4904-4911	CPU timer
4913-4919	Clock comparator
4928-4991	Access registers 0-15
4992-5119	Control registers 0-15

The word at real location 244 is the external-damage code. This field, when implemented and indicated as valid, describes the cause of external damage. The field is valid only when the external-damage bit and the external-damage-code-validity bit (bits 5 and 26 in the machine-check-interruption code) are both ones. The presence and extent of reporting an external-damage code depend on the model.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	S	C	0	0	0	0	S	I	C	C	0	0	0	0
0																		19						24				28				31

If the condition happens more than once before the interruption occurs, the condition is generated only once. The condition is generated for all CPUs in the configuration, and the condition for a CPU is cleared by the interruption taken by the CPU.

The machine-check condition is generated for all CPUs in the configuration, and the condition for a CPU is cleared by the interruption taken by the CPU.

Chapter 11. Machine-Check Handling **11-21**

server attempts to communicate over an STP link and both of the following conditions are detected:

- The servers have matching CTN IDs, and
- Both servers have valid stratum-1-configurations and the configurations do not match.

The island condition indicates that two separate CTNs may be operating with the same CTN ID.

The machine-check condition is generated for all CPUs in the configuration, and the condition for a CPU is cleared by the interruption taken by the CPU.

CTN Configuration Change (CC): Bit 26, when one, indicates that a CTN-configuration-change has occurred in the CTN. The conditions that cause the CTN-configuration-change condition to be generated include the following:

- A new node with a matching CTN ID has attached to this node.
- An attached node that had a matching CTN ID is no longer attached to this node
- The CTN ID for this node has changed.
- The stratum level of this node has changed, or a change in the stratum-1 or alternate-stratum-1 for the network has changed.
- STP active or maximum version number has changed.

The machine-check condition is generated for all CPUs in the configuration, and the condition for a CPU is cleared by the interruption taken by the CPU.

STP Clock Source Error (CS): Bit 27, when one, indicates that an STP-clock-source error has occurred. The STP-clock-source condition is generated when the configuration is in the STP-timing mode and the STP-clock-source state changes from the usable state to the not-usable state.

The machine-check condition is generated for all CPUs in the configuration, and the condition for a CPU is cleared by the interruption taken by the CPU.

Reserved: Bits 0-7, 10-16, 21-23, and 28-31 are reserved for future expansion and are always set to zero.

Failing-Storage Address

When storage error uncorrected, storage error corrected, or storage-key error uncorrected is indicated in the machine-check-interruption code, the associated address, called the failing-storage address, is stored at real locations 248-255. The field is valid only if the failing-storage-address validity bit, bit 24 of the machine-check-interruption code, is one.

In the case of storage errors, the failing-storage address may designate any byte within the checking block. For storage-key error uncorrected, the failing-storage address may designate any address within the block of storage associated with the storage key that is in error. When an error is detected in more than one location before the interruption, the failing-storage address may designate any of the failing locations. The address stored is an absolute address; that is, the value stored is the address that is used to reference storage after dynamic address translation and prefixing have been applied.

Handling of Machine-Check Conditions

Floating Interruption Conditions

An interruption condition which is made available to any CPU in a multiprocessing configuration is called a floating interruption condition. The first CPU that accepts the interruption clears the interruption condition, and it is no longer available to any other CPU in the configuration.

Floating interruption conditions include service-signal external-interruption and I/O-interruption conditions. Two machine-check-interruption conditions, channel report pending and channel-subsystem damage, are floating interruption conditions. Depending on the model, some machine-check-interruption conditions associated with system recovery and warning may also be floating interruption conditions.

A floating interruption is presented to the first CPU in the configuration which is enabled for the interruption condition and can accept the interruption. A CPU cannot accept the interruption when the CPU is in the check-stop state, has an invalid prefix, is performing an unending string of interruptions due to a PSW-for-

mat error of the type that is recognized early, or is in the stopped state. However, a CPU with the rate control set to instruction step can accept the interruption when the start key is activated.

Programming Note: When a CPU enters the check-stop state in a multiprocessing configuration, the program on another CPU can determine whether a floating interruption may have been reported to the failing CPU and then lost. This can be accomplished if the interruption program places zeros in the real storage locations containing old PSWs and interruption codes after the interruption has been handled (or has been moved into another area for later processing). After a CPU enters the check-stop state, the program on another CPU can inspect the old-PSW and interruption-code locations of the failing CPU. A nonzero value in an old PSW or interruption code indicates that the CPU has been interrupted but the program did not complete the handling of the interruption.

Floating Machine-Check-Interruption Conditions

Floating machine-check-interruption conditions are reset only by the manually initiated resets through the operator facilities. When a machine check occurs which prohibits completion of a floating machine-check interruption, the interruption condition is no longer considered a floating interruption condition, and system damage is indicated.

Floating I/O Interruptions

The detection of a machine malfunction by the channel subsystem, while in the process of presenting an I/O-interruption request for a floating I/O interruption, may be reported as channel report pending or as channel-subsystem damage. Detection of a machine malfunction by a CPU, while in the process of accepting a floating I/O interruption, is reported as system damage.

Machine-Check Masking

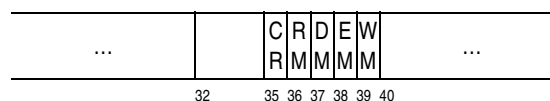
All machine-check interruptions are under control of the machine-check mask, PSW bit 13. In addition, some machine-check conditions are controlled by subclass masks in control register 14.

The exigent machine-check conditions (system damage and instruction-processing damage) are controlled only by the machine-check mask, PSW bit 13.

When PSW bit 13 is one, an exigent condition causes a machine-check interruption. When PSW bit 13 is zero, the occurrence of an exigent machine-check condition causes the CPU to enter the check-stop state.

The repressible machine-check conditions, except channel-subsystem damage and service-processor damage, are controlled both by the machine-check mask, PSW bit 13, and by five subclass-mask bits in control register 14. If PSW bit 13 is one and one of the subclass-mask bits is one, the associated condition initiates a machine-check interruption. If a subclass-mask bit is zero, the associated condition does not initiate an interruption but is held pending. However, when a machine-check interruption is initiated because of a condition for which the CPU is enabled, those conditions for which the CPU is not enabled may be presented along with the condition which initiates the interruption. All conditions presented are then cleared.

Control register 14 contains mask bits that specify whether certain conditions can cause machine-check interruptions. It has the following format:



Bits 35-39 of control register 14 are the subclass masks for repressible machine-check conditions. In addition, bit 32 of control register 14 is initialized to one but is otherwise ignored by the machine.

Programming Note: The program should avoid, whenever possible, operating with PSW bit 13, the machine-check mask, set to zero, since any exigent machine-check condition which is recognized during this situation will cause the CPU to enter the check-stop state. In particular, the program should avoid executing I/O instructions or allowing I/O interruptions with PSW bit 13 zero.

Channel-Report-Pending Subclass Mask

Bit 35 (CM) of control register 14 controls channel-report-pending interruption conditions. This bit is initialized to zero.

Recovery Subclass Mask

Bit 36 (RM) of control register 14 controls system-recovery interruption conditions. This bit is initialized to zero.

Degradation Subclass Mask

Bit 37 (DM) of control register 14 controls degradation interruption conditions. This bit is initialized to zero.

External-Damage Subclass Mask

Bit 38 (EM) of control register 14 controls timing-facility-damage and external-damage interruption conditions. This bit is initialized to one.

Warning Subclass Mask

Bit 39 (WM) of control register 14 controls warning interruption conditions. This bit is initialized to zero.

Machine-Check Logout

As part of the machine-check interruption, some models may place model-dependent information in the fixed-logout area. This area is 16 bytes in length and starts at real location 4864.

Summary of Machine-Check Masking

A summary of machine-check masking is given in Figure 11-4 and Figure 11-5.

Machine-Check Condition		Sub-Class Mask	Action when CPU Disabled for Subclass
MCIC Bit	Subclass		
0	System damage	-	Check stop
1	Instruction-processing damage	-	Check stop
2	System recovery	RM	Y
4	Timing-facility damage	EM	P
5	External damage	EM	P
7	Degradation	DM	P
8	Warning	WM	P
9	Channel report pending	CM	P
10	Service-processor damage	-	P
11	Channel-subsystem damage	-	P
Explanation:			
-	The condition does not have a subclass mask.		
P	Indication is held pending.		
Y	Indication may be held pending or may be discarded.		
CM	Channel-report-pending subclass mask (bit 35 of CR14).		
DM	Degradation subclass mask (bit 37 of CR14).		
EM	External-damage subclass mask (bit 38 of CR14).		
RM	Recovery subclass mask (bit 36 of CR14).		
WM	Warning subclass mask (bit 39 of CR14).		

Figure 11-4. Machine-Check-Condition Masking

Bit Description	Control Register 14 Bit Position	State of Bit on Initial CPU Reset
Channel-report-pending subclass mask	35	0
Recovery subclass mask	36	0
Degradation subclass mask	37	0
External-damage subclass mask	38	1
Warning subclass mask	39	0

Figure 11-5. Machine-Check Control-Register Bits

Chapter 12. Operator Facilities

Manual Operation	12-1	Load-Unit-Address Controls	12-4
Basic Operator Facilities	12-1	Manual Indicator	12-4
Address-Compare Controls	12-1	Power Controls	12-4
Alter-and-Display Controls	12-2	Rate Control	12-4
Architectural-Mode Indicator	12-2	Restart Key	12-4
Architectural-Mode-Selection Controls	12-2	Start Key	12-5
Check-Stop Indicator	12-2	Stop Key	12-5
IML Controls	12-3	Store-Status Key	12-5
Interrupt Key	12-3	System-Reset-Clear Key	12-5
Load Indicator	12-3	System-Reset-Normal Key	12-5
Load-Clear Key	12-3	Test Indicator	12-5
Load-Clear-List-Directed Key	12-3	TOD-Clock Control	12-6
Load-Normal Key	12-3	Wait Indicator	12-6
Load-with-Dump Key	12-3	Multiprocessing Configurations	12-6

Manual Operation

The operator facilities provide functions for the manual operation and control of the machine. The functions include operator-to-machine communication, indication of machine status, control over the setting of the TOD clock, initial program loading, resets, and other manual controls for operator intervention in normal machine operation. A model may not implement all of the operator facilities described in this chapter.

A model may provide additional operator facilities which are not described in this chapter. Examples are the means to indicate specific error conditions in the equipment, to change equipment configurations, and to facilitate maintenance. Furthermore, controls covered in this chapter may have additional settings which are not described here. Such additional facilities and settings may be described in the appropriate System Library publication.

Most models provide, in association with the operator facilities, a console device which may be used as an I/O device for operator communication with the program; this console device may also be used to implement some or all of the facilities described in this chapter.

The operator facilities may be implemented on different models in various technologies and configurations. On some models, more than one set of physical representations of some keys, controls, and

indicators may be provided, such as on multiple local or remote operating stations, which may be effective concurrently.

A machine malfunction that prevents a manual operation from being performed correctly, as defined for that operation, may cause the CPU to enter the check-stop state or give some other indication to the operator that the operation has failed. Alternatively, a machine malfunction may cause a machine-check-interruption condition to be recognized

Basic Operator Facilities

Address-Compare Controls

The address-compare controls provide a way to stop the CPU when a preset address matches the address used in a specified type of main-storage reference.

One of the address-compare controls is used to set up the address to be compared with the storage address.

Another control provides at least two positions to specify the action, if any, to be taken when the address match occurs:

1. The normal position disables the address-compare operation.

2. The stop position causes the CPU to enter the stopped state on an address match. When the control is in this setting, the test indicator is on. Depending on the model and the type of reference, pending I/O, external, and machine-check interruptions may or may not be taken before entering the stopped state.

A third control may specify the type of storage reference for which the address comparison is to be made. A model may provide one or more of the following positions, as well as others:

1. The any position causes the address comparison to be performed on all storage references.
2. The data-store position causes address comparison to be performed when storage is addressed to store data.
3. The I/O position causes address comparison to be performed when storage is addressed by the channel subsystem to transfer data or to fetch a channel-command or indirect-data-address word. Whether references to the measurement block, interruption-response block, channel-path-status word, channel-report word, subchannel-status word, subchannel-information block, and operation-request block cause a match to be indicated depends on the model.
4. The instruction-address position causes address comparison to be performed when storage is addressed to fetch an instruction. The rightmost bit of the address setting may or may not be ignored. The match is indicated only when the first byte of the instruction is fetched from the selected location (which includes fetching the target instruction of an execute-type instruction).

Depending on the model and the type of reference, address comparison may be performed on virtual, real, or absolute addresses, and it may be possible to specify the type of address.

In a multiprocessing configuration, it depends on the model whether the address setting applies to one or all CPUs in the configuration and whether an address match causes one or all CPUs in the configuration to stop.

Alter-and-Display Controls

The operator facilities provide controls and procedures to permit the operator to alter and display the contents of locations in storage, the storage keys, the general, floating-point, floating-point-control, access, and control registers, the prefix, and the PSW.

Before alter-and-display operations may be performed, the CPU must first be placed in the stopped state. During alter-and-display operations, the manual indicator may be turned off temporarily, and the start and restart keys may be inoperative.

Addresses used to select storage locations for alter-and-display operations are real addresses. The capability of specifying logical, virtual, or absolute addresses may also be provided.

Architectural-Mode Indicator

The architectural-mode indicator shows the architectural mode of operation (the ESA/390 mode, z/Architecture mode, or some other mode) selected by the last architectural-mode-selection operation and by the last SIGNAL PROCESSOR set-architecture order or the last reset that determined the mode.

Architectural-Mode-Selection Controls

The architectural-mode-selection controls provide for the selection of either the ESA/390 architectural mode of operation or, possibly, some other architectural mode of operation. (The z/Architecture mode is selected from the ESA/390 mode by the SIGNAL PROCESSOR set-architecture order.) Depending on the model, the architectural-mode selection may be provided as part of the IML operation or may be a separate operation.

As part of the architectural-mode-selection process, all CPUs and the associated channel-subsystem components in a particular configuration are placed in the same architectural mode.

Check-Stop Indicator

The check-stop indicator is on when the CPU is in the check-stop state. Reset operations normally

cause the CPU to leave the check-stop state and thus turn off the indicator. The manual indicator may also be on in the check-stop state.

IML Controls

The IML controls provided with some models perform initial machine loading (IML), which is the loading of licensed internal code into the machine. The IML operation, when provided, may be used to select the ESA/390 mode or, possibly, some other mode of operation.

When the IML operation is completed, the state of the affected CPUs, channel subsystem, main storage, and operator facilities is the same as if a power-on reset had been performed, except that the value and state of the TOD clock are not changed. The contents of expanded storage may have been cleared to zeros with valid checking-block code or may have remained unchanged, depending on the model.

The IML controls are effective while the power is on.

Interrupt Key

When the interrupt key is activated, an external-interruption condition indicating the interrupt key is generated. (See “Interrupt Key” on page 6-11.)

The interrupt key is effective when the CPU is in the operating or stopped state. It depends on the model whether the interrupt key is effective when the CPU is in the load state.

Load Indicator

The load indicator is on during CCW-type initial program loading, indicating that the CPU is in the load state. The indicator goes on for a particular CPU when the load-clear or load-normal key is activated for that CPU and the corresponding operation is started. It goes off after the new PSW is loaded successfully. For details, see “Initial Program Loading” on page 4-56.)

Load-Clear Key

Activating the load-clear key causes a reset operation to be performed and CCW-type initial program loading to be started by using the I/O device designated by the load-unit-address controls. Clear reset is performed on the configuration. For details, see “Resets” on page 4-50 and “Initial Program Loading” on page 4-56.

The load-clear key is effective when the CPU is in the operating, stopped, load, or check-stop state.

Load-Clear-List-Directed Key

Activating the load-clear-list-directed key causes a reset operation to be performed and list-directed initial program loading to be started using the I/O device designated by the list-directed IPL parameters. Clear reset is performed on the configuration. For details, see “Resets” on page 4-50 and “List-Directed IPL” on page 17-19.

The load-clear-list-directed key is effective when the CPU is in the operating, stopped, load, or check-stop state.

Load-Normal Key

Activating the load-normal key causes a reset operation to be performed and CCW-type initial program loading to be started by using the I/O device designated by the load-unit-address controls. Initial CPU reset is performed on the CPU for which the load-normal key was activated, CPU reset is propagated to all other CPUs in the configuration, and a subsystem reset is performed on the remainder of the configuration. For details, see “Resets” on page 4-50 and “Initial Program Loading” on page 4-56.

The load-normal key is effective when the CPU is in the operating, stopped, load, or check-stop state.

Load-with-Dump Key

Activating the load-with-dump key causes a reset operation to be performed and list-directed initial program loading to be started by using the I/O devices designated by the list-directed IPL parameters. A store-status operation followed by an initial CPU reset is performed on the CPU for which the load-

with-dump key was activated, CPU reset is propagated to all other CPUs in the configuration, and a subsystem reset is performed on the remainder of the configuration. For details, see “Resets” on page 4-50 and “List-Directed IPL” on page 17-19.

The load-with-dump key is effective when the CPU is in the operating, stopped, load, or check-stop state.

Load-Unit-Address Controls

The load-unit-address controls specify four hexadecimal digits that provide the device number used for CCW-type initial program loading. For details, see “Initial Program Loading” on page 4-56.

Manual Indicator

The manual indicator is on when the CPU is in the stopped state. Some functions and several manual controls are effective only when the CPU is in the stopped state.

Power Controls

The power controls are used to turn the power on and off.

The CPUs, storage, channel subsystem, operator facilities, and I/O devices may all have their power turned on and off by common controls, or they may have separate power controls. When a particular unit has its power turned on, that unit is reset. The sequence is performed so that no instructions or I/O operations are performed until explicitly specified. The controls may also permit power to be turned on in stages, but the machine does not become operational until power on is complete.

When the power is completely turned on, an IML operation is performed on models which have an IML function. A power-on reset is then initiated (see “Resets” on page 4-50). It depends on the model whether the architectural mode of operation can be selected when the power is turned on, or whether the mode-selection controls have to be used to change the mode after the power is on.

Rate Control

A model may provide the rate control, the setting of which determines the effect of the start function and the manner in which instructions are executed.

When the rate control is provided, it has at least two positions. The normal position is the process position. Another position is the instruction-step position. When the rate control is set to the process position and the start function is performed, the CPU starts operating at normal speed. When the rate control is set to the instruction-step position and the wait-state bit is zero, one instruction or, for interruptible instructions, one unit of operation is executed, and all pending allowed interruptions are taken before the CPU returns to the stopped state. When the rate control is set to the instruction-step position and the wait-state bit is one, no instruction is executed, but all pending allowed interruptions are taken before the CPU returns to the stopped state. For details, see “Stopped, Operating, Load, and Check-Stop States” on page 4-2.

The test indicator is on while the rate control is not set to the process position.

If the setting of the rate control is changed while the CPU is in the operating or load state, the results are unpredictable.

When the rate control is not provided, the CPU operates as though the rate control was in the process position, and instruction stepping is not available.

Restart Key

Activating the restart key initiates a restart interruption. (See “Restart Interruption” on page 6-49.)

The restart key is effective when the CPU is in the operating or stopped state. The key is not effective when the CPU is in the check-stop state. It depends on the model whether the restart key is effective when any CPU in the configuration is in the load state.

The effect is unpredictable when the restart key is activated while any CPU in the configuration is in the load state. In particular, if the CPU performs a restart interruption and enters the operating state while another CPU is in the load state, operations such as

I/O instructions, the SIGNAL PROCESSOR instruction, and the INVALIDATE PAGE TABLE ENTRY instruction may not operate according to the definitions given in this publication

Start Key

Activating the start key causes the CPU to perform the start function. (See “Stopped, Operating, Load, and Check-Stop States” on page 4-2.)

The start key is effective only when the CPU is in the stopped state. The effect is unpredictable when the stopped state has been entered by a reset.

Stop Key

Activating the stop key causes the CPU to perform the stop function. (See “Stopped, Operating, Load, and Check-Stop States” on page 4-2.)

The stop key is effective only when the CPU is in the operating state.

Operation Note: Activating the stop key has no effect when:

- An unending string of certain program or external interruptions occurs.
- The prefix register contains an invalid address.
- The CPU is in the load or check-stop state.

Store-Status Key

Activating the store-status key initiates a store-status operation. (See “Store Status” on page 4-56.)

The store-status key is effective only when the CPU is in the stopped state.

Operation Note: The store-status operation may be used in conjunction with a standalone dump program for the analysis of major program malfunctions. For such an operation, the following sequence would be called for:

1. Activation of the stop or system-reset-normal key
2. Activation of the store-status key
3. Activation of the load-normal key to enter a stand-alone dump program

The system-reset-normal key must be activated in step 1 when (1) the stop key is not effective because a continuous string of interruptions is occurring, (2) the prefix register contains an invalid address, or (3) the CPU is in the check-stop state.

System-Reset-Clear Key

Activating the system-reset-clear key causes a clear-reset operation to be performed on the configuration. For details, see “Resets” on page 4-50.

The system-reset-clear key is effective when the CPU is in the operating, stopped, load, or check-stop state.

System-Reset-Normal Key

Activating the system-reset-normal key causes a CPU-reset operation and a subsystem-reset operation to be performed. In a multiprocessing configuration, a CPU reset is propagated to all CPUs in the configuration. For details, see “Resets” on page 4-50.

The system-reset-normal key is effective when the CPU is in the operating, stopped, load, or check-stop state.

Test Indicator

The test indicator is on when a manual control for operation or maintenance is in an abnormal position that can affect the normal operation of a program.

Setting the address-compare controls to the stop position or setting the rate control to the instruction-step position turns on the test indicator.

The test indicator may be on when one or more diagnostic functions under the control of DIAGNOSE are activated, or when other abnormal conditions occur.

The abnormal setting of a manual control causes the test indicator of the affected CPU to be turned on; however, in a multiprocessing configuration, the operation of other CPUs may be affected even though their test indicators are not turned on.

Operation Note: If a manual control is left in a setting intended for maintenance purposes, such an abnormal setting may, among other things, result in

false machine-check indications or cause actual machine malfunctions to be ignored. It may also alter other aspects of machine operation, including instruction execution, channel-subsystem operation, and the functioning of operator controls and indicators, to the extent that operation of the machine does not comply with that described in this publication.

TOD-Clock Control

When the TOD-clock control is not activated, that is, the control is set to the secure position, the state and value of the TOD clock are protected against unauthorized or inadvertent change by not permitting the instructions SET CLOCK or DIAGNOSE to change the state or value.

When the TOD-clock control is activated, that is, the control is set to the enable-set position, alteration of the clock state or value by means of SET CLOCK or DIAGNOSE is permitted. This setting is momentary, and the control automatically returns to the secure position.

If there is more than one physical representation of the TOD-clock control, the TOD clock is secure only if all TOD-clock controls in the configuration are set to the secure position.

Wait Indicator

The wait indicator is on when the wait-state bit in the current PSW is one. Instead of a wait indicator, a

model may have a means of indicating a time-averaged value of the wait-state bit.

Multiprocessing Configurations

In a multiprocessing configuration, one of each of the following keys and controls is provided for each CPU: alter and display, interrupt, rate, restart, start, stop, and store status. The load-clear key, load-normal key, and load-unit-address controls are provided for each CPU capable of performing I/O operations. Alternatively, a single set of initial-program-loading keys and controls may be used together with a control to select the desired CPU.

There need not be more than one of each of the following keys and controls in a multiprocessing configuration: address compare, IML, power, system reset clear, system reset normal, and TOD clock.

One check-stop, manual, test, and wait indicator is provided for each CPU. A load indicator is provided only on a CPU capable of performing I/O operations. Alternatively, a single set of indicators may be switched to more than one CPU.

There need not be more than one architectural-mode indicator in a multiprocessing configuration.

In a system capable of reconfiguration, there must be a separate set of keys, controls, and indicators in each configuration.

Chapter 13. I/O Overview

Input/Output (I/O)	13-1	Transport Command Word (TCW)	13-7
The Channel Subsystem	13-1	Channel Program Organization	13-7
Subchannel Sets	13-2	CCW Channel Program	13-7
Subchannels	13-2	TCW Channel Program	13-7
Attachment of Input/Output Devices	13-3	Execution of I/O Operations	13-8
Channel Paths	13-3	Start-Function Initiation	13-8
Control Units	13-4	Subchannel Operation Modes	13-9
I/O Devices	13-5	Path Management	13-9
I/O Addressing	13-5	Channel-Program Execution	13-9
Subchannel-Set Identifier	13-5	Conclusion of I/O Operations	13-10
Channel-Path Identifier	13-5	Chaining When Using a CCW Channel	
Subchannel Number	13-5	Program	13-11
Device Number	13-6	Chaining When Using a TCW Channel	
Device Identifier	13-6	Program	13-11
Fibre-Channel Extensions	13-6	Premature Conclusion of I/O Operations	13-12
I/O-Command Words	13-6	I/O Interruptions	13-12

Input/Output (I/O)

The terms “input” and “output” are used to describe the transfer of data between I/O devices and main storage. An operation involving this kind of transfer is referred to as an I/O operation. The facilities used to control I/O operations are collectively called the channel subsystem. (I/O devices and their control units attach to the channel subsystem.) This chapter provides a brief description of the basic components and operation of the channel subsystem.

The Channel Subsystem

The channel subsystem directs the flow of information between I/O devices and main storage. It relieves CPUs of the task of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O processing. The channel subsystem uses one or more channel paths as the communication link in managing the flow of information to or from I/O devices. As part of I/O processing, the channel subsystem also performs a path-management operation by testing for channel-path availability, chooses an available channel path, and initiates the performance of the I/O operation by the device.

Within the channel subsystem are subchannels. One subchannel is provided for and dedicated to each I/O device accessible to the program through the channel subsystem.

The multiple-subchannel-set facility is an optional facility. When it is installed, subchannels are partitioned into multiple subchannel sets, and each subchannel set may provide one dedicated subchannel to an I/O device. Depending on the model and the interface used, some I/O devices may only be allowed to be accessed via certain subchannel sets.

Each subchannel provides information concerning the associated I/O device and its attachment to the channel subsystem. The subchannel also provides information concerning I/O operations and other functions involving the associated I/O device. The subchannel is the means by which the channel subsystem provides information about associated I/O devices to CPUs, which obtain this information by executing I/O instructions. The actual number of subchannels provided depends on the model and the configuration; the maximum addressability is 0-65,535 in each subchannel set.

I/O devices are attached through control units to the channel subsystem by means of channel paths. Control units may be attached to the channel subsystem by more than one channel path, and an I/O device may be attached to more than one control unit. In all,

an individual I/O device may be accessible to the channel subsystem by as many as eight different channel paths via a subchannel, depending on the model and the configuration. The total number of channel paths provided by a channel subsystem depends on the model and the configuration; the maximum addressability is 0-255.

The performance of a channel subsystem depends on its use and on the system model in which it is implemented. Channel paths are provided with different data-transfer capabilities, and an I/O device designed to transfer data only at a specific rate (a magnetic-tape unit or a disk storage, for example) can operate only on a channel path that can accommodate at least this data rate.

The channel subsystem contains common facilities for the control of I/O operations. When these facilities are provided in the form of separate, autonomous equipment designed specifically to control I/O devices, I/O operations are completely overlapped with the activity in CPUs. The only main-storage cycles required by the channel subsystem during I/O operations are those needed to transfer data and control information to or from the final locations in main storage, along with those cycles that may be required for the channel subsystem to access the subchannels when they are implemented as part of nonaddressable main storage. These cycles do not delay CPU programs, except when both the CPU and the channel subsystem concurrently attempt to reference the same main-storage area.

Subchannel Sets

When the multiple-subchannel-set facility is installed, subchannels are partitioned into multiple subchannel sets. There may be up to four subchannel sets, each identified by a subchannel-set identifier (SSID). When the multiple-subchannel-set facility is not installed, there is only one subchannel set with an SSID of zero. When the multiple-subchannel-set facility is not enabled, only subchannel set zero is visible to the program. See “Multiple-Subchannel-Set Facility” on page 17-29 for more information on the multiple-subchannel-set facility.

Subchannels

A subchannel provides the logical appearance of a device to the program and contains the information

required for sustaining a single I/O operation. The subchannel consists of internal storage that contains information in the form of a channel-program designation, channel-path identifier, device number, count, status indications, and I/O-interruption-subclass code, as well as information on path availability and functions pending or being performed. I/O operations are initiated with a device by the execution of I/O instructions that designate the subchannel associated with the device.

Each device is accessible by means of one subchannel in each channel subsystem to which it is assigned during configuration at installation time. The device may be a physically identifiable unit or may be housed internal to a control unit. For example, in certain disk-storage devices, each actuator used in retrieving data is considered to be a device. In all cases, a device, from the point of view of the channel subsystem, is an entity that is uniquely associated with one subchannel and that responds to selection by the channel subsystem by using the communication protocols defined for the type of channel path by which it is accessible.

On some models, subchannels are provided in blocks. On these models, more subchannels may be provided than there are attached devices. Subchannels that are provided but do not have devices assigned to them are not used by the channel subsystem to perform any function and are indicated by storing the associated device-number-valid bit as zero in the subchannel-information block of the subchannel.

The number of subchannels provided by the channel subsystem is independent of the number of channel paths to the associated devices. For example, a device accessible through alternate channel paths still is represented by a single subchannel. Each subchannel is addressed by using a 16-bit binary subchannel number and a two-bit SSID when the subchannel-set facility is installed.

After I/O processing at the subchannel has been requested by the execution of START SUBCHANNEL, the CPU is released for other work, and the channel subsystem assembles or disassembles data and synchronizes the transfer of data bytes between the I/O device and main storage. To accomplish this, the channel subsystem maintains and updates an address and a count that describe the destination or source of data in main storage. Similarly, when an I/O device provides signals that should be brought to the

attention of the program, the channel subsystem transforms the signals into status information and stores the information in the subchannel, where it can be retrieved by the program.

Attachment of Input/Output Devices

Channel Paths

The channel subsystem communicates with I/O devices by means of channel paths between the channel subsystem and control units. A control unit may be accessible by the channel subsystem by more than one channel path. Similarly, an I/O device may be accessible by the channel subsystem through more than one control unit, each having one or more channel paths to the channel subsystem.

Devices that are attached to the channel subsystem by multiple channel paths configured to a subchannel, may be accessed by the channel subsystem using any of the available channel paths. Similarly, a device having the dynamic-reconnection feature and operating in the multipath mode can be initialized to operate such that the device may choose any of the available channel paths configured to the subchannel, when logically reconnecting to the channel subsystem to continue a chain of I/O operations.

The channel subsystem may contain more than one type of channel path. Examples of channel-path types used by the channel subsystem are the ESCON I/O interface, FICON I/O interface, FICON-converted I/O interface, and IBM System/360 and System/370 I/O interface. The term “serial-I/O interface” is used to refer the ESCON I/O interface, the FICON I/O interface, and the FICON-converted I/O interface. The term “parallel-I/O interface” is used to refer to the IBM System/360 and System/370 I/O interface.

The ESCON I/O interface is described in the System Library publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

The FICON I/O interface is described in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*

The IBM System/360 and System/370 I/O interface is described in the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974.

Depending on the type of channel path, the facilities provided by the channel path, and the I/O device, an I/O operation may occur in one of three modes, frame-multiplex mode, burst mode, or byte-multiplex mode.

In the frame-multiplex mode, the I/O device may stay logically connected to the channel path for the duration of the execution of a channel program. The facilities of a channel path capable of operating in the frame-multiplex mode may be shared by a number of concurrently operating I/O devices. In this mode the information required to complete an I/O operation is divided into frames that may be interleaved with frames from I/O operations for other I/O devices. During this period, multiple I/O devices are considered to be logically connected to the channel path.

In the burst mode, the I/O device monopolizes a channel path and stays logically connected to the channel path for the transfer of a burst of information. No other device can communicate over the channel path during the time a burst is transferred. The burst can consist of a few bytes, a whole block of data, a sequence of blocks with associated control and status information (the block lengths may be zero), or status information that monopolizes the channel path. The facilities of the channel path capable of operating in the burst mode may be shared by a number of concurrently operating I/O devices.

Some channel paths can tolerate an absence of data transfer for about a half minute during a burst-mode operation, such as occurs when a long gap on magnetic tape is read. An equipment malfunction may be indicated when an absence of data transfer exceeds the prescribed limit.

In the byte-multiplex mode, the I/O device stays logically connected to the channel path only for a short interval of time. The facilities of a channel path capable of operating in the byte-multiplex mode may be shared by a number of concurrently operating I/O devices. In this mode, all I/O operations are split into short intervals of time during which only a segment of information is transferred over the channel path. During such an interval, only one device and its associated subchannel are logically connected to the channel path. The intervals associated with the con-

current operation of multiple I/O devices are sequenced in response to demands from the devices. The channel-subsystem facility associated with a subchannel exercises its controls for any one operation only for the time required to transfer a segment of information. The segment can consist of a single byte of data, a few bytes of data, a status report from the device, or a control sequence used for the initiation of a new operation.

Ordinarily, devices with high data-transfer-rate requirements operate with the channel path in the frame-multiplex mode, slower devices operate in the burst mode, and the slowest devices operate in the byte-multiplex mode. Some control units have a manual switch for setting the desired mode of operation.

An I/O operation that occurs on a parallel-I/O-interface type of channel path may occur in either the burst mode or the byte-multiplex mode depending on the facilities provided by the channel path and the I/O device. For improved performance, some channel paths and control units are provided with facilities for high-speed transfer and data streaming. See the *System Library publication IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI, GA22-6974*, for a description of those two facilities.

An I/O operation that occurs on a serial-I/O-interface type of channel path may occur in either the frame-multiplex mode or the burst mode. For improved performance, some control units attaching to the serial-I/O interface provide the capability to provide sense data to the program concurrent with the presentation of unit-check status, if permitted to do so by the program. (See "Concurrent Sense" on page 17-24.)

Depending on the control unit or channel subsystem, access to a device through a subchannel may be restricted to a single channel-path type.

The modes and features described above affect only the protocol used to transfer information over the channel path and the speed of transmission. No effects are observable by CPU or channel programs with respect to the way these programs are executed.

Control Units

A control unit provides the logical capabilities necessary to operate and control an I/O device and adapts

the characteristics of each device so that it can respond to the standard form of control provided by the channel subsystem.

Communication between the control unit and the channel subsystem takes place over a channel path. The control unit accepts control signals from the channel subsystem, controls the timing of data transfer over the channel path, and provides indications concerning the status of the device.

The I/O device attached to the control unit may be designed to perform only certain limited operations, or it may perform many different operations. A typical operation is moving a recording medium and recording data. To accomplish its operations, the device needs detailed signal sequences peculiar to its type of device. The control unit decodes the commands received from the channel subsystem, interprets them for the particular type of device, and provides the signal sequence required for the performance of the operation.

A control unit may be housed separately, or it may be physically and logically integrated with the I/O device, the channel subsystem, or a CPU. In the case of most electromechanical devices, a well-defined interface exists between the device and the control unit because of the difference in the type of equipment the control unit and the device require. These electromechanical devices often are of a type where only one device of a group attached to a control unit is required to transfer data at a time (magnetic-tape units or disk-access mechanisms, for example), and the control unit is shared among a number of I/O devices. On the other hand, in some electronic I/O devices, such as the channel-to-channel adapter, the control unit does not have an identity of its own.

From the programmer's point of view, most functions performed by the control unit can be merged with those performed by the I/O device. Therefore, this publication normally makes no specific mention of the control-unit function; the performance of I/O operations is described as if the I/O devices communicated directly with the channel subsystem. Reference is made to the control unit only when emphasizing a function performed by it or when describing how the sharing of the control unit among a number of devices affects the performance of I/O operations.

I/O Devices

An input/output (I/O) device provides external storage, a means of communication between data-processing systems, or a means of communication between a system and its environment. I/O devices include such equipment as magnetic-tape units, direct-access-storage devices (for example, disks), display units, typewriter-keyboard devices, printers, teleprocessing devices, and sensor-based equipment. An I/O device may be physically distinct equipment, or it may share equipment with other I/O devices.

Most types of I/O devices, such as printers, or tape devices, use external media, and these devices are physically distinguishable and identifiable. Other types are solely electronic and do not directly handle physical recording media. The channel-to-channel adapter, for example, provides for data transfer between two channel paths, and the data never reaches a physical recording medium outside main storage. Similarly, communication controllers may handle the transmission of information between the data-processing system and a remote station, and its input and output are signals on a transmission line.

In the simplest case, an I/O device is attached to one control unit and is accessible from one channel path. Switching equipment is available to make some devices accessible from two or more channel paths by switching devices among control units and by switching control units among channel paths. Such switching equipment provides multiple paths by which an I/O device may be accessed. Multiple channel paths to an I/O device are provided to improve performance or I/O availability, or both, within the system. The management of multiple channel paths to devices is under the control of the channel subsystem and the device, but the channel paths may indirectly be controlled by the program.

I/O Addressing

Four different types of I/O addressing are provided by the channel subsystem for the necessary addressing of the various components: channel-path identifiers, subchannel numbers, device numbers, and, though not visible to programs, addresses dependent on the channel-path type. When the multiple-subchannel-

set facility is installed, the subchannel-set identifier (SSID) is also used in I/O addressing.

Subchannel-Set Identifier

The subchannel-set identifier (SSID) is a two-bit value assigned to each provided subchannel set.

Channel-Path Identifier

The channel-path identifier (CHPID) is a system-unique eight-bit value assigned to each installed channel path of the system. A CHPID is used to address a channel path. A CHPID is specified by the second-operand address of RESET CHANNEL PATH and used to designate the channel path that is to be reset. The channel paths by which a device is accessible are identified in the subchannel-information block (SCHIB), each by its associated CHPID, when STORE SUBCHANNEL is executed. The CHPID can also be used in operator messages when it is necessary to identify a particular channel path. A system model may provide as many as 256 channel paths. The maximum number of channel paths and the assignment of CHPIDs to channel paths depends on the system model.

Subchannel Number

A subchannel number is a system-unique 16-bit value used to address a subchannel. This value is unique within a subchannel set of a channel subsystem. The subchannel is addressed by eight I/O instructions: CANCEL SUBCHANNEL, CLEAR SUBCHANNEL, HALT SUBCHANNEL, MODIFY SUBCHANNEL, RESUME SUBCHANNEL, START SUBCHANNEL, STORE SUBCHANNEL, and TEST SUBCHANNEL. All I/O functions relative to a specific I/O device are specified by the program by designating a subchannel assigned to the I/O device. Subchannels in each subchannel set are always assigned subchannel numbers within a single range of contiguous numbers. The lowest-numbered subchannel is subchannel 0. The highest-numbered subchannel of the channel subsystem has a subchannel number equal to one less than the number of subchannels provided. A maximum of 65,536 subchannels can be provided in each subchannel set. Normally, subchannel numbers are only used in communication between the CPU program and the channel subsystem.

Device Number

Each subchannel that has an I/O device assigned to it also contains a parameter called the device number. The device number is a 16-bit value that is assigned as one of the parameters of the subchannel at the time the device is assigned to the subchannel. The device number uniquely identifies a device to the program.

The device number provides a means to identify a device, independent of any limitations imposed by the system model, the configuration, or channel-path protocols. The device number is used in communications concerning the device that take place between the system and the system operator. For example, the device number is entered by the system operator to designate the input device to be used for initial program loading.

Programming Note: The device number is assigned at device-installation time and may have any value. However, the user must observe any restrictions on device-number assignment that may be required by the control program, support programs, or the particular control unit or I/O device.

Device Identifier

A device identifier is an address, not apparent to the program, that is used by the channel subsystem to communicate with I/O devices. The type of device identifier used depends on the specific channel-path type and the protocols provided. Each subchannel contains one or more device identifiers.

For a channel path of the parallel-I/O-interface type, the device identifier is called a device address and consists of an eight-bit value. For the ESCON I/O interface, the device identifier consists of a four-bit control-unit address and an eight-bit device address. For the FICON I/O interface, the device identifier consists of an eight-bit control-unit-image ID and an eight-bit device address. For the FICON-converted I/O interface, the device identifier consists of a four-bit control-unit address and an eight-bit device address.

The device address identifies the particular I/O device (and, on the parallel-I/O interface, the control unit) associated with a subchannel. The device address may identify, for example, a particular mag-

netic-tape drive, disk-access mechanism, or transmission line. Any number in the range 0-255 can be assigned as a device address.

For further information about the device identifier used with a particular channel-path type, see the appropriate publication for the channel-path type.

Fibre-Channel Extensions

The fibre-channel-extensions (FCX) facility is an optional facility that provides for the formation of a channel program that is composed of a transport-control word (TCW) that designates a transport-command-control block (TCCB) and a transport-status block (TSB). The TCCB includes a transport-command area (TCA) which contains a list of up to 30 I/O commands that are in the form of device-command words (DCWs). A TCW and its TCCB may specify either a read or a write operation. In addition to the IRB, the TSB contains the completion status and other information related to the TCW channel program.

The FCX facility provides the ability to directly or indirectly designate any or all of the TCCB, the input data storage area, and the output data storage area. When a storage area is designated directly, the TCW specifies the location of a single, contiguous block of storage. When a storage area is designated indirectly, the TCW designates the location of a list of one or more transport-indirect-data-address words (TIDAWs). TIDAW lists and the storage area designated by each TIDAW in a list are restricted from crossing 4K-byte boundaries

The FCX facility also provides an interrogate operation that may be initiated by the CANCEL SUBCHANNEL instruction to determine the state of an I/O operation.

I/O-Command Words

An I/O-command word specifies a command and contains information associated with the command. When the FCX facility is installed, there are two elemental forms of I/O command words which are the channel-command word (CCW) and the device-command word (DCW).

A CCW is 8-bytes in length and specifies the command to be executed. For commands that initiate certain operations the CCW also designates the storage area associated with the operation, the count of data bytes, the action to be taken when the command completes, and other options. All I/O devices recognize CCWs.

A DCW is 8-bytes in length and specifies the command to be executed, the count of data bytes, and other options. I/O devices that support FCX recognize DCWs.

Transport Command Word (TCW)

A TCW designates a transport-command-control block (TCCB) which contains a list of commands to be transported to and executed by an I/O device. The TCW also designates the storage areas for the commands in the TCCB as well as a transport-status block (TSB) to contain the status of the I/O operation.

Channel Program Organization

When the FCX facility is not installed, there is a single form of channel program which is the CCW channel program. When the FCX facility is installed, there is an additional form of channel program which is the TCW channel program. Both forms of channel programs are described below.

CCW Channel Program

A channel program that is composed of one or more CCWs is called a CCW channel program (CCP). Such a channel program contains one or more CCWs that are logically linked and arranged for sequential execution by the channel subsystem.

The following figure shows a conceptual example of a simple CCW channel program in program storage. This channel program contains 4 CCWs and specifies the transfer of data to or from contiguous areas of storage. The first CCW designates a control com-

mand and the remaining CCWs designate the transfer of data.

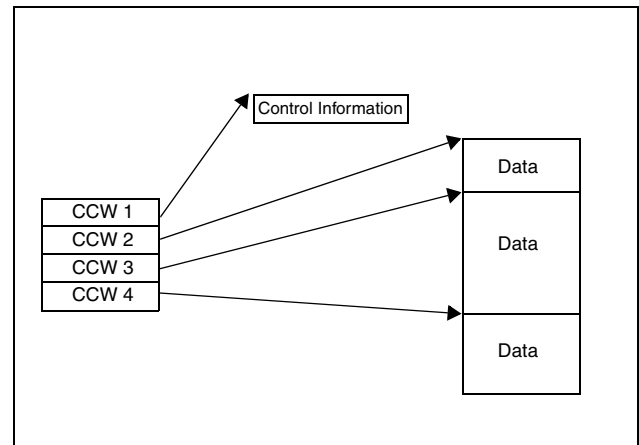


Figure 1. CCW Channel Program Example

TCW Channel Program

A channel program that is composed of a single TCW is called a TCW channel program. A TCW designates a transport-command-control block (TCCB) that contains from 1 to 30 DCWs.

When DCWs within the TCCB are logically linked and arranged for sequential execution. For DCWs that specify control information, the TCCB also contains the control information for those commands. The TCW also designates the storage area or areas for the DCWs that specify the transfer of data from or to the device and the location of a transport-status block (TSB) for completion status. The TCCB and the storage areas for the transfer of data may be specified as either contiguous or noncontiguous storage.

The following figure shows a conceptual example of a TCW channel program in program storage. This channel program is similar to the CCW channel program in Figure 1 on page 13-7. This channel program is composed of a TCW that designates a TCCB containing 4 DCWs, all of which specify the transfer of data either to or from contiguous areas of storage. The first DCW designates a control command and the remaining DCWs designate the transfer of data.

The TCW also designates a TSB for completion status.

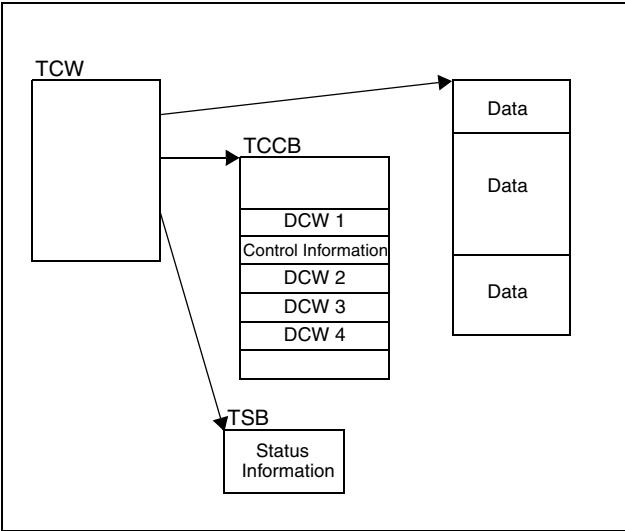


Figure 2. Example of TCW Channel Program Example Designating I/O

Execution of I/O Operations

I/O operations are initiated and controlled by information with four types of formats: the instruction START SUBCHANNEL, transport-command words, I/O-command words, and orders. The START SUBCHANNEL instruction is executed by a CPU and is part of the CPU program that supervises the flow of requests for I/O operations from other programs that manage or process the I/O data.

When START SUBCHANNEL is executed, parameters are passed to the target subchannel requesting that the channel subsystem perform a start function with the I/O device associated with the subchannel. The channel subsystem performs the start function by using information at the subchannel, including the information passed during the execution of the START SUBCHANNEL instruction, to find an accessible channel path to the device. Once the device has been selected, the execution of an I/O operation is accomplished by the decoding and execution of a CCW by the channel subsystem and the I/O device, for CCW channel programs, or for TCW channel programs, by transporting the TCCB to the I/O device by the channel subsystem and the decoding and execution of a DCW by the device. I/O-command words, and transport-command words are fetched from main storage, although the modifier bits in the command

code of a CCW DCW may specify device-dependent conditions for the execution of an operation at the device.

Operations peculiar to a device, such as rewinding tape or positioning the access mechanism on a disk drive, are specified by orders that are decoded and executed by I/O devices. Orders may be transferred to the device as modifier bits in the command code of a control command, may be transferred to the device as data during a control or write operation, or may be made available to the device by other means.

Start-Function Initiation

CPU programs initiate I/O operations with the instruction START SUBCHANNEL. This instruction passes the contents of an operation-request block (ORB) to the subchannel.

If the ORB specifies a CCW channel program, the contents of the ORB include the subchannel key, the address of the first CCW to be executed, and a specification of the format of the CCWs. The CCW specifies the command to be executed and the storage area, if any, to be used. If the ORB specifies a TCW channel program, the contents of the ORB include the subchannel key and the address of the TCW to be executed. The TCW designates the TCCB which contains the commands to be transported to the device for execution, the storage area or areas, if any, to be used for data transfer, and the TSB to contain the status of the I/O operation.

When the ORB contents have been passed to the subchannel, the execution of START SUBCHANNEL is complete. The results of the execution of the instruction are indicated by the condition code set in the program-status word.

When facilities become available and the ORB specifies a CCW channel program, the channel subsystem fetches the first CCW and decodes it according to the format bit specified in the ORB. If the format bit is zero, format-0 CCWs are specified. If the format bit is one, format-1 CCWs are specified. Format-0 and format-1 CCWs contain the same information, but the fields are arranged differently in the format-1 CCW so that 31-bit addresses can be specified directly in the CCW. When facilities become available and the ORB specifies a TCW channel program, the channel subsystem fetches the designated TCW and transports the designated TCCB to the device. Storage areas

designated by the TCW for the transfer of data to or from the device are 64-bit addresses.

Subchannel Operation Modes

There are two modes of subchannel operation. A subchannel enters transport mode when the FCX facility is installed and the start function is set at the subchannel as the result of the execution of a START SUBCHANNEL instruction that specifies a TCW channel program. The subchannel remains in transport mode until the start function is reset at the subchannel. At all other times, the subchannel is in command mode.

Path Management

If ORB specifies a CCW channel program and the first CCW passes certain validity tests and does not have the suspend flag specified as one or if the ORB specifies a TCW channel program and the designated TCW passes certain validity tests, the channel subsystem attempts device selection by choosing a channel path from the group of channel paths that are available for selection. A control unit that recognizes the device identifier connects itself logically to the channel path and responds to its selection.

If the ORB specifies a CCW channel program, the channel subsystem sends the command-code part of the CCW over the channel path, and the device responds with a status byte indicating whether the command can be executed. The control unit may logically disconnect from the channel path at this time, or it may remain connected to initiate data transfer.

If the ORB specifies a TCW channel program, the channel subsystem uses information in the designated TCW to transfer the TCCB to the control unit. The contents of the TCCB are ignored by the channel subsystem and only have meaning to the control unit and I/O device.

If the attempted selection does not occur as a result of either a busy indication or a path-not-operational condition, the channel subsystem attempts to select the device by an alternate channel path if one is available. When selection has been attempted on all paths available for selection and the busy condition persists, the operation remains pending until a path becomes free. If a path-not-operational condition is detected on one or more of the channel paths on which device selection was attempted, the program is alerted by a subsequent I/O interruption. The I/O

interruption occurs either upon execution of the channel program (assuming the device was selected on an alternate channel path) or as a result of the execution being abandoned because path-not-operational conditions were detected on all of the channel paths on which device selection was attempted.

Channel-Program Execution

If the command is initiated at the device and command execution does not require any data to be transferred to or from the device, the device may signal the end of the operation immediately on receipt of the command code. In operations that involve the transfer of data, the subchannel is set up so that the channel subsystem will respond to service requests from the device and assume further control of the operation.

An I/O operation may involve the transfer of data to or from one storage area, designated by a single CCW or TCW or to or from a number of noncontiguous storage areas. In the latter case, generally a list of CCWs is used for the execution of the I/O operation, with each CCW designating a contiguous storage area and the CCWs are coupled by data chaining. Data chaining is specified by a flag in the CCW and causes the channel subsystem to fetch another CCW upon the exhaustion or filling of the storage area designated by the current CCW. The storage area designated by a CCW fetched on data chaining pertains to the I/O operation already in progress at the I/O device, and the I/O device is not notified when a new CCW is fetched.

Provision is made in the CCW format for the programmer to specify that, when the CCW is decoded, the channel subsystem request an I/O interruption as soon as possible, thereby notifying a CPU program that chaining has progressed at least as far as that CCW in the channel program.

To complement dynamic address translation in CPUs, CCW indirect data addressing and modified CCW indirect data addressing are provided.

When the ORB specifies a CCW channel program and CCW-indirect-data addressing is used, a flag in the CCW specifies that an indirect-data-address list is to be used to designate the storage areas for that CCW. Each time the boundary of a block of storage is reached, the list is referenced to determine the next block of storage to be used. The ORB specifies

whether the size of each block of storage is 2K bytes or 4K bytes.

When the ORB specifies a CCW channel program and modified-CCW-indirect-data addressing is used, a flag in the ORB and a flag in the CCW specify that a modified-indirect-data-address list is to be used to designate the storage areas for that CCW. Each time the count of bytes specified for a block of storage is reached, the list is referenced to determine the next block of storage to be used. Unlike when indirect data addressing is used, the block may be specified on any boundary and length up to 4K, provided a data transfer across a 4K-byte boundary is not specified.

When the ORB specifies a TCW channel program and transport-indirect-data addressing is used, flags in the TCW specify whether a transport-indirect-data-address list is to be used to designate the storage areas containing the TCCB and whether a transport-indirect-data-address list is used to designate the data storage areas associated with the DCWs in the TCCB. Each time the count of bytes specified for a block of storage is reached, the corresponding transport-indirect-data-address list is referenced to determine the next storage block to be used.

CCW indirect data addressing and modified CCW indirect data addressing permit essentially the same CCW sequences to be used for a program running with dynamic address translation active in the CPU as would be used if the CPU were operating with equivalent contiguous real storage. CCW indirect data addressing permits the program to designate data blocks having absolute storage addresses up to $2^{64}-1$ independent of whether format-0 or format-1 CCWs have been specified in the ORB. Modified CCW indirect data addressing permits the program to designate data blocks having absolute storage addresses up to $2^{64}-1$, independent of whether format-0 or format-1 CCWs have been specified in the ORB.

In general, the execution of an I/O operation or chain of operations involves as many as three levels of participation:

1. Except for effects due to the integration of CPU and channel-subsystem equipment, a CPU is busy for the duration of the execution of START SUBCHANNEL, which lasts until the addressed subchannel has been passed the ORB contents.

2. The subchannel is busy for a new START SUBCHANNEL from the receipt of the ORB contents until the primary interruption condition is cleared at the subchannel.
3. The I/O device is busy from the initiation of the first operation at the device until either the subchannel becomes suspended or the secondary interruption condition is placed at the subchannel. In the case of a suspended subchannel, the device again becomes busy when the execution of the suspended channel program is resumed.

Conclusion of I/O Operations

The conclusion of an I/O operation normally is indicated by two status conditions: channel end and device end. The channel-end condition indicates that the I/O device has received or provided all data associated with the operation and no longer needs channel-subsystem facilities. This condition is called the primary interruption condition, and the channel end in this case is the primary status. Generally, the primary interruption condition is any interruption condition that relates to an I/O operation and that signals the conclusion at the subchannel of the I/O operation or chain of I/O operations.

The device-end signal indicates that the I/O device has concluded execution and is ready to perform another operation. This condition is called the secondary interruption condition, and the device end in this case is the secondary status. Generally, the secondary interruption condition is any interruption condition that relates to an I/O operation and that signals the conclusion at the device of the I/O operation or chain of operations. The secondary interruption condition can occur concurrently with, or later than, the primary interruption condition.

Concurrent with the primary or secondary interruption conditions, both the channel subsystem and the I/O device can provide indications of unusual situations.

The conditions signaling the conclusion of an I/O operation can be brought to the attention of the program by I/O interruptions or, when the CPUs are disabled for I/O interruptions, by programmed interrogation of the channel subsystem. In the former case, these conditions cause storing of the I/O-interruption code, which contains information concerning the interrupting source. In the latter case, the inter-

ruption code is stored as a result of the execution of TEST PENDING INTERRUPTION.

When the primary interruption condition is recognized, the channel subsystem attempts to notify the program, by means of an interruption request, that a subchannel contains information describing the conclusion of an I/O operation at the subchannel. For command-mode interruptions, the information identifies the last CCW used and may provide its residual byte count, thus describing the extent of main storage used. For transport-mode interruptions, the information identifies the current TCW and the TSB associated with the channel program that contains additional status about the I/O operation, such as residual byte count. In addition to information about the channel program, both the channel subsystem and the I/O device may provide additional indications of unusual conditions as part of either the primary or the secondary interruption condition. The information contained at the subchannel may be stored by the execution of TEST SUBCHANNEL or the execution of STORE SUBCHANNEL. This information, when stored, is called a subchannel-status word (SCSW).

Chaining When Using a CCW Channel Program

When the ORB specifies a CCW channel program, facilities are provided for the program to initiate the execution of a chain of I/O operations with a single START SUBCHANNEL instruction. When the current CCW specifies command chaining and no unusual conditions have been detected during the operation, the receipt of the device-end signal causes the channel subsystem to fetch a new CCW. If the CCW passes certain validity tests and the suspend flag is not specified as a one in the new CCW, execution of a new command is initiated at the device. If the CCW fails to pass the validity tests, the new command is not initiated, command chaining is suppressed, and the status associated with the new CCW causes an interruption condition to be generated. If the suspend flag is specified as a one and this value is valid because of a one value in the suspend control, bit 4 of word 1 of the associated ORB, execution of the new command is not initiated, and command chaining is concluded.

Execution of the new command is initiated by the channel subsystem in the same way as in the previous operation. The ending signals occurring at the conclusion of an operation caused by a CCW specifying command chaining are not made available to

the program. When another I/O operation is initiated by command chaining, the channel subsystem continues execution of the channel program. If, however, an unusual condition has been detected, command chaining is suppressed, the channel program is terminated, an interruption condition is generated, and the ending signals causing the termination are made available to the program.

The suspend-and-resume function provides the program with control over the execution of a channel program. The initiation of the suspend function is controlled by the setting of the suspend-control bit in the ORB. The suspend function is signaled to the channel subsystem during channel-program execution when the suspend-control bit in the ORB is one and the suspend flag in the first CCW or in a CCW fetched during command chaining is one.

Suspension occurs when the channel subsystem fetches a CCW with the suspend flag validly (because of a one value of the suspend-control bit in the ORB) specified as one. The command in this CCW is not sent to the I/O device, and the device is signaled that the chain of commands is concluded. A subsequent RESUME SUBCHANNEL instruction informs the channel subsystem that the CCW that caused suspension may have been modified and that the channel subsystem must refetch the CCW and examine the current setting of the suspend flag. If the suspend flag is found to be zero in the CCW, the channel subsystem resumes execution of the chain of commands with the I/O device.

Chaining When Using a TCW Channel Program

When the ORB specifies a TCW channel program, facilities are also provided for the program to initiate the execution of a chain of device operations with a single START SUBCHANNEL instruction. Command chaining may be specified for those DCWs designated by a single TCW. When the current DCW specifies command chaining and no unusual conditions have been detected during the operation, recognition of the successful execution of the DCW causes the next DCW in the current TCCB to be processed.

If the next DCW passes certain validity tests, execution of a new command is initiated at the device and the DCW becomes the current DCW. If the DCW fails to pass the validity tests, the new command is not initiated, command chaining is suppressed, the channel program is terminated, and the status associated

with the new DCW causes an interruption condition to be generated.

Execution of the new command is initiated in the same way as in the previous operation. The ending signals occurring at the conclusion of an operation caused by a DCW that is not the last specified DCW are not made available to the program. When another I/O operation is initiated by command chaining, execution of the channel program continues. If, however, an unusual condition has been detected, command chaining is suppressed, the channel program is terminated, an interruption condition is generated, and status is made available to the program that identifies the unusual condition.

Premature Conclusion of I/O Operations

Channel-program execution may be terminated prematurely by CANCEL SUBCHANNEL, HALT SUBCHANNEL or CLEAR SUBCHANNEL. The execution of CANCEL SUBCHANNEL causes the channel subsystem to terminate the start function at the subchannel if the channel program has not been initiated at the device. When the start function is terminated by the execution of CANCEL SUBCHANNEL, the channel subsystem sets condition code 0 in response to the CANCEL SUBCHANNEL instruction. The execution of HALT SUBCHANNEL causes the channel subsystem to issue the halt signal to the I/O device and terminate channel-program execution at the subchannel. When channel-program execution is terminated by the execution of HALT SUBCHANNEL, the program is notified of the termination by means of an I/O-interruption request. When the subchannel is in command mode, the interruption request is generated when the device presents status for the terminated operation. When the subchannel is in transport mode, the interruption request is generated immediately. If, however, the halt signal was issued to the device during command chaining after the receipt of device end but before the next command was transferred to the device, the interruption request is generated after the device has been signaled. In the latter case, the device-status field of the SCSW will contain zeros. The execution of CLEAR SUBCHANNEL clears the subchannel of indications of the channel program in execution, causes the channel subsystem to issue the clear signal to the I/O device, and causes the channel subsystem to generate an I/O-interruption request to notify the program of the completion of the clear function.

I/O Interruptions

Conditions causing I/O-interruption requests are asynchronous to activity in CPUs, and more than one condition can occur at the same time. The conditions are preserved at the subchannels until cleared by TEST SUBCHANNEL or CLEAR SUBCHANNEL, or reset by an I/O-system reset.

When an I/O-interruption condition has been recognized by the channel subsystem and indicated at the subchannel, an I/O-interruption request is made pending for the I/O-interruption subclass specified at the subchannel. The I/O-interruption subclass for which the interruption is made pending is under programmed control through the use of MODIFY SUBCHANNEL. A pending I/O interruption may be accepted by any CPU that is enabled for interruptions from its I/O-interruption subclass. Each CPU has eight mask bits, in control register 6, that control the enablement of that CPU for each of the eight I/O-interruption subclasses, with the I/O mask, bit 6 in the PSW, being the master I/O-interruption mask for the CPU.

When an I/O interruption occurs at a CPU, the I/O-interruption code is stored in the I/O-communication area of that CPU, and the I/O-interruption request is cleared. The I/O-interruption code identifies the subchannel for which the interruption was pending. The conditions causing the generation of the interruption request may then be retrieved from the subchannel explicitly by TEST SUBCHANNEL or by STORE SUBCHANNEL.

A pending I/O-interruption request may also be cleared by TEST PENDING INTERRUPTION when the corresponding I/O-interruption subclass is enabled but the PSW has I/O interruptions disabled or by TEST SUBCHANNEL when the CPU is disabled for I/O interruptions from the corresponding I/O-interruption subclass. A pending I/O-interruption request may also be cleared by CLEAR SUBCHANNEL. Both CLEAR SUBCHANNEL and TEST SUBCHANNEL clear the preserved interruption condition at the subchannel as well.

Normally, unless the interruption request is cleared by CLEAR SUBCHANNEL, the program issues TEST SUBCHANNEL to obtain information concerning the execution of the operation.

Chapter 14. I/O Instructions

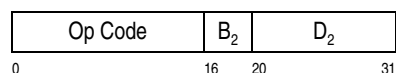
I/O-Instruction Formats	14-1	RESET CHANNEL PATH	14-9
I/O-Instruction Execution	14-2	RESUME SUBCHANNEL	14-10
Serialization	14-2	SET ADDRESS LIMIT	14-12
Operand Access	14-2	SET CHANNEL MONITOR	14-13
Condition Code	14-2	START SUBCHANNEL	14-15
Program Exceptions	14-2	STORE CHANNEL PATH STATUS	14-17
Instructions	14-2	STORE CHANNEL REPORT WORD	14-17
CANCEL SUBCHANNEL	14-4	STORE SUBCHANNEL	14-18
CLEAR SUBCHANNEL	14-5	TEST PENDING INTERRUPTION	14-19
HALT SUBCHANNEL	14-6	TEST SUBCHANNEL	14-20
MODIFY SUBCHANNEL	14-8		

All the I/O instructions described here are provided for the control of channel-subsystem operations. The I/O instructions are listed in Figure 14-1 on page 14-3. All of the I/O instructions are privileged instructions.

Several I/O instructions result in the channel subsystem being signaled to perform functions asynchronous to the execution of the instructions. The description of each instruction of this type contains a section, "Associated Functions," that summarizes the asynchronous functions.

I/O-Instruction Formats

I/O instructions use the S format:

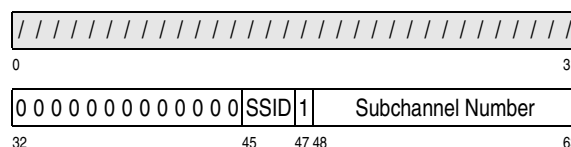


The use of the second-operand address and general registers 1 and 2 (as implied operands) depends on the I/O instruction. Figure 14-1 on page 14-3 defines which operands are used to execute each I/O instruction. In addition, detailed information regarding operand usage appears in the description of each I/O instruction.

All I/O instructions that reference a subchannel use the contents of general register 1 as an implied oper-

and. For these I/O instructions, general register 1 contains the subsystem-identification word (SID).

When the multiple-subchannel set (MSS) facility is installed, the format of the subsystem-identification word is as follows:

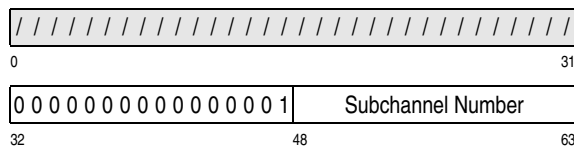


Bit positions 0-31 are ignored. Bit positions 32-44 must contain zeros and bit 47 must contain 1. (Exception conditions are listed later in the section.)

Bit positions 45-46 of general register 1 contain the subchannel-set identifier (SSID) uniquely identifying a subchannel set (SS). When the multiple-subchannel-set facility is not installed, the SSID must be equal to zero. Bit positions 48-63 of general register 1 contain the binary number of the subchannel within the specified subchannel set to be used for the function specified by the instruction.

(See “Multiple-Subchannel-Set Facility” on page 17-29 for more information on SSID.)

The format of the subsystem-identification word when the MSS facility is not installed is as follows:



Bit positions 48-63 of general register 1 contain the binary number of the subchannel to be used for the function specified by the instruction. Bit positions 0-31 of general register 1 are ignored and bits 32-47 specify the binary number one.

I/O-Instruction Execution

Serialization

The execution of any I/O instruction causes a serialization and checkpoint synchronization function to occur. For a definition of the serialization of CPU operations, see “CPU Serialization” on page 5-100.

Operand Access

During the execution of an I/O instruction, the order in which fields of the operand and fields of the subchannel, if applicable, are accessed is unpredictable. It is also unpredictable whether fetch accesses are made to fields of an operand or the subchannel, as applicable, when those fields are not needed to complete the execution of the I/O instruction. (See “Relation between Operand Accesses” on page 5-98.)

Condition Code

During the execution of some I/O instructions, the results of certain tests are used to set one of four condition codes in the PSW. The I/O instructions for which execution can result in the setting of the condition code are listed in Figure 14-1 on page 14-3. The condition code indicates the result of the execution of the I/O instruction. The general meaning of the condition code for I/O instructions is given below; the meaning of the condition code for a specific instruction appears in the description of that instruction.

Condition Code 0: Instruction execution produced the expected or most probable result. (See “Deferred Condition Code (CC)” on page 16-9 for a description

of conditions that can be encountered subsequent to the presentation of condition code 0 that result in a nonzero deferred condition code.)

Condition Code 1: Instruction execution produced the alternate or second-most-probable result, or status conditions were present that may or may not have prevented the expected result.

Condition Code 2: Instruction execution was ineffective because the designated subchannel or channel-subsystem facility was busy with a previously initiated function.

Condition Code 3: Instruction execution was ineffective because the designated element was not operational or because some condition precluded initiation of the normal function.

In situations where conditions exist that could cause more than one nonzero condition code to be set, the priority of the condition codes is as follows:

Condition code 3 has precedence over condition codes 1 and 2.

Condition code 1 has precedence over condition code 2.

Program Exceptions

The program exceptions that the I/O instructions can encounter are access, operand, privileged-operation, and specification exceptions. Figure 14-1 on page 14-3 shows the exceptions that are applicable to each of the I/O instructions. The execution of the instruction is suppressed for privileged-operation, operand, and specification exceptions. Except as indicated otherwise in the section “Special Conditions” for each instruction, the instruction ending for access exceptions is as described in “Recognition of Access Exceptions” on page 6-40).

Instructions

The mnemonics, format, and operation codes of the I/O instructions are given in Figure 14-1 on page 14-3. The figure also indicates the conditions that can cause a program interruption and whether the condition code is set.

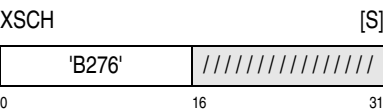
In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For START SUBCHANNEL, for

example, SSCH is the mnemonic and $D_2(B_2)$ the operand designation.

Name	Mnemonic	Characteristics							Op-code	Page
CANCEL SUBCHANNEL	XSCH	S	C	P	OP	¢	GS		B276	14-4
CLEAR SUBCHANNEL	CSCH	S	C	P	OP	¢	GS		B230	14-5
HALT SUBCHANNEL	HSCH	S	C	P	OP	¢	GS		B231	14-6
MODIFY SUBCHANNEL	MSCH	S	C	P A SP	OP	¢	GS	B_2	B232	14-8
RESET CHANNEL PATH	RCHP	S	C	P					B23B	14-9
RESUME SUBCHANNEL	RSCH	S	C	P	OP	¢	GS		B238	14-10
SET ADDRESS LIMIT	SAL	S		P	OP	¢	G1		B237	14-12
SET CHANNEL MONITOR	SCHM	S		P	OP	¢	GM		B23C	14-13
START SUBCHANNEL	SSCH	S	C	P A SP	OP	¢	GS	B_2	B233	14-15
STORE CHANNEL PATH STATUS	STCPS	S		P A SP		¢		ST B_2	B23A	14-17
STORE CHANNEL REPORT WORD	STCRW	S	C	P A SP		¢		ST B_2	B239	14-17
STORE SUBCHANNEL	STSCH	S	C	P A SP	OP	¢	GS	ST B_2	B234	14-18
TEST PENDING INTERRUPTION	TPI	S	C	P A ¹ SP		¢		ST B_2	B236	14-19
TEST SUBCHANNEL	TSCH	S	C	P A SP	OP	¢	GS	ST B_2	B235	14-20
Explanation: ¢ Causes a serialization and checkpoint synchronization function. A Access exceptions for logical addresses. A ¹ When the effective address is zero, it is not used to access storage, and no access exceptions can occur, except that access exceptions may occur during access-register translation. B_2 B_2 field designates an access register in the access-register mode. C Condition code is set. G1 Instruction execution includes implied use of general register 1 as a parameter. GM Instruction execution includes the implied use of multiple general registers. GS Instruction execution includes the implied use of general register 1 as the subsystem-identification word. OP Operand exception. P Privileged-operation exception. S S instruction format. SP Specification exception. ST PER storage-alteration event.										

Figure 14-1. Summary of I/O Instructions

CANCEL SUBCHANNEL



The current start function, if any, is terminated at the designated subchannel if CANCEL SUBCHANNEL is applicable.

General register 1 contains a subsystem-identification word that designates the subchannel for which the current START FUNCTION, if any, is to be terminated.

If the subchannel (1) is not subchannel active, (2) is start pending, resume pending, or suspended, and (3) is performing only the start function, then the start function at the subchannel is terminated, and the subchannel is made no longer start pending, resume pending, or suspended, as appropriate. In addition, internal indications of busy are reset for the subchannel.

Condition code 0 is set to indicate that the actions described above have been taken.

If an invalid ORB field or a no-path-available condition is present for a previously initiated start function and the condition was not reported during the execution of START SUBCHANNEL, condition code 0 may be indicated for CANCEL SUBCHANNEL provided that the subchannel is not yet status pending to report the error condition; if condition code 0 is presented, no subsequent status is generated to indicate the error condition.

During the execution of CANCEL SUBCHANNEL for a transport-mode operation, the channel subsystem may be signaled to asynchronously perform the interrogate function.

Special Conditions

Condition code 1 is set, and no other action is taken, when the subchannel is status pending with any status.

Condition code 2 is set, and no other action is taken, when CANCEL SUBCHANNEL is not applicable and the subchannel is not status pending, or when conditions exist that prevent immediate determination of internal conditions for the subchannel

and the CPU has determined to end the instruction. CANCEL SUBCHANNEL is not applicable when the subchannel (1) has no function specified, (2) has a function other than the start function alone specified, (3) is not resume pending, is not start pending, and is not suspended, or (4) is subchannel active.

When the subchannel is operating in transport mode and condition code 2 is set, the CPU may signal the channel subsystem to asynchronously perform the interrogate function, and end the instruction.

Condition code 3 is set, and no other action is taken, when the subchannel is not operational for CANCEL SUBCHANNEL. A subchannel is not operational for CANCEL SUBCHANNEL when the subchannel is not provided by the channel subsystem, has no valid device number assigned to it, or is not enabled.

CANCEL SUBCHANNEL can encounter the program exceptions described or listed below.

When the multiple-subchannel-set facility is not installed, bits 32-47 of general register 1 must contain 0001 hex; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is installed, bits 32-44 of general register 1 must contain zeros, bits 45-46 must contain a valid value, and bit 47 must contain the value one; otherwise, an operand exception is recognized.

Resulting Condition Code:

- 0 Start function canceled
- 1 Status pending
- 2 CANCEL SUBCHANNEL not applicable
- 3 Not operational

Program Exceptions:

- Operand
- Privileged operation

Programming Notes:

1. The actions taken by CANCEL SUBCHANNEL are completed during the execution of the instruction.

If condition code 0 is presented, there is no subsequent I/O interruption resulting from the termi-

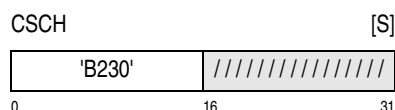
nated I/O operation. However, the device may have signaled a busy condition while the canceled operation was start pending. In this case, the device owes a no-longer-busy signal to the channel subsystem. This may result in unsolicited device-end status before the next operation is initiated at the device.

Condition code 2 may be presented to indicate that the CPU has determined to end the instruction because the internal conditions of the specified subchannel cannot be immediately determined. Subsequent execution of STORE SUBCHANNEL may store an SCSW indicating that CANCEL SUBCHANNEL is applicable (i.e., the subchannel is performing the start function alone, is start pending, resume pending, or suspended, and is not subchannel-active). In such a case, the program can either recognize that the I/O operation has not been terminated and allow the operation to start or the program can retry CANCEL SUBCHANNEL in an attempt to terminate the start function. However, retrying CANCEL SUBCHANNEL does not guarantee that the instruction will end with condition code 0.

In transport mode, condition code 2 may be presented to indicate that the CPU has determined to asynchronously signal the channel subsystem to perform the interrogate function, and end the instruction. The LPUM must contain a nonzero value for the interrogate to be initiated.

2. Upon the completion of CANCEL SUBCHANNEL with condition code 0, the subchannel is ready to accept a new start function initiated by START SUBCHANNEL.

CLEAR SUBCHANNEL



The designated subchannel is cleared, the current start or halt function, if any, is terminated at the designated subchannel, and the channel subsystem is signaled to asynchronously perform the clear function at the designated subchannel and at the associated device.

General register 1 contains a subsystem-identification word (SID) that designates the subchannel to be cleared.

If a start or halt function is in progress, it is terminated at the subchannel.

The subchannel is made no longer status pending. All activity, as indicated in the activity-control field of the SCSW, is cleared at the subchannel, except that the subchannel is made clear pending. Any functions in progress, as indicated in the function-control field of the SCSW, are cleared at the subchannel, except for the clear function that is to be performed because of the execution of this instruction.

When the subchannel is operating in transport mode and condition code 2 is set, the CPU may signal the channel subsystem to asynchronously perform the interrogate function, and end the instruction.

The channel subsystem is signaled to asynchronously perform the clear function. The clear function is summarized below in the section "Associated Functions" and is described in detail in "Clear Function" on page 15-14.

Condition code 0 is set to indicate that the actions described above have been taken.

Associated Functions

Subsequent to the execution of CLEAR SUBCHANNEL, the channel subsystem asynchronously performs the clear function. If conditions allow, the channel subsystem chooses a channel path and attempts to issue the clear signal to the device to terminate the I/O operation, if any. The subchannel then becomes status pending. Conditions encountered by the channel subsystem that preclude issuing the clear signal to the device do not prevent the subchannel from becoming status pending (see "Clear Function" on page 15-14).

When the subchannel becomes status pending as a result of performing the clear function, data transfer, if any, with the associated device has been terminated. The SCSW stored when the resulting status is cleared by TEST SUBCHANNEL has the clear-function bit stored as one. If the channel subsystem can determine that the clear signal was issued to the device, the clear-pending bit is stored as zero in the SCSW. Otherwise, the clear-pending bit is stored as one, and other indications are provided that describe

in greater detail the condition that was encountered. (See “*Interruption-Response Block*” on page 16-6.)

Measurement data is not accumulated, and device-connect time is not stored in the extended-status word for the subchannel, for a start function that is terminated by CLEAR SUBCHANNEL.

Special Conditions

Condition code 3 is set, and no other action is taken, when the subchannel is not operational for CLEAR SUBCHANNEL. A subchannel is not operational for CLEAR SUBCHANNEL when the subchannel is not provided in the channel subsystem, has no valid device number assigned to it, or is not enabled.

CLEAR SUBCHANNEL can encounter the program exceptions described or listed below.

When the multiple-subchannel-set facility is not installed, bits 32-47 of general register 1 must contain 0001 hex; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is installed, bits 32-44 of general register 1 must contain zeros, bits 45-46 must contain a valid value, and bit 47 must contain the value one; otherwise, an operand exception is recognized.

Resulting Condition Code:

- 0 Function initiated
- 1 —
- 2 —
- 3 Not operational

Program Exceptions:

- Operand
- Privileged operation

HALT SUBCHANNEL



The current start function, if any, is terminated at the designated subchannel, and the channel subsystem is signaled to asynchronously perform the halt func-

tion at the designated subchannel and at the associated device.

General register 1 contains a subsystem-identification word that designates the subchannel to be halted.

If a start function is in progress, it is terminated at the subchannel.

The subchannel is made halt pending, and the halt function is indicated at the subchannel.

When HALT SUBCHANNEL is executed and the designated subchannel is subchannel-and-device active and status pending with intermediate status, the status-pending indication is eliminated (see the discussion of bits 24, 25, and 28 in “*Activity Control (AC)*” on page 16-14). The status-pending condition is reestablished as part of the halt function (see the section “*Associated Functions*” below).

The channel subsystem is signaled to asynchronously perform the halt function. The halt function is summarized below in the section “*Associated Functions*” and is described in detail in “*Halt Function*” on page 15-16.

Condition code 0 is set to indicate that the actions described above have been taken.

Associated Functions

Subsequent to the execution of HALT SUBCHANNEL, the channel subsystem asynchronously performs the halt function. If conditions allow, the channel subsystem chooses a channel path and attempts to issue the halt signal to the device to terminate the I/O operation, if any. The subchannel then becomes status pending.

When the subchannel becomes status pending as a result of performing the halt function, data transfer, if any, with the associated device has been terminated. The SCSW stored when the resulting status is cleared by TEST SUBCHANNEL has the halt-function bit stored as one. If the halt signal was issued to the device, the halt-pending bit is stored as zero. Otherwise, the halt-pending bit is stored as one, and other indications are provided that describe in greater detail the condition that was encountered. (See “*Interruption-Response Block*” on page 16-6. and “*Halt Function*” on page 15-16.)

On some models, path availability is tested as part of the halt function instead of as part of the execution of the instruction. In these models, when no channel path is available for selection, the halt signal is not issued, and the subchannel is made status pending. When the status-pending condition is subsequently cleared by TEST SUBCHANNEL, the halt-pending bit is stored as one in the SCSW.

If a status-pending condition is eliminated during the execution of HALT SUBCHANNEL, then this condition is reestablished along with the other status conditions when the completion of the halt function is indicated to the program.

The halt-pending condition may not be recognized by the channel subsystem if a status-pending condition has been generated. This situation could occur, for example, when alert status is presented or generated while the subchannel is already start pending or resume pending, or when primary status is presented during the attempt to initiate the I/O operation for the first command as specified by the start function or implied by the resume function. If recognition of the status-pending condition by the channel subsystem has occurred logically prior to recognition of the halt-pending condition, the SCSW, when cleared by TEST SUBCHANNEL, has the halt-pending bit stored as one.

If measurement data is being accumulated when a start function is terminated by HALT SUBCHANNEL, the measurement data continues to be accumulated for the subchannel and reflects the extent of subchannel and device usage required, if any, while performing the currently terminated start function. The measurement data, if any, is accumulated in the measurement block for the subchannel or placed in the extended-status word, as appropriate, when the subchannel becomes status-pending with primary or secondary status. (See “Channel-Subsystem Monitoring” on page 17-1.)

Special Conditions

Condition code 1 is set, and no other action is taken, when the subchannel is status pending alone or is status pending with any combination of alert, primary, or secondary status.

Condition code 2 is set, and no other action is taken, when the subchannel is busy for HALT SUB-

CHANNEL. The subchannel is busy for HALT SUBCHANNEL when a halt function or clear function is already in progress at the subchannel.

Condition code 3 is set, and no other action is taken, when the subchannel is not operational for HALT SUBCHANNEL. A subchannel is not operational for HALT SUBCHANNEL when the subchannel is not provided in the channel subsystem, has no valid device number assigned to it, or is not enabled. On some models, a subchannel is also not operational for HALT SUBCHANNEL when no channel path is available for selection by the device. (See “Channel-Path Availability” on page 15-13 for a description of channel paths that are available for selection.)

HALT SUBCHANNEL can encounter the program exceptions described or listed below.

When the multiple-subchannel-set facility is not installed, bits 32-47 of general register 1 must contain 0001 hex; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is installed, bits 32-44 of general register 1 must contain zeros, bits 45-46 must contain a valid value, and bit 47 must contain the value one; otherwise, an operand exception is recognized.

Resulting Condition Code:

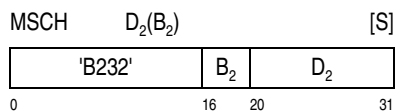
- 0 Function initiated
- 1 Status pending with other than intermediate status
- 2 Busy
- 3 Not operational

Program Exceptions:

- Operand
- Privileged operation

Programming Note: After the execution of HALT SUBCHANNEL, the status-pending condition indicating the completion of the halt function may be delayed for an extended period of time, for example, when the device is a magnetic-tape unit executing a rewind command.

MODIFY SUBCHANNEL



The information contained in the subchannel-information block (SCHIB) is placed in the program-modifiable fields at the subchannel. As a result, the program influences, for that subchannel, certain aspects of I/O processing relative to the clear, halt, resume, and start functions and certain I/O support functions.

General register 1 contains a subsystem-identification word (SID) that designates the subchannel that is to be modified as specified by certain fields of the SCHIB. The second-operand address is the logical address of the SCHIB and must be designated on a word boundary; otherwise, a specification exception is recognized.

The channel-subsystem operations that may be influenced due to placement of SCHIB information in the subchannel are:

- I/O processing (E field)
- Interruption processing (interruption parameter and ISC field)
- Path management (D, LPM, and POM fields)
- Monitoring and address-limit checking (measurement-block index, LM, and MM fields)
- Measurement-block-format control (F field)
- Extended-measurement-word-mode enable (X field)
- Concurrent-sense facility (S field)
- Measurement-block address (MBA)

Bits 0, 1, 6, and 7 of word 1, and bits 0-28 of word 6 of the SCHIB operand must be zeros, and bits 9 and 10 of word 1 must not both be ones. When the extended-I/O-measurement-block facility is installed and a format-1 measurement block is specified, bits 26-31 of word 11 must be specified as zeros. When the extended-I/O-measurement-block facility is not installed, bit 29 of word 6 must be specified as zero; otherwise, an operand exception is recognized. When the extended-I/O-measurement-word facility is not installed, or is installed but not enabled, bit 30 of word 6 must be specified as zero; otherwise, an operand exception is recognized. The remaining fields of the SCHIB are ignored and do not affect the processing of MODIFY SUBCHANNEL. (For further

details, see “Subchannel-Information Block” on page 15-2.)

Condition code 0 is set to indicate that the information from the SCHIB has been placed in the program-modifiable fields at the subchannel, except that, when the device-number-valid bit (V) at the designated subchannel is zero, then condition code 0 is set, and the information from the SCHIB is not placed in the program-modifiable fields.

Special Conditions

Condition code 1 is set, and no other action is taken, when the subchannel is status pending. (See “Status Control (SC)” on page 16-17.)

Condition code 2 is set, and no other action is taken, when a clear, halt, or start function is in progress at the subchannel. (See “Function Control (FC)” on page 16-13.)

Condition code 3 is set, and no other action is taken, when the subchannel is not operational for MODIFY SUBCHANNEL. A subchannel is not operational for MODIFY SUBCHANNEL when the subchannel is not provided in the channel subsystem.

MODIFY SUBCHANNEL can encounter the program exceptions described or listed below.

In word 1 of the SCHIB, bits 0, 1, 6, and 7 must be zeros and, when the address-limit-checking facility is installed, bits 9 and 10 must not both be ones. In word 6 of the SCHIB, bits 0-28 must be zeros. Otherwise an operand exception is recognized.

When the extended-I/O-measurement-block facility is installed and a format-1 measurement block is specified, bits 26-31 of word 11 must be specified as zeros; otherwise, an operand exception is recognized. When the extended-I/O-measurement-block facility is not installed, bit 29 of word 6 must be specified as zero; otherwise, an operand exception is recognized. When the extended-I/O-measurement-word facility is not installed, or is installed but not enabled, bit 30 of word 6 must be specified as zero; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is not installed, bits 32-47 of general register 1 must contain 0001 hex; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is installed, bits 32-44 of general register 1 must contain zeros, bits 45-46 must contain a valid value, and bit 47 must contain the value one; otherwise, an operand exception is recognized.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The execution of MODIFY SUBCHANNEL is suppressed on all addressing and protection exceptions.

Resulting Condition Code:

- 0 Function completed
- 1 Status pending
- 2 Busy
- 3 Not operational

Program Exceptions:

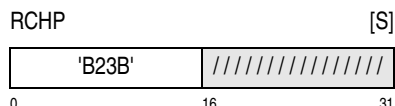
- Access (fetch, operand 2)
- Operand
- Privileged operation
- Specification

Programming Notes:

1. If a device signals I/O-error alert while the associated subchannel is disabled, the channel subsystem issues the clear signal to the device and discards the I/O-error-alert indication without generating an I/O-interruption condition.
2. If a device presents unsolicited status while the associated subchannel is disabled, that status is discarded by the channel subsystem without generating an I/O-interruption condition. However, if the status presented contains unit check, the channel subsystem issues the clear signal for the associated subchannel and does not generate an I/O-interruption condition. This should be taken into account when the program uses MODIFY SUBCHANNEL to enable a subchannel. For example, the medium on the associated device that was present when the subchannel became disabled may have been replaced, and, therefore, the program should verify the integrity of that medium.
3. It is recommended that the program inspect the contents of the subchannel by subsequently

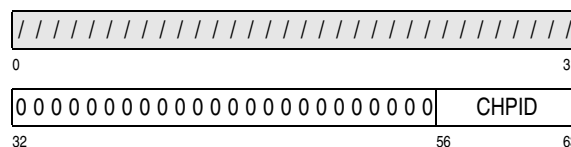
issuing STORE SUBCHANNEL when MODIFY SUBCHANNEL sets condition code 0. Use of STORE SUBCHANNEL is a method for determining if the designated subchannel was changed or not. Failure to inspect the subchannel following the setting of condition code 0 by MODIFY SUBCHANNEL may result in conditions that the program does not expect to occur.

RESET CHANNEL PATH



The channel-path-reset facility is signaled to perform the channel-path-reset function on the channel path designated by the contents of general register 1.

The format of general register 1 is as follows:



Channel-Path Identifier (CHPID): Bit positions 56-63 of general register 1 contain an unsigned binary integer that designates the channel path on which the channel-path-reset function is to be performed.

Bit positions in general register 1 that are shown as zeros, are reserved and must contain zeros; otherwise, an operand exception is recognized. Bit positions 0-31 of general register 1 are ignored.

If conditions allow, the channel-path-reset facility is signaled to asynchronously perform the channel-path-reset function on the designated channel path. The channel-path-reset function is summarized below in the section “Associated Functions” and is described in detail in “Channel-Path Reset” on page 17-12.

Condition code 0 is set to indicate that the channel-path-reset facility has been signaled.

Associated Functions

Subsequent to the execution of RESET CHANNEL PATH, the channel-path-reset facility asynchronously performs the channel-path-reset function. Certain indications are reset at all subchannels that have access to the designated channel path, and the reset signal is issued on that channel path. Any I/O functions in progress at the devices are reset, but only for the channel path on which the reset signal is received. An I/O operation or chain of I/O operations taking place in the multipath mode may be able to continue to be executed on other channel paths in the multipath group, if any. (See “Channel-Path-Reset Function” on page 15-75.)

The result of performing the channel-path-reset function on the designated channel path is communicated to the program by means of a channel report (see “Channel Report” on page 17-25).

Special Conditions

Condition code 2 is set, and no other action is taken, when, on some models, the channel-path-reset facility is busy performing the channel-path-reset function for a previous execution of the RESET CHANNEL PATH instruction.

Condition code 3 is set, and no other action is taken, when, on some models, the designated channel path is not operational for the execution of RESET CHANNEL PATH. On these models, the channel path is not operational for the execution of RESET CHANNEL PATH when the designated channel path is not physically available.

If the channel-path-reset facility is busy and the designated channel path is not physically available, it depends on the model whether condition code 2 or 3 is set.

RESET CHANNEL PATH can encounter the program exceptions described or listed below.

Bit positions 32-55 of general register 1 must contain zeros; otherwise, an operand exception is recognized. Bit positions 0-31 of general register 1 are ignored.

Resulting Condition Code:

- | | |
|---|--------------------|
| 0 | Function initiated |
| 1 | — |

- | | |
|---|-----------------|
| 2 | Busy |
| 3 | Not operational |

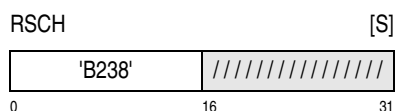
Program Exceptions:

- Operand
- Privileged operation

Programming Notes:

1. To eliminate the possibility of a data-integrity exposure for devices that have the capability of generating unsolicited device-end status, I/O operations in progress with such devices on the channel path for which RESET CHANNEL PATH is to be executed must be terminated by the execution of either HALT SUBCHANNEL or CLEAR SUBCHANNEL. Otherwise, subsequent to receiving the reset signal, the device may present an unsolicited device end that may be interpreted by the channel subsystem as a solicited device end and cause command chaining to occur.
2. If the status-verification facility is being used and RESET CHANNEL PATH is executed without first stopping all ongoing operations associated with the channel path being reset, erroneous device-status-check conditions may be detected

RESUME SUBCHANNEL



The channel subsystem is signaled to perform the resume function at the designated subchannel.

General register 1 contains a subsystem-identification word that designates the subchannel at which the resume function is to be performed.

The subchannel is made resume pending.

Logically prior to the setting of condition code 0 and only if the subchannel is currently in the suspended state, path-not-operational conditions at the subchannel, if any, are cleared.

The channel subsystem is signaled to asynchronously perform the resume function. The resume

function is summarized below in the section “Associated Functions” and is described in detail in “Start Function and Resume Function” on page 15-20

Condition code 0 is set to indicate that the actions described above have been taken.

Associated Functions

Subsequent to the execution of RESUME SUBCHANNEL, the channel subsystem asynchronously performs the resume function. Except when the subchannel is subchannel active, if the execution of RESUME SUBCHANNEL results in the setting of condition code 0, performance of the resume function causes execution of a currently suspended channel program to be resumed with the associated device, provided that the suspend flag for the current CCW has been set to zero by the program. If the suspend flag remains one, execution of the channel program remains suspended. But, if the subchannel is subchannel active at the time the execution of RESUME SUBCHANNEL results in the setting of condition code 0, then it is unpredictable whether execution of the current program is resumed or whether it is found by the resume function that the subchannel has become suspended in the interim. The subchannel is found to be suspended by the resume function only if the subchannel is status pending with intermediate status when the resume-pending condition is recognized by the channel subsystem. (See “Start Function and Resume Function” on page 15-20.)

Special Conditions

Condition code 1 is set, and no other action is taken, when the subchannel is status pending.

Condition code 2 is set, and no other action is taken, when the resume function is not applicable. The resume function is not applicable when the subchannel (1) has any function other than the start function alone specified, (2) has no function specified, (3) is resume pending, or (4) does not have suspend control specified for the start function in progress.

Condition code 3 is set, and no other action is taken, when the subchannel is not operational for the resume function. A subchannel is not operational for the resume function if the subchannel is not provided in the channel subsystem, has no valid device number assigned to it, or is not enabled.

RESUME SUBCHANNEL can encounter the program exceptions described or listed below.

When the multiple-subchannel-set facility is not installed, bits 32-47 of general register 1 must contain 0001 hex; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is installed, bits 32-44 of general register 1 must contain zeros, bits 45-46 must contain a valid value, and bit 47 must contain the value one; otherwise, an operand exception is recognized.

Resulting Condition Code:

- 0 Function initiated
- 1 Status pending
- 2 Function not applicable
- 3 Not operational

Program Exceptions:

- Operand
- Privileged operation

Programming Notes:

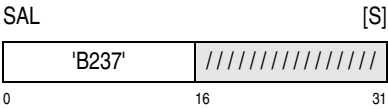
1. When channel-program execution is resumed from the suspended state, the device views the resumption as the beginning of a new chain of commands. When the suspension of channel-program execution occurs and the device requires that certain commands be first or appear only once in a chain of commands (for example, direct-access-storage devices), the program must ensure that the appropriate commands in the proper sequence are fetched by the channel subsystem after channel-program execution is resumed. One way the program can ensure proper sequencing of commands at the device is by allowing the I/O interruption to occur for an intermediate interruption condition due to suspension.

It is not reliable to notify the program that the subchannel is suspended by using the PCI flag in the CCW that contains the S flag because the PCI I/O interruption may occur before the subchannel is suspended. The SCSW would indicate that an I/O operation is in progress at the subchannel and device in this case.

The suspend flag of the target CCW should be set to zero before RESUME SUBCHANNEL is executed; otherwise, it is possible that the resume-pending condition may be recognized and the CCW refetched while the suspend flag is still one, in which case the resume-pending condition would be reset, and the execution of the channel program would be suspended. If the suspend flag of the target CCW is set to zero before the execution of RESUME SUBCHANNEL, the channel program is not suspended, provided that the subchannel is not subchannel active at the time the execution of RESUME SUBCHANNEL results in the setting of condition code 0. If condition code 0 is set while the subchannel is still subchannel active, it is unpredictable whether the resume-pending condition is recognized by the channel subsystem or whether it is found by the resume function that the subchannel has become suspended in the interim. The subchannel is found to be suspended by the resume function only if the subchannel is status pending with intermediate status at the time the resume-pending condition is recognized. When the subchannel is suspended, the execution of TEST SUBCHANNEL, which clears the intermediate interruption condition, also clears the indication of resume pending.

- 2. Some models recognize a resume-pending condition only after a CCW having an S flag validly set to one is fetched. Therefore, if a subchannel is resume pending and, during the execution of the channel program, no CCW is fetched having an S flag validly set to one, the subchannel remains resume pending until the primary interruption condition is cleared by TEST SUBCHANNEL.
- 3. Path availability is not tested during the execution of RESUME SUBCHANNEL. Instead, path availability is tested when the channel subsystem begins performance of the resume function.
- 4. The contents of the CCW fetched during performance of the resume function may be different from the contents of the same CCW when it was previously fetched and contained an S flag validly set to one.

SET ADDRESS LIMIT

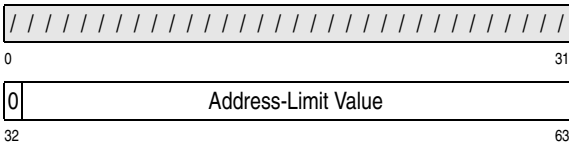


The address-limit-checking facility is signaled to use the specified address as the address-limit value, and the specified address is passed to the facility. Depending on the model, this instruction may not be provided. When this instruction is not provided, it is checked for operand exception and privileged-operation exception, and then is suppressed.

When the address-limit-checking facility is installed, the SET ADDRESS LIMIT instruction is available for use. When the address-limit-checking facility is not installed, or the FCX facility is installed, the SET ADDRESS LIMIT instruction is not provided.

General register 1 contains the absolute address to be used as the address-limit value. The specified address must be on a 64K-byte boundary and may designate a maximum absolute storage address of 2,147,418,112 (7FFF0000 hex) regardless of whether the CPU is operating in the 24-bit, 31-bit, or 64-bit addressing mode. Bits 0-31 of general register 1 are ignored, and bit 32 must be zero.

General register 1 has the following format:



Associated Functions

The value that is used by the address-limit-checking facility when determining whether to permit or prohibit a data access is called the address-limit value. The initial address-limit value is zero. The initial address-limit value is used by the address-limit-checking facility until the facility recognizes a signal, caused by the execution of SET ADDRESS LIMIT, to use a specified address. The recognition of this specified address as the new address-limit value occurs asynchronously with respect to the execution of SET ADDRESS LIMIT.

If address-limit checking is specified for a subchannel, then whether the specified address is used by the address-limit-checking facility, when determining whether to permit or prohibit a data access, depends on whether SET ADDRESS LIMIT was executed before, during, or after the execution of START SUBCHANNEL for that subchannel. If SET ADDRESS LIMIT is executed before START SUBCHANNEL, the specified address is used by the address-limit-checking facility. If SET ADDRESS LIMIT is executed during or after the execution of START SUBCHANNEL, it is unpredictable whether the specified address is used by the address-limit-checking facility for that particular start function. For a description of the manner in which address-limit checking is performed, see “Address-Limit Checking” on page 17-22.

Special Conditions

SET ADDRESS LIMIT can encounter the program exceptions described or listed below.

The address in general register 1 must be designated on a 64K byte boundary, and bit 32 of general register 1 must be zero; otherwise, an operand exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

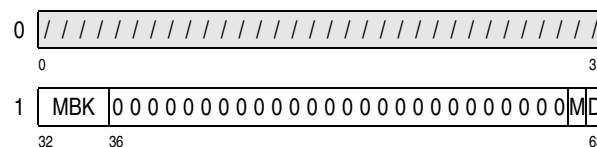
- Operand
- Privileged operation

SET CHANNEL MONITOR



Each of the measurement-block-update mode and device-connect-time-measurement mode of the channel subsystem is made either active or inactive, depending on the values of the measurement-mode-control bits in general register 1. If the measurement-mode-control bit for measurement-block update is one, the measurement-block origin and measurement-block key are passed to the channel subsystem.

General register 1 has the following format:



Ignored: Bit positions 0-31 of general register 1 are ignored.

Measurement-Block Key (MBK): Bit positions 32-35 of general register 1 contain the measurement-block key. When bit 62 is one, MBK specifies the access key that is to be used by the channel subsystem when it accesses the measurement-block area and, when the extended-I/O-measurement-block facility is installed, to access format-1 measurement blocks. Otherwise, MBK is ignored.

Measurement-Block-Update Control (M): Bit 62 of general register 1 is the measurement-mode-control bit that controls the measurement-block-update mode. When bit 62 of general register 1 is one and conditions allow, the measurement-block-update facility is signaled to asynchronously make the measurement-block-update mode active. In addition, the measurement-block-origin (MBO) address in general register 2 and the measurement-block key (MBK) in general register 1 are passed to the measurement-block-update facility.

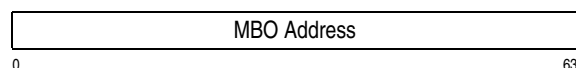
The measurement-block origin is used to determine the location of format-0 measurement blocks; the address of format-1 measurement blocks is stored at the subchannel using MODIFY SUBCHANNEL. The measurement-block key is used to access both format-0 and format-1 measurement blocks. The asynchronous functions that are performed by the measurement-block-update facility are summarized below in the section “Associated Functions” and are described in detail in “Channel-Subsystem Monitoring” on page 17-1.

When bit 62 of general register 1 is zero and conditions allow, the measurement-block-update mode is made inactive if it is active or remains inactive if it is inactive. The contents of bit positions 32-35 (MBK) of general register 1 and the contents of general register 2 are ignored.

Device-Connect-Time-Measurement Control (D): Bit 63 of general register 1 is the measurement-mode-control bit (D). When bit 63 is one and conditions allow, the device-connect-time-measurement mode is made active if it is inactive or remains active if it is active. When bit 63 is zero and conditions allow, the device-connect-time-measurement mode is made inactive if it is active or remains inactive if it is inactive.

Bit positions 36-61 of general register 1 are reserved and must contain zeros; otherwise, an operand exception is recognized.

General register 2 has the following format:



Measurement-Block-Origin (MBO) Address:

When bit 62 (M) of general register 1 is one, bit positions 0-63 of general register 2 contain the absolute address of the measurement-block origin (MBO), which is the beginning of the measurement-block area. The MBO address is used by the channel subsystem to locate format-0 measurement blocks. The origin of the measurement-block area must be designated on a 32-byte boundary otherwise, an operand exception is recognized. When bit 62 of general register 1 is zero, the contents of general register 2 are ignored.

If the channel-subsystem timer that is used by the channel-subsystem-monitoring facilities is in the error state, the state is reset. This happens independent of the setting of the two measurement-mode-control bits. (See “Channel-Subsystem Timing” on page 17-2 for a description of the timing facilities.)

Associated Functions

When the measurement-block-update facility is signaled (by means of SET CHANNEL MONITOR) to make the measurement-block-update mode active, the functions that are performed by the facility depend on whether or not the mode is already active when the signal is generated.

If the measurement-block-update mode is inactive when the signal is generated, the mode remains inactive until the measurement-block-update facility

recognizes the signal. When the measurement-block-update facility recognizes the signal, the measurement-block-update mode is made active, and the MBK that was passed when the signal was generated is used to access all measurement blocks, and the MBO that was passed when the signal was generated is used to determine the address of format-0 measurement blocks.

If the measurement-block-update mode is active when the signal is generated, the mode remains active, and the MBK and MBO associated with the execution of a previous SET CHANNEL MONITOR instruction continue to be used to control the storing of measurement data until the measurement-block-update facility recognizes the signal. When the measurement-block-update facility recognizes the signal, the MBK and MBO associated with that signal are used instead of the MBK and MBO associated with the execution of a previous SET CHANNEL MONITOR instruction. The SET CHANNEL MONITOR instruction does not affect the measurement-block address used for format-1 measurement blocks, but the MBK associated with the signal becomes the key used to access the measurement block.

In all above cases, the measurement-block-update facility recognizes the signal during, or subsequent to, the execution of the SET CHANNEL MONITOR instruction that caused the signal to be generated and logically prior to the performance of any start function that is initiated by the subsequent execution of START SUBCHANNEL for a subchannel that is enabled for measurement by this facility. If a subchannel that is enabled for measurement by this facility already has a start function in progress when the signal is generated, it is unpredictable when measurement data for that subchannel is stored by using the MBK and MBO associated with that signal.

While the measurement-block-update mode is active, performance measurements are accumulated for subchannels that are enabled for measurement-block update. Measurements for a subchannel are either accumulated in a single 32-byte format-0 measurement block within the measurement-block area, or a 64-byte format-1 measurement block pointed to by the measurement-block address at the subchannel. A subchannel is enabled for the measurement-block-update mode by setting the measurement-block-update-enable bit to one in the SCHIB and then executing the MODIFY SUBCHANNEL instruction for that subchannel. The measurement-block-format-control bit (F) at the subchannel specifies whether a

Condition code 0 is set to indicate that the actions described above have been taken.

Associated Functions

Subsequent to the execution of START SUBCHANNEL, the channel subsystem asynchronously performs the start function.

The contents of the ORB, other than the fields that must contain all zeros, are checked for validity. On some models, the fields of the ORB that must contain zeros are checked asynchronously, instead of during the execution of the instruction. When invalid fields are detected asynchronously, the subchannel becomes status pending with primary, secondary, and alert status and with deferred condition code 1 and program check indicated. (See “Program Check” on page 16-25.) In this situation, the I/O operation or chain of I/O operations is not initiated at the device, and the condition is indicated by the start-pending bit being stored as one when the SCSW is cleared by the execution of TEST SUBCHANNEL. (See “Subchannel-Status Word” on page 16-7).

On some models, path availability is tested asynchronously, instead of during the execution of the instruction. When no channel path is available for selection, the subchannel becomes status pending with primary and secondary status and with deferred condition code 3 indicated. The I/O operation or chain of I/O operations is not initiated at the device, and this condition is indicated by the start-pending bit being stored as one when the SCSW is cleared by the execution of TEST SUBCHANNEL.

If conditions allow, a channel path is chosen, and execution of the channel program that is designated in the ORB is initiated. (See “Start Function and Resume Function” on page 15-20.)

Special Conditions

Condition code 1 is set, and no other action is taken, when the subchannel is status pending when START SUBCHANNEL is executed. On some models, condition code 1 is not set when the subchannel is status pending with only secondary status; instead, the status-pending condition is discarded.

Condition code 2 is set, and no other action is taken, when a start, halt, or clear function is currently in progress at the subchannel (see “Function Control (FC)” on page 16-13).

Condition code 3 is set, and no other action is taken, when the subchannel is not operational for START SUBCHANNEL. A subchannel is not operational for START SUBCHANNEL if the subchannel is not provided in the channel subsystem, has no valid device number associated with it, or is not enabled.

A subchannel is also not operational for START SUBCHANNEL, on some models, when no channel path is available for selection. On these models, the lack of an available channel path is detected as part of the START SUBCHANNEL execution. On other models, channel-path availability is only tested as part of the asynchronous start function.

START SUBCHANNEL can encounter the program exceptions described or listed below.

In word 1 of the command-mode ORB, bits 26-30 must be zeros, and, in word 2 of the command-mode ORB, bit 0 must be zero. Otherwise, on some models, an operand exception is recognized. On other models, an I/O-interruption condition is generated, indicating program check, as part of the asynchronous start function.

START SUBCHANNEL can also encounter the program exceptions listed below.

When the multiple-subchannel-set facility is not installed, bits 32-47 of general register 1 must contain 0001 hex; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is installed, bits 32-44 of general register 1 must contain zeros, bits 45-46 must contain a valid value, and bit 47 must contain the value one; otherwise, an operand exception is recognized.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

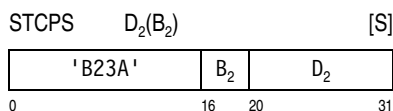
The execution of START SUBCHANNEL is suppressed on all addressing and protection exceptions.

Resulting Condition Code:

- 0 Function initiated
- 1 Status pending
- 2 Busy
- 3 Not operational

Program Exceptions:

- Access (fetch, operand 2)
- Operand
- Privileged operation
- Specification

STORE CHANNEL PATH STATUS

Depending on the model, this instruction may not be provided. When this instruction is not provided, it is checked for privileged operation exception and the instruction is suppressed by the machine

A channel-path-status word of up to 256 bits is stored at the designated location.

The second-operand address is the logical address of the location where the channel-path-status word is to be stored and must be designated on a 32-byte boundary; otherwise, a specification exception is recognized.

The channel-path-status word indicates which channel paths are actively communicating with a device at the time STORE CHANNEL PATH STATUS is executed. Bit positions 0-255 correspond, respectively, to the channel paths having the channel-path identifiers 0-255. Each of the 256 bits at the designated location is set to one, set to zero, or left unchanged, as follows:

- For all channel paths in the configuration that are actively communicating with devices at the time STORE CHANNEL PATH STATUS is executed, the corresponding bits are stored as ones.
- For all channel paths that are (1) provided in the system (PIM bit in the PMCW is one) and (2) in the configuration but not currently being used by the channel subsystem in actively communicating with devices, the corresponding bits are stored as zeros.
- For all channel paths that are not provided in the system (PIM bit in the PMCW is zero), the corresponding bits either are not stored or are stored as zeros.

- For all channel paths in the configuration that are in the channel-path-terminal state or are not physically available (the corresponding PAM bit in the PMCW is zero), the corresponding bits are stored as zeros.

Special Conditions

STORE CHANNEL PATH STATUS can encounter the program exceptions described or listed below.

The second operand must be designated on a 32-byte boundary; otherwise, a specification exception is recognized.

The execution of STORE CHANNEL PATH STATUS is suppressed on all addressing and protection exceptions.

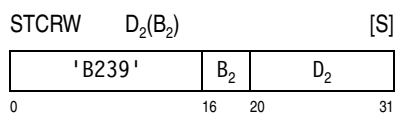
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Privileged operation
- Specification

Programming Notes:

1. To ensure a consistent interpretation of channel-path-status-word bits, the program should, prior to the initial use of the area, store zeros at the location where the channel-path-status word is to be stored.

STORE CHANNEL REPORT WORD

A CRW containing information affecting the channel subsystem is stored at the designated location.

The second-operand address is the logical address of the location where the CRW is to be stored and must be designated on a word boundary; otherwise, a specification exception is recognized.

When a malfunction or other condition affecting channel-subsystem operation is recognized, a channel report (consisting of one or more CRWs) describ-

ing the condition is made pending for retrieval and analysis by the program. The channel report contains information concerning the identity and state of a facility following the detection of the malfunction or other condition. For a description of the channel report, the CRW, and program-recovery actions related to the channel subsystem, see “Channel-Subsystem Recovery” on page 17-24.

When one or more channel reports are pending, the instruction causes a CRW to be stored at the designated location and condition code 0 to be set. A pending CRW can only be stored by the execution of STORE CHANNEL REPORT WORD and, once stored, is no longer pending. Thus, each pending CRW is presented only once to the program.

When no channel reports are pending in the channel subsystem execution of STORE CHANNEL REPORT WORD causes zeros to be stored at the designated location and condition code 1 to be set.

Special Conditions

STORE CHANNEL REPORT WORD can encounter the program exceptions described or listed below.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The execution of STORE CHANNEL REPORT WORD is suppressed on all addressing and protection exceptions.

Resulting Condition Code:

- 0 CRW stored
- 1 Zeros stored
- 2 —
- 3 —

Program Exceptions:

- Access (store, operand 2)
- Privileged operation
- Specification

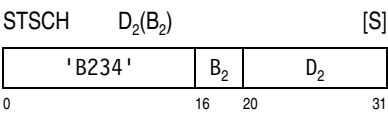
Programming Notes:

1. CRW overflow conditions may occur if STORE CHANNEL REPORT WORD is not executed to clear pending channel reports. If the overflow condition is encountered, one or more channel-

report words have been lost. (See “Channel-Subsystem Recovery” on page 17-24 for details.)

2. A pending CRW can be cleared by any CPU in the configuration executing STORE CHANNEL REPORT WORD, regardless of whether a machine-check interruption has occurred in any CPU.

STORE SUBCHANNEL



Control and status information for the designated subchannel is stored in the designated SCHIB.

General register 1 contains a subsystem-identification word that designates the subchannel for which the information is to be stored. The second-operand address is the logical address of the SCHIB and must be designated on a word boundary; otherwise, a specification exception is recognized.

When the extended-I/O-measurement-block facility is not installed, the information that is stored in the SCHIB consists of a path-management-control word, a SCSW, and three words of model-dependent information. When the extended-I/O-measurement-block facility is installed, the information that is stored in the SCHIB consists of a path-management-control word, a SCSW, the measurement-block-address field, and one word of model-dependent information. (See “Subchannel-Information Block” on page 15-2.)

The execution of STORE SUBCHANNEL does not change any information at the subchannel.

Condition code 0 is set to indicate that control and status information for the designated subchannel has been stored in the SCHIB. When the execution of STORE SUBCHANNEL results in the setting of condition code 0, the information in the SCHIB indicates a consistent state of the subchannel.

Special Conditions

Condition code 3 is set, and no other action is taken, when the designated subchannel is not operational for STORE SUBCHANNEL. A subchannel is

not operational for STORE SUBCHANNEL if the subchannel is not provided in the channel subsystem.

STORE SUBCHANNEL can encounter the program exceptions described or listed below.

When the multiple-subchannel-set facility is not installed, bits 32-47 of general register 1 must contain 0001 hex; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is installed, bits 32-44 of general register 1 must contain zeros, bits 45-46 must contain a valid value, and bit 47 must contain the value one; otherwise, an operand exception is recognized.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 SCHIB stored
- 1 —
- 2 —
- 3 Not operational

Program Exceptions:

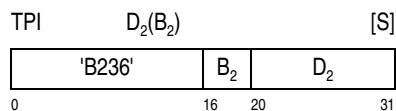
- Access (store, operand 2)
- Operand
- Privileged operation
- Specification

Programming Notes:

1. Device status that is stored in the SCSW may include device-busy, control-unit-busy, or control-unit-end indications.
2. The information that is stored in the SCHIB is obtained from the subchannel. The STORE SUBCHANNEL instruction does not cause the channel subsystem to interrogate the addressed device.
3. STORE SUBCHANNEL may be executed at any time to sample conditions existing at the subchannel, without causing any pending status conditions to be cleared.
4. Repeated execution of STORE SUBCHANNEL without an intervening delay (for example, to determine when a subchannel changes state)

should be avoided because repeated accesses of the subchannel by the CPU may delay or prohibit access of the subchannel by a channel subsystem to update the subchannel.

TEST PENDING INTERRUPTION



The I/O-interruption code for a pending I/O interruption at a subchannel is stored at the location designated by the second-operand address, and the pending I/O-interruption request is cleared.

The second-operand address, when nonzero, is the logical address of the location where the two-word I/O-interruption code, consisting of words 0 and 1, is to be stored. The second-operand address must be designated on a word boundary; otherwise, a specification exception is recognized.

If the second-operand address is zero, the three-word I/O-interruption code, consisting of words 0-2, is stored at real locations 184-195. In this case, low-address protection and key-controlled protection do not apply.

In the access-register mode when the second-operand address is zero, it is unpredictable whether access-register translation occurs for access register B_2 . If the translation occurs, the resulting address-space-control element is not used; that is, the interruption code still is stored at real locations 184-195.

Pending I/O-interruption requests are accepted only for those I/O-interruption subclasses allowed by the I/O-interruption-subclass mask in control register 6 of the CPU executing the instruction. If no I/O-interruption requests exist that are allowed by control register 6, the I/O-interruption code is not stored, the second-operand location is not modified, and condition code 0 is set.

If a pending I/O-interruption request is accepted, the I/O-interruption code is stored, the pending I/O-interruption request is cleared, and condition code 1 is set. The I/O-interruption code that is stored is the same as would be stored if an I/O interruption had occurred. However, PSWs are not swapped as when an I/O-interruption occurs.

Control and status information for the subchannel is stored in the designated IRB.

General register 1 contains a subsystem-identification word that designates the subchannel for which the information is to be stored. The second-operand address is the logical address of the IRB and must be designated on a word boundary; otherwise, a specification exception is recognized.

The information that is stored in the IRB consists of a SCSW, an extended-status word, and an extended-control word. (See “Interrupt-Response Block” on page 16-6.)

If the subchannel is status pending, the status-pending bit of the status-control field is stored as one. Whether or not the subchannel is status pending has an effect on the functions that are performed when TEST SUBCHANNEL is executed.

When the subchannel is status pending and TEST SUBCHANNEL is executed, information, as described above, is stored in the IRB, followed by the clearing of certain conditions and indications that exist at the subchannel as described in Figure 14-2 on page 14-21. If the subchannel is in transport mode, the clearing of these conditions, specifically the start function, places the subchannel in command mode. (See Figure 14-2 on page 14-21. If an I/O-interruption request is pending for the subchannel, the request is cleared. Condition code 0 is set to indicate that these actions have been taken.

When the subchannel is not status pending and TEST SUBCHANNEL is executed, information (as described above) is stored in the IRB, and no conditions or indications are cleared. Condition code 1 is set to indicate that these actions have been taken.

Figure 14-2 on page 14-21 describes which conditions and indications are cleared by TEST SUBCHANNEL when the subchannel is status pending. All other conditions and indications at the subchannel remain unchanged.

Field	Subchannel Conditions*									
	Alert Status Pending		Int Status Pending		Pri Status Pending		Sec. Status Pending		Status Pending Alone	
	CM	TM	CM	TM	CM	TM	CM	TM	CM	TM
Function Control	C	C	Nc	Nm	C	C	C	C	C	C
Activity Control	Cp	Cp	Nr	Nm	Cp	Cp	Cp	Cp	Cp	Cp
Status Control	Cs	Cs	Cs	C	Cs	Cs	Cs	Cs	Cs	Cs
N Condition	C	C	Nr	Nm	C	C	C	C	C	C
Q Condition	-	C	-	C	-	C	-	C	-	C
Explanation:										
-	N/A.									
CM	Command-Mode Operation									
TM	Transport-Mode Operation									
*	Note that the rightmost column applies to status pending when it is alone. The other four status-pending conditions result in the clearing actions given. These actions apply both when a single status-pending condition occurs and when a combination of the four status-pending conditions occurs. In the combination case, all the clearing actions of the individual cases apply.									
C	Cleared.									
Nm	Not reset or modified by TEST SUBCHANNEL.									
Cp	The resume-, start-, halt-, clear pending, and suspended conditions are cleared.									
Cs	The status-pending condition is cleared.									
Nc	Not changed unless function control indicates the halt function and activity control indicates suspended. If both the halt function and suspended are indicated, conditions are cleared as for status pending alone.									
Nr	Not changed unless activity control indicates suspended and function control indicates the start function with or without the halt function. If the halt function is indicated, the conditions are cleared as for status pending alone. If only the start function is indicated, the resume-pending condition and the N condition are cleared.									
Note: The clearing of certain subchannel conditions, places the subchannel in command mode. A subchannel in the idle state, or with the function control bits zero is considered to be in command-mode.										

Figure 14-2. Conditions and Indications Cleared at the Subchannel by TEST SUBCHANNEL

Special Conditions

Condition code 3 is set, and no other action is taken, when the subchannel is not operational for TEST SUBCHANNEL. A subchannel is not operational for TEST SUBCHANNEL if the subchannel is not provided, has no valid device number associated with it, or is not enabled.

TEST SUBCHANNEL can encounter the program exceptions described or listed below.

When the multiple-subchannel-set facility is not installed, bits 32-47 of general register 1 must contain 0001 hex; otherwise, an operand exception is recognized.

When the multiple-subchannel-set facility is installed, bits 32-44 of general register 1 must contain zeros, bits 45-46 must contain a valid value, and bit 47 must contain the value one; otherwise, an operand exception is recognized.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

When the execution of TEST SUBCHANNEL is terminated on addressing and protection exceptions, the state of the subchannel is not changed.

The priority of recognition of program exceptions for the instruction is shown in Figure 14-3 on page 14-22.

Resulting Condition Code:

- 0 IRB stored; subchannel status pending
- 1 IRB stored; subchannel not status pending
- 2 —
- 3 Not operational

Program Exceptions:

- Access (store, operand 2)
- Operand
- Privileged operation

• Specification

1.-7	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
8.A	Specification exception due to the address specified by the second operand not being on a word boundary.
8.B	Access exceptions for an access to the subchannel-status-word, extended-status-word, and extended-control-word sections of the IRB designated by the second operand.
8.C	Operand exception due to an invalid field in the the subsystem-identification word specified in register 1.
8.D	Access exceptions for an access to the extended-measurement-word section of the IRB designated by the second operand when the extended-I/O-measurement-word mode is enabled at the designated subchannel.

Figure 14-3. Priority of Execution: TEST SUBCHANNEL

Programming Notes:

1. Device status that is stored in the SCSW may include device-busy, control-unit-busy, or control-unit-end indications.
2. The information that is stored in the IRB is obtained from the subchannel. The TEST SUBCHANNEL instruction does not cause the channel subsystem to interrogate the addressed device.
3. When an I/O interruption occurs, it is the result of a status-pending condition at the subchannel, and typically TEST SUBCHANNEL is executed to clear the status. TEST SUBCHANNEL may also be executed at any other time to sample conditions existing at the subchannel.
4. Repeated execution of TEST SUBCHANNEL to determine when a start function has been completed should be avoided because there are conditions under which the completion of the start function may or may not be indicated. For example, if the channel subsystem is holding an interface-control-check (IFCC) condition in abeyance (for any subchannel) because another subchannel is already status pending, and if the start function being tested by TEST SUBCHANNEL has as the only path available for selection the channel path with the IFCC condition, then the

start function may not be initiated until the status-pending condition in the other subchannel is cleared, allowing the IFCC condition to be indicated at the subchannel to which it applies.

5. Repeated execution of TEST SUBCHANNEL without an intervening delay, for example, to determine when a subchannel changes state, should be avoided because repeated accesses of the subchannel by the CPU may delay or prohibit accessing of the subchannel by the channel subsystem. Execution of TEST SUBCHANNEL by multiple CPUs for the same subchannel at approximately the same time may have the same effect and also should be avoided.
6. The priority of I/O-interruption handling by a CPU can be modified by the execution of TEST SUBCHANNEL. When TEST SUBCHANNEL is executed and the designated subchannel has an I/O-interruption request pending, that I/O-interruption request is cleared, and the SCSW is stored, without regard to any previously established priority. The relative priority of the remaining I/O-interruption requests is unchanged.

Chapter 15. Basic I/O Functions

Control of Basic I/O Functions	15-1	Device-Command Word	15-37
Subchannel-Information Block	15-2	Transport-Command-Area Trailer	15-38
Path-Management-Control Word	15-2	Transfer-CBC-Offset Block DCW	15-40
Subchannel-Status Word	15-9	CBC-Offset Block	15-40
Model-Dependent Area/Measurement		Interrogate TCCB	15-41
Block Address	15-9	Interrogate DCW	15-41
Summary of Modifiable Fields	15-10	Interrogate Data	15-41
Channel-Path Allegiance	15-12	Transport Status Block	15-43
Working Allegiance	15-12	Transport-Status Header (TSH)	15-43
Working Allegiance for Subchannels		I/O-Status TSA	15-44
Operating in Command Mode	15-12	Device-Detected-Program-Check TSA	15-45
Working Allegiance for Subchannels		Interrogate TSA	15-50
Operating in Transport Mode	15-12	Command Code	15-52
Active Allegiance	15-13	Designation of Storage Area	15-52
Dedicated Allegiance	15-13	CCW Channel Program Chaining	15-55
Channel-Path Availability	15-13	Data Chaining	15-56
Control-Unit Type	15-14	Command Chaining	15-58
Clear Function	15-14	TCW Channel Program Chaining	15-58
Clear-Function Path Management	15-14	Skipping	15-59
Clear-Function Subchannel Modification	15-15	Program-Controlled Interruption	15-59
Clear-Function Signaling and Completion	15-15	Indirect-Storage Designator (ISD)	15-61
Halt Function	15-16	CCW Indirect Data Addressing	15-61
Halt-Function Path Management	15-16	Modified CCW Indirect Data Addressing	15-63
Halt-Function Signaling and Completion	15-17	Transport Indirect Data Addressing	15-65
Start Function and Resume Function	15-20	Suspension of CCW Channel-Program	
Start-Function and Resume-Function		Execution	15-68
Path Management	15-20	Commands and Flags for CCWs	15-70
Interrogate Function	15-22	Branching in CCW Channel Programs	15-71
Interrogate-Function Path Management	15-22	Transfer in Channel	15-72
Interrogate TCCB Transportation		Command Retry	15-72
and Completion	15-22	Concluding I/O Operations before Initiation	15-72
Execution of I/O Operations	15-23	Concluding I/O Operations during Initiation	15-73
Blocking of Data	15-24	Immediate Conclusion of Command-Mode	
Operation-Request Block	15-24	I/O Operations	15-73
Command-Mode ORB	15-25	Concluding I/O Operations During Data	
Transport-Mode ORB	15-30	Transfer	15-74
Channel-Command Word	15-31	Channel-Path-Reset Function	15-75
Transport Control Word	15-33	Channel-Path-Reset-Function Signaling	15-76
Transport-Command-Control Block	15-36	Channel-Path-Reset-Function-Completion	
Transport-Command-Area Header	15-36	Signaling	15-76
Transport-Command Area	15-37		

Some I/O instructions specify to the channel subsystem that a function is to be performed. Collectively, these functions are referred to as the basic I/O functions. The basic I/O functions are the clear, halt, start, resume, and channel-path-reset functions.

Control of Basic I/O Functions

Information that is present at the subchannel controls how the clear, halt, resume, and start functions are performed. This information is communicated to the

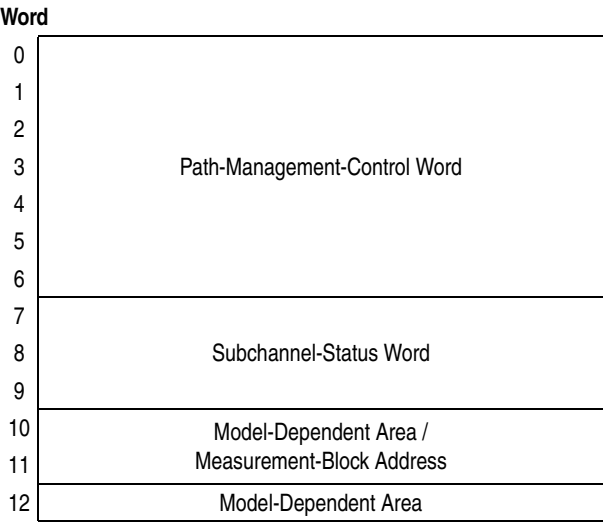
program in the subchannel-information block during the execution of STORE SUBCHANNEL.

Subchannel-Information Block

The subchannel-information block (SCHIB) is the operand of the MODIFY SUBCHANNEL and STORE SUBCHANNEL instructions. The two rightmost bits of the SCHIB address are zeros, designating the SCHIB on a word boundary. The SCHIB contains three major fields: the path-management-control word (PMCW), the subchannel-status word (SCSW), and a model-dependent area. When the extended-I/O-measurement-block facility is installed the SCHIB also contains the measurement-block-address field. (Figure 15-1 on page 15-2. shows the format of the PMCW, and Figure 16-3 on page 16-8 shows the format of the command-mode SCSW and Figure 16-8 on page 16-34 shows the format of the transport-mode SCSW.)

STORE SUBCHANNEL is used to store the current PMCW, the SCSW, model-dependent data, and, when the extended-I/O-measurement-block facility is installed, the measurement-block-address field, for the designated subchannel. MODIFY SUBCHANNEL alters certain PMCW fields and, when the extended-I/O-measurement-block facility is installed, the measurement-block address in the subchannel. When the program needs to change the contents of one or more of the PMCW fields, the normal procedure is to (1) issue STORE SUBCHANNEL to obtain the current contents, (2) perform the required modifications to the PMCW field or the measurement-block-address field in main storage, and (3) issue MODIFY

SUBCHANNEL to pass the new information to the subchannel. The SCHIB has the following format:



Path-Management-Control Word

Words 0-6 of the SCHIB contain the path-management-control word (PMCW). The PMCW has the format shown in Figure 15-1 on page 15-2 when the subchannel is valid (see “Device Number Valid (V)” on page 15-4).

The format of the PMCW is as follows:

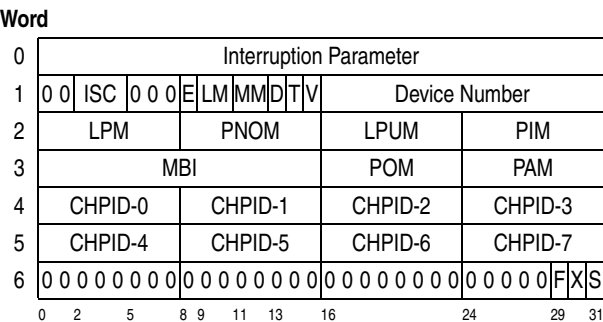


Figure 15-1. PMCW Format

Interruption Parameter: Bit positions 0-31 of word 0 contain the interruption parameter that is stored as word 1 of the interruption code. The interruption parameter can be set to any value by START SUBCHANNEL and MODIFY SUBCHANNEL. The initial value of the interruption parameter is zero.

I/O-Interruption-Subclass Code (ISC): Bits 2-4 of word 1 are an unsigned binary integer, in the range 0-7, that corresponds to the bit position of the I/O-

interruption subclass-mask bit in control register 6 of each CPU in the configuration. The setting of that mask bit in control register 6 of a CPU controls the recognition of interruption requests relating to this subchannel by that CPU (see “Priority of Interruptions” on page 16-5). The ISC can be set to any value by MODIFY SUBCHANNEL. The initial value of the ISC is zero.

Reserved: Bits 0, 1, and 5-7 of word 1 are reserved and stored as zeros by STORE SUBCHANNEL. Bits 0, 1, 6, and 7 must be zeros when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized. Bit 5 of word 1 is ignored when MODIFY SUBCHANNEL is executed.

Enabled (E): Bit 8 of word 1, when one, indicates that the subchannel is enabled for all I/O functions. When the E bit is zero, status presented by the device is not made available to the program, and I/O instructions other than MODIFY SUBCHANNEL and STORE SUBCHANNEL that are executed for the designated subchannel cause condition code 3 to be set. The E bit can be either zero or one when MODIFY SUBCHANNEL is executed; initially, all subchannels are not enabled; IPL causes the IPL I/O device to become enabled.

Limit Mode (LM): When the address-limit-checking facility is installed, bits 9 and 10 of word 1 define the limit mode (LM) of the subchannel. The limit mode is used by the channel subsystem when address-limit checking is invoked for an I/O operation. (See “Address-Limit Checking” on page 17-22.) Address-limit checking is under the control of the address-limit-checking-control bit that is passed to the subchannel in the operation-request block (ORB) during the execution of START SUBCHANNEL. (See “Address-Limit-Checking Control (A)” on

page 15-27.) The definitions of the LM bits, whose values are used during data transfer, are as follows:

Bit 9	Bit 10	Function
0	0	Initialized value. No limit checking is performed for this subchannel.
0	1	Data address must be equal to or greater than the current address limit.
1	0	Data address must be less than the current address limit.
1	1	Reserved.

Bit positions 9 and 10 can contain any of the first three bit combinations shown above when MODIFY SUBCHANNEL is executed. Specification of the reserved bit combination in the operand causes an operand exception to be recognized when MODIFY SUBCHANNEL is executed.

When the address-limit-checking facility is not installed, bits 9-10 of word 1 may be set as described above; however, they are ignored and are not set at the specified subchannel. When bits 9-10 are not specified as described above and MODIFY SUBCHANNEL is executed, an operand exception is recognized.

Measurement-Mode Enable (MM): Bits 11 and 12 of word 1 enable the measurement-block-update mode and the device-connect-time-measurement mode, respectively, of the subchannel. These bits can have any value when MODIFY SUBCHANNEL is executed; initially, neither measurement mode is

enabled. The definition of each of these bits is as follows:

Bit

11 Measurement-Block-Update Enable:

- 0 Initialized value. The subchannel is not enabled for measurement-block update. Storing of measurement-block data does not occur.
- 1 The subchannel is enabled for measurement-block update. If the measurement-block-update mode is active, measurement data is accumulated in the measurement block at the time channel-program execution is completed or suspended at the subchannel or completed at the device, as appropriate, provided no error conditions described by subchannel logout have been detected. (See “Measurement-Block Update” on page 17-2.) If the measurement-block-update mode is inactive, no measurement-block data is stored.

Bit

12 Device-Connect-Time-Measurement Enable:

- 0 Initialized value. The subchannel is not enabled for device-connect-time measurement. Storing of the device-connect-time interval (DCTI) in the extended-status word (ESW) does not occur.
- 1 The subchannel is enabled for device-connect-time measurement. If the device-connect-time-measurement mode is active and timing facilities are provided for the subchannel, the value of the DCTI is stored in the ESW when TEST SUBCHANNEL is executed after channel-program execution is completed or suspended at the subchannel, provided no error conditions described by subchannel logout have been detected. If the device-connect-time-measurement mode is inactive, no measurement values are stored in the ESW.

The meaning of the measurement-mode-enable bits (MM), described above, applies when the timing-facility bit for the subchannel is one. When the timing-facility bit is zero, the effect of the MM bits is changed, as described below under “Timing Facility (T)” on page 15-4. (For more discussion on measurement modes, see “Measurement-Block Update” on

page 17-2 and “Device-Connect-Time Measurement” on page 17-9.)

Multipath Mode (D): Bit 13 of word 1, when one, indicates that the subchannel operates in the multipath mode when performing an I/O operation or chain of I/O operations. For proper operation in the multipath mode when more than one channel path is available for selection, the associated device must have the dynamic-reconnection feature installed and must be set up for multipath-mode operation. During performance of a start function in the multipath mode, a device is allowed to request service from the channel subsystem over any of the channel paths indicated at the subchannel as being available for selection (see “Logical-Path Mask (LPM)” on page 15-5 and “Path-Available Mask (PAM)” on page 15-7). Bit 13, when zero, indicates that the subchannel operates in single-path mode when performing an I/O operation or chain of I/O operations. In the single-path mode, the entire start function is performed by using the channel path on which the first command of the I/O operation or chain of I/O operations was accepted by the device. The D bit can be either zero or one when MODIFY SUBCHANNEL is executed; initially, the subchannel is in the single-path mode.

Timing Facility (T): Bit 14 of word 1, when one, indicates that the channel-subsystem-timing facility is available for the subchannel and is under the control of the two measurement-mode-enable bits (MM) and SET CHANNEL MONITOR. Bit 14, when zero, indicates that the channel-subsystem-timing facility is not available for the subchannel. When bit 14 is zero, the START SUBCHANNEL count is the only measurement data that can be accumulated in the measurement block for the subchannel. Storing of the START SUBCHANNEL count is under the control of bit 11 and SET CHANNEL MONITOR, as described above under “Measurement-Mode Enable (MM)” on page 15-3. Similarly, if the T bit is zero, no device-connect-time-interval (DCTI) values can be measured for the subchannel. (See “Measurement-Block Update” on page 17-2 and “Device-Connect-Time Measurement” on page 17-9.)

Device Number Valid (V): Bit 15 of word 1, when one, indicates that the device-number field (see below) contains a valid device number and that a device associated with this subchannel may be physically installed. Bit 15, when zero, indicates that the subchannel is not valid, there is no I/O device currently associated with the subchannel, and the con-

tents of all other defined fields of the SCHIB are unpredictable.

Device Number: Bit positions 16-31 of word 1 contain the binary representation of the four-digit hexadecimal device number of the device that is associated with this subchannel. The device number is a system-unique parameter that is assigned to the subchannel and the associated device when the device is installed.

Logical-Path Mask (LPM): Bits 0-7 of word 2 indicate the logical availability of channel paths to the associated device. Each bit of the LPM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. A bit set to one means that the corresponding channel path is logically available; a zero means the corresponding channel path is logically not available. When a channel path is logically not available, the channel subsystem does not use that channel path to initiate performance of any clear, halt, resume, or start function, except when a dedicated allegiance exists for that channel path. When a dedicated allegiance exists at the subchannel for a channel path, the logical availability of the channel path is ignored whenever a clear, halt, resume, or start function is performed. (See “Channel-Path Allegiance” on page 15-12). If the subchannel is idle, the logical availability of the channel path is ignored whenever the control unit initiates a request to present alert status to the channel subsystem. The logical availability of a channel path associated with the subchannel can be changed by setting the corresponding LPM bit in the SCHIB and then issuing MODIFY SUBCHANNEL, or by setting the corresponding LPM bit in the ORB and then issuing START SUBCHANNEL. Initially, each installed channel path is logically available.

Path-Not-Operational Mask (PNOM): Any of bits 8-15 of word 2, when one, indicates that a path-not-operational condition has been recognized on the corresponding channel path. Each bit of the PNOM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. The channel subsystem recognizes a path-not-operational condition when, during an attempted device selection in order to perform a clear, halt, resume, or start function, the device associated with the subchannel appears not operational on a channel path that is operational for the subchannel. When a path-not-operational condition is recognized, the state of the channel path changes from

operational for the subchannel to not operational for the subchannel. A channel path is operational for the subchannel if the associated device appeared operational on that channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function. A device appears to be operational on a channel path when the device responds to an attempted device selection. A channel path is not operational for the subchannel if the associated device appeared not operational on that channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function. Any of bits 8-15 of word 2, when zero, indicates that a path-not-operational condition has not been recognized on the corresponding channel path.

Initially, each of the eight possible channel paths associated with each subchannel is considered to be operational, regardless of whether the respective channel paths are installed or available; therefore, unless a path-not-operational condition is recognized during initial program loading, the PMCW, if stored, contains a PNOM of all zeros if stored prior to the execution of a CLEAR SUBCHANNEL, HALT SUBCHANNEL, RESUME SUBCHANNEL, or START SUBCHANNEL instruction.

Programming Note: The PNOM indicates those channel paths for which a path-not-operational condition has been recognized during the performance of the most recent clear, halt, resume, or start function. That is, the PNOM indicates which of the channel paths associated with the subchannel have made a transition from the operational to the not-operational state for the subchannel during the performance of the most recent clear, halt, resume, or start function. However, the transition of a channel path from the not-operational to the operational state for the subchannel is indicated in the POM. Therefore, the POM must be examined in order to determine whether any of the channel paths that are associated with a designated subchannel are operational for the subchannel.

Furthermore, while performing either a start function or a resume function, the transition of a channel path from the not-operational to the operational state for the subchannel is recognized by the channel subsystem only during the initiation sequence for the first command specified by the start function or implied by the resume function. Therefore, a channel path that is currently not operational for the subchannel can be used by the device associated with the subchannel

when reconnecting to the channel subsystem in order to continue command chaining; however, the channel subsystem does not indicate a transition of that channel path from the not-operational to the operational state for the subchannel in the POM.

POM Value and Device State before Selection Attempt		Value of Specified Bit Subsequent to Selection Attempt		
Device State ¹	POM	POM	PNOM ²	SCSW N Bit
OP	0	1	0	0
NOP	0	0	0	0
OP	1	1	0	0
NOP	1	0	1	1 ³

Explanation:

¹ Device state as it appears on the corresponding channel path.

² Prior to the attempted device selection during the performance of either a start function or a resume function while the subchannel is suspended, the channel subsystem clears all existing path-not-operational conditions, if any, at the designated subchannel.

³ The N bit (bit 15 of word 0 of the SCSW) is indicated to the program and the N condition is cleared at the subchannel when TEST SUBCHANNEL is executed the next time the subchannel is status pending for other than intermediate status alone provided that it is not also suspended.

NOP The device is not operational on the corresponding channel path.

OP The device is operational on the corresponding channel path.

Figure 15-2. Resulting POM, PNOM, and N-Bit Values Subsequent to Selection Attempt

Last-Path-Used Mask (LPUM): Bits 16-23 of word 2 indicate the channel path that was last used for communicating or transferring information between the channel subsystem and the device. Each bit of the LPUM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB.

For a subchannel operating in command mode, each bit of the LPUM is stored as zero, except for the bit that corresponds to the channel path last used, whenever one of the following occurs:

1. The first command of a start or resume function is accepted by the device (see “Activity Control (AC)” on page 16-14).

2. The device and channel subsystem are actively communicating when the suspend function is performed for the channel program in execution.
3. Status has been accepted from the device and is recognized as an interruption condition, or a condition has been recognized that suppresses command chaining (see “Interruption Conditions” on page 16-2).
4. An interface-control-check condition has been recognized (see “Interface-Control Check” on page 16-29), and no subchannel-logout information is currently present in the subchannel.

For a subchannel operating in transport mode, each bit of the LPUM is stored as zero, except for the bit that corresponds to the channel path last used, whenever one of the following occurs:

1. The path has been selected for transporting the TCCB for the operation.
2. Status has been accepted from the device and is recognized as an interruption condition. If the accepted status is an interrogate response, the LPUM may be different than that stored with primary or secondary status or both.
3. An interface-control-check has been recognized and no subchannel-logout information is currently present in the subchannel.

The LPUM field of the PMCW contains the most recent setting. For transport-mode operations the LPUM in the subchannel is set to zeros when the start function is set and no dedicated-allegiance condition exists for the subchannel.

Path-Installed Mask (PIM): Bits 24-31 of word 2 indicate which of the channel paths 0-7 to the I/O device are physically installed. The PIM indicates the validity of the channel-path identifiers (see below) for those channel paths that are physically installed. Each bit of the PIM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. A PIM bit stored as one indicates that the corresponding channel path is installed. A PIM bit stored as zero indicates that the corresponding channel path is not installed. The PIM always reflects the full complement of installed paths to the device, regardless of how the system is configured. Therefore, some of the channel paths indicated in the PIM may not be physically available in that configuration, as indicated by

the bit settings in the path-available mask (see below). The initial value of the PIM indicates all the physically installed channel paths to the device.

Measurement-Block Index (MBI): Bits 0-15 of word 3 form an index value used by the measurement-block-update facility when the measurement-block-update mode is active (see “SET CHANNEL MONITOR” on page 14-13.) and the subchannel is enabled for the mode (see “Measurement-Mode Enable (MM)” on page 15-3).

When the measurement-block index is used, five zero bits are appended on the right, and the result is added to the measurement-block-origin address designated by SET CHANNEL MONITOR. The calculated address, called the measurement-block address, designates the beginning of a 32-byte storage area where measurement data is stored. (See “Measurement Block” on page 17-3.) The MBI can contain any value when MODIFY SUBCHANNEL is executed; the initial value is zero.

Programming Note: The measurement-block-origin address specified by SET CHANNEL MONITOR is used as the origin address for the measurement blocks for subchannels in all subchannel sets available to the program. However, the two-byte measurement-block-index field is capable of supporting a maximum of 65,536 unique MBI values. It is the responsibility of the program to coordinate use of the measurement-block index among subchannels and subchannel sets.

Path-Operational Mask (POM): Bits 16-23 of word 3 indicate the last known operational state of the device on the corresponding channel paths. Each bit of the POM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. If the associated device appeared operational on a channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function, then the channel path is operational for the subchannel, and the bit corresponding to the channel path in the POM is one. A device appears to be operational on a channel path when the device responds to an attempted device selection. A channel path is also operational for the subchannel if MODIFY SUBCHANNEL is executed and the bit corresponding to that channel path in the POM is specified as one.

If the associated device appeared not operational on a channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function, then the channel path is not operational for the subchannel, and the bit corresponding to the channel path in the POM is zero. A channel path is also not operational for the subchannel if MODIFY SUBCHANNEL is executed and the bit corresponding to that channel path in the POM is specified as zero.

If the device associated with the subchannel appears not operational on a channel path that is operational for the subchannel during an attempted device selection in order to perform a clear, halt, resume, or start function, then the channel subsystem recognizes a path-not-operational condition. If an SCSW is subsequently stored, then bit 15 of word 0 is one, indicating the path-not-operational condition. When a path-not-operational condition is recognized, the state of the channel path changes from operational for the subchannel to not operational for the subchannel.

When the channel path is not operational for the subchannel, a path-not-operational condition cannot be recognized. Moreover, a channel path that is not operational for the subchannel may be available for selection; if the channel subsystem chooses that channel path while performing a path-management operation, and if, during the attempted device selection, the device appears to be operational again on that channel path, then the state of the channel path changes from not operational for the subchannel to operational for the subchannel.

The POM can contain any value when MODIFY SUBCHANNEL is executed. Initially, each of the eight possible channel paths associated with each subchannel is considered to be operational, regardless of whether the respective channel paths are installed or available; therefore, unless a path-not-operational condition is recognized during initial program loading, the PMCW, if stored, contains a POM of all ones if stored prior to the execution of a CLEAR SUBCHANNEL, HALT SUBCHANNEL, RESUME SUBCHANNEL, or START SUBCHANNEL instruction.

Path-Available Mask (PAM): Bits 24-31 of word 3 indicate the physical availability of installed channel paths. Each bit of the PAM corresponds one-for-one, by relative bit position, with a CHPID located in an associated byte of words 4 and 5 of the SCHIB. A PAM bit of one indicates that the corresponding channel path is physically available for use in access-

ing the device. A PAM bit of zero indicates the channel path is not physically available for use in accessing the device. When a channel path is not physically available, it may, depending on the model and the extent of failure, be used during performance of the reset-channel-path function. A channel path that is physically available may become not physically available as a result of reconfiguring the system, or this may occur as a result of the performance of the channel-path-reset function. The initial value of the PAM reflects the set of channel paths by which the I/O device is physically accessible at the time of initialization.

Note: The change in the availability of a channel path affects all subchannels having access to that channel path. Whenever the setting of a PAM bit is referred to in conjunction with the availability status of a channel path, for brevity, reference is made in this chapter to a single PAM bit instead of to the respective PAM bits in all of the affected subchannels.

Channel-Path Identifiers (CHPIDs): Words 4 and 5 contain eight one-byte channel-path identifiers corresponding to channel paths 0-7 of the PIM. A CHPID is valid if the corresponding PIM bit is one. Each valid CHPID contains the identifier of a physical channel path to a control unit by which the associated I/O device may be accessed. A unique CHPID is assigned to each physical channel path in the system.

Different devices that are accessible by the same physical channel path have, in their respective subchannels, the same CHPID value. The CHPID value may, however, appear in each subchannel in different locations in the CHPID fields 0-7.

Subchannels that share an identical set of channel paths have the same corresponding PIM bits set to ones. The channel-path identifiers (CHPIDs) for these channel paths are the same and occupy the same respective locations in each SCHIB.

Reserved: Bits 0-28 of word 6 are reserved and are stored as zeros by STORE SUBCHANNEL. They must be zeros when MODIFY SUBCHANNEL is executed; otherwise, an operand exception may be recognized.

Measurement Block Format Control (F): When the extended-I/O-measurement-block facility is installed, bit 29 of word 6 specifies the format of the measurement block to be stored when the subchan-

nel is enabled for the measurement-block-update mode, and measurement-block-update mode is active. The bit can contain any value when MODIFY SUBCHANNEL is executed. The initial value is zero. The definition of the bit is as follows:

Bit

29 Measurement-Block-Format Control:

- 0 Format-0 measurement block. Specifies that a format-0 measurement block is used when performing a measurement-block update for the subchannel. The address of the 32-byte measurement block is obtained using the MBI provided by MSCH in conjunction with the MBO provided by SCHM.
- 1 Format-1 measurement block. Specifies that a format-1 measurement block is used when performing a measurement-block update for the subchannel. The address of the 64-byte format-1 measurement block is provided by MSCH.

If the extended-measurement-block facility is not installed, bit 29 of word 6 of the SCHIB operand must be zero when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized.

Extended Measurement Word Mode Enable (X):

When the extended-measurement-word facility is installed and enabled, bit 30 of word 6 enables the extended-measurement-word mode for the subchannel. Initially, the extended-measurement-word mode is not enabled. The definition of the bit is as follows:

Bit

30 Measurement-Block-Word-Mode Enable:

- 0 Initialized value. The subchannel is not enabled for extended-measurement-word mode. Storing of the extended-measurement word does not occur.
- 1 The subchannel is enabled for extended-measurement-word mode. Measurement data is stored in the extended-measurement word at the time channel-program execution is completed or suspended at the subchannel or completed at the device, as appropriate, provided no error conditions described by subchannel logout have been detected.

If the extended-measurement-word facility is not installed, or is installed but is not enabled, bit 30 of word 6 of the SCHIB operand must be zero when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized.

Concurrent Sense (S): Bit 31 of word 6, when one, indicates that the subchannel is in the concurrent-sense mode. When the subchannel is in command mode and concurrent-sense mode, whenever the subchannel becomes status pending with alert status, and the status byte accepted from the device contains the unit-check indication, then the channel subsystem may attempt to retrieve sense information from the associated device and place that sense information in the extended-control word.

When the subchannel is operating in transport mode, any available sense information is provided in the TSB regardless of the setting of bit 31 of word 6.

If the concurrent-sense facility is not installed, bit 31 of word 6 of the SCHIB operand must be zero when MODIFY SUBCHANNEL is executed; otherwise, an operand exception is recognized.

Subchannel-Status Word

Words 7-9 of the SCHIB contain a copy of the SCSW. The format of the SCSW is described in "Subchannel-Status Word" on page 16-7. The SCSW is stored

by the execution of either STORE SUBCHANNEL or TEST SUBCHANNEL (see "STORE SUBCHANNEL" on page 14-18 and "TEST SUBCHANNEL" on page 14-20).

Model-Dependent Area/Measurement Block Address

When the extended-I/O-measurement-block facility is not installed, words 10-12 of the SCHIB contain model-dependent information.

When the extended-I/O-measurement-block facility is installed, words 10-11 are defined as the measurement-block-address field. Word 12 contains model-dependent information.

When (1) the measurement-block-update mode is active (see "SET CHANNEL MONITOR" on page 14-13), (2) the subchannel is enabled for the mode (see "Measurement-Mode Enable (MM)" on page 15-3), and (3) the format-1-measurement block is specified (see "Measurement Block Format Control (F)" on page 15-8) at the subchannel, the measurement-block-address field contains the absolute storage address of the measurement block used by the measurement-block-update facility. The measurement-block address designates the beginning of a 64-byte storage area and must be designated on 64-byte boundary. The initial value of the measurement block address is zero.

Summary of Modifiable Fields

*Figure 15-3 on page 15-10 lists the initial settings for fields in a subchannel whose device-number-valid bit is one and indicates what modifies the fields.

All of the PMCW fields contain meaningful information when STORE SUBCHANNEL is executed and the designated subchannel is idle. Subchannel fields

that the channel subsystem does not modify contain valid information whenever STORE SUBCHANNEL is executed, provided that the device-number-valid bit is one. The validity of the subchannel fields that are modifiable by the channel subsystem depends on the state of the subchannel at the time STORE SUBCHANNEL is executed.

Subchannel Field	Initial Value ¹	Program Modifies by Executing	Modified by Channel Subsystem ²
Interruption parameter	Zeros	MSCH,SSCH	No
I/O-interruption-subclass code	Zeros	MSCH	No
Enabled (E)	Zero	MSCH	No
Limit mode (LM)	Zeros	MSCH ⁷	No
Measurement mode (MM)	Zeros	MSCH	Yes ³
Multipath mode (D)	Zero	MSCH	No
Timing facility (T)	Installed value ⁴	None	No
Device number valid (V)	Installed value ⁴	None	No
Device number	Installed value ⁴	None	No
Logical-path mask (LPM)	Path-installed-mask value	MSCH,SSCH	No
Path-not-operational mask (PNOM)	Zeros	CSCH,SSCH,RSCH ⁵	Yes
Last-path-used mask (LPUM)	Zeros	CSCH	Yes
Path-installed mask (PIM)	Installed value ⁴	None	No
Measurement-block index (MBI)	Zeros	MSCH	No
Path-operational mask (POM)	Ones	CSCH,MSCH,RSCH ⁵	Yes
Path-available mask (PAM)	Installed values ^{4 6}	None	Yes ⁶
Channel-path ID 0-7	Installed value ₄	None	No
Concurrent sense (S)	ZERO	MSCH	No
Subchannel-status word (SCSW)	Zero	TSCH	Yes
Model-dependent area	*	None	*
Measurement-block-format control (F)	Zero	MSCH	No
Extended-measurement-word enable (X)	Zero	MSCH	No
Measurement-block address	Zeros	MSCH	No

Figure 15-3. Modification of Subchannel Fields (Part 1 of 2)

Explanation:

- * Model dependent.
- 1 These fields are not meaningful if the subchannel is not valid. Initialization of a subchannel is performed when I/O-system reset occurs. (See the section "I/O-System Reset" in Chapter 17, "I/O Support Functions.") One or more of the installed-value parameters that are unmodifiable by the program may be set when the subchannel is idle. In this case, all the program-modifiable fields are set to their initialized values, and the program is notified of such a change by a channel report. (See the section "Channel-Report Word" in Chapter 17, "I/O Support Functions.")
- 2 Subchannel fields that are not normally modifiable by the channel subsystem may be modified as a result of dynamic configuration changes or as a result of external actions. When this occurs, the program is notified of the change by a channel report that is made pending at the time of the change.
- 3 When any of the following error conditions associated with the measurement-block-update mode is detected, the measurement-block-update mode is disabled by the channel subsystem (bit 11 of word 1 of the SCHIB is zero) in the affected subchannel. The device-connect-time-measurement-enable bit (bit 12 of word 1 of the SCHIB) is never modified by the channel subsystem.
 - Measurement program check
 - Measurement protection check
 - Measurement data check
 - Measurement key check
- 4 This information is entered when the channel-subsystem configuration is established
- 5 The mask is modified by the resume function only when the subchannel is in the suspended state at the time RESUME SUBCHANNEL is executed.
- 6 The channel subsystem may modify the PAM to reflect changes in the system configuration caused by partitioning or unpartitioning channel paths because of reconfiguration or permanent failure of part of the I/O system.

Figure 15-3. Modification of Subchannel Fields (Part 2 of 2)

Programming Notes:

1. System performance may be degraded if the LPM is not used to make channel paths for which a path-not-operational condition has been indicated in the PNOM logically not available.
2. If, during the performance of a start function, a channel path becomes not physically available because a channel-path failure has been recognized, continued performance of the start function may be precluded. That is, the program may or may not be notified, and the subchannel may remain in the subchannel-and-device-active state until cleared by the performance of the clear function.
3. If the same MBI is placed in more than one subchannel by the program, the channel-subsystem-monitoring facility updates the same locations with measurement data relating to more than one subchannel. In this case, the values stored in the measurement data are unpredictable. (See "Measurement-Block Update" on page 17-2.)
4. Modification of the I/O configuration (reconfiguration) may be accomplished in various ways depending on the model. If the reconfiguration procedure affects the physical availability of a channel path, then any change in availability can be detected by executing STORE SUBCHANNEL for a subchannel that has access to the channel path and by subsequently examining the PAM bits of the SCHIB.
5. The definitions of the PNOM, POM, and N bit are such that a path-not-operational condition is reported to the program only the first time the condition is detected by the channel subsystem after the corresponding POM bit is set to one.

For example, if the POM bit for every channel path available for selection is one and the device appears not operational on all corresponding channel paths while the channel subsystem is attempting to initiate a start function at the device, the channel subsystem makes the subchannel status pending, with deferred condition code 3 and with the N bit stored as one. The PNOM in the SCHIB indicates the channel path or channel paths that appeared not operational, for which the corresponding POM bits have been set to zeros. The next START SUBCHANNEL causes the channel subsystem to again attempt device selection by choosing a channel path from among all of the channel paths that are available

for selection. If device selection is not successful and all channel paths available for selection have again been chosen, deferred condition code 3 is set, but the N bit in the SCSW is zero. The POM contains zeros in at least those bit positions that correspond to the channel paths that are available for selection. (See “Channel-Path Availability” on page 15-13 for a description of the term “available for selection.”) When the N bit in the SCSW is zero, the PNOM is also zero.

6. If the program is to detect path-not-operational conditions, the PNOM should be inspected following the execution of TEST SUBCHANNEL (which results in the setting of condition code zero and the valid storing of the N bit as one) and preceding the performance of another start, resume, halt, or clear function at the subchannel.

Channel-Path Allegiance

The channel subsystem establishes allegiance conditions between subchannels and channel paths. The kind of allegiance established at a subchannel for a channel path or set of channel paths depends upon the state of the subchannel, the device, and the information, if any, transferred between the channel subsystem and device. The way in which path management is handled during the performance of a clear, halt, resume, or start function is determined by the kind of allegiance, if any, currently recognized between a subchannel and a channel path.

Performing the clear function at a subchannel clears any currently existing allegiance condition in the subchannel for all channel paths.

Performing the reset-channel-path function clears all currently existing allegiances for that channel path in all subchannels.

When a channel path becomes not physically available, all internal indications of prior allegiance conditions are cleared in all subchannels having access to the designated channel path.

Note that allegiance rules do not apply for interrogate operations. An interrogate operation may be initiated and successfully complete while a subchannel is in transport mode. Furthermore, interrogate operations do not alter any allegiance conditions that may exist.

Working Allegiance

Working Allegiance for Subchannels Operating in Command Mode

A subchannel has a working allegiance for a channel path when the subchannel is operating in command mode and becomes device active on that channel path. Once a working allegiance is established, the channel subsystem maintains the working allegiance at the subchannel for the channel path until either the subchannel is no longer device active or a dedicated allegiance is recognized, whichever occurs earlier. Unless a dedicated allegiance is recognized, a working allegiance for a channel path is extended to the set of channel paths that are available for selection if the device is specified to be operating in the multipath mode (that is, the multipath-mode bit is stored as one in the SCHIB). Otherwise, the working allegiance remains only for that channel path over which the start function was initiated.

Once a working allegiance is established for a channel path or set of channel paths, the working allegiance is not changed until the subchannel is no longer device active or until a dedicated allegiance is established. If the subchannel is operating in the single-path mode, a working allegiance is maintained only for a single path.

While a working allegiance exists at a subchannel, an active allegiance can occur only for a channel path for which the working allegiance is being maintained, unless the device is specified as operating in the multipath mode. When the device is specified as operating in the multipath mode, an active allegiance may also occur for a channel path that is not available for selection if the presentation of status by the device on that channel path causes an alert interruption condition to be recognized.

A working allegiance is cleared in any subchannel having access to a channel path if the channel path becomes not physically available.

Working Allegiance for Subchannels Operating in Transport Mode

A subchannel has a working allegiance for a channel path when the subchannel is operating in transport mode, becomes start pending, and the designated TCCB is transported over that channel path to the I/O device. Once a working allegiance is established, the channel subsystem maintains the working allegiance

at the subchannel for the channel path until a busy condition is encountered or the subchannel is made status pending with primary status. The working allegiance remains only for that channel path over which the start function was initiated.

While a working allegiance exists at a subchannel that is operating in transport mode, an active allegiance can occur only for the channel path for which the working allegiance is being maintained.

A working allegiance is cleared in any subchannel having access to a channel path if the channel path becomes not physically available.

Active Allegiance

A subchannel has an active allegiance established for a channel path no later than when active communication has been initiated on that channel path with an I/O device. The subchannel can have an active allegiance to only one channel path at a time. While the subchannel has an active allegiance for a channel path, the channel subsystem does not actively communicate with that device on any other channel path. When the channel subsystem accepts a no-longer-busy indication from the device that does not cause an interruption condition, this status does not constitute the initiation of active communication. An active allegiance at a subchannel for a channel path is terminated when the channel subsystem is no longer actively communicating with the I/O device on that channel path.

A working allegiance can become an active allegiance.

Dedicated Allegiance

If a channel path is physically available (that is, if the corresponding PAM bit is one), a dedicated allegiance may be recognized for that channel path. If a channel path is not physically available, a dedicated allegiance cannot be recognized for the corresponding channel path. The channel subsystem establishes a dedicated allegiance at the subchannel for a channel path when (1) the subchannel is operating in command mode, the subchannel becomes status pending with alert status, and device status containing the unit-check indication is present but (2) concurrent-sense information is not present at the

subchannel. A dedicated allegiance is maintained until the subchannel is no longer start pending (unless it becomes suspended) or resume pending following performance of the next start function, clear function, or channel-path-reset function or the next resume function if applicable. If the subchannel becomes suspended, the dedicated allegiance remains until the resume function is initiated and the subchannel is no longer resume pending. Unless a clear or channel-path-reset function is performed, the subchannel establishes a working allegiance when the dedicated allegiance ends. This occurs when the subchannel becomes device active. While a dedicated allegiance exists at a subchannel for a channel path, only that channel path is available for selection until the dedicated-allegiance condition is cleared.

Dedicated allegiance does not apply to subchannels operating in transport mode. When a subchannel operating in transport mode becomes status pending with unit check indicated, the sense information has already been transferred from the I/O device into the TSB for the I/O operation.

A dedicated allegiance can become an active allegiance. While a dedicated allegiance exists, an active allegiance can only occur for the same channel path.

A currently existing dedicated allegiance is cleared at any subchannel having access to a channel path when the channel path becomes not physically available or whenever the device appears not operational on the channel path for which the dedicated allegiance exists.

Channel-Path Availability

When a channel path is not physically available, the channel subsystem does not use the channel path to perform any of the basic I/O functions except, in some cases, the channel-path-reset function and does not respond to any control-unit-initiated requests on that same channel path. If a channel path is not physically available, the condition is indicated by the corresponding path-available-mask PAM bit being zero when STORE SUBCHANNEL is executed (see "Path-Available Mask (PAM)" on page 15-7). Furthermore, if the channel path is not physically available for the subchannel designated by STORE SUBCHANNEL, then it is not physically available for any subchannel that has a device which is accessible by that channel path.

Unless a dedicated allegiance exists at a subchannel for the channel path, a channel path becomes available for selection if it is logically available and physically available, as indicated by the bits in the LPM and PAM corresponding to the channel path being stored as ones when STORE SUBCHANNEL is executed. If a dedicated allegiance exists at a subchannel for the channel path, only that channel path is available for selection, and the setting of the corresponding LPM bit is ignored. If the channel path is currently being used and a dedicated allegiance exists at the subchannel for the channel path, selection of the device is delayed until the channel path is no longer being used.

The availability status of the eight logical paths to the associated device described in Figure 15-4 on page 15-14 is determined by the hierarchical arrangement of the corresponding bit values contained in the PIM, PAM, and LPM and by existing conditions, if any, recognized by the channel subsystem.

Value of Bit 'n'			Channel-Path Condition ¹	Channel-Path State
PIM	PAM	LPM		
0	0 ²	-	X	Not installed
1	0	-	X	Not physically available
1	1	0 ³	X	Not logically available
1	1	1 ³	Active	Available for selection ⁴
1	1	1	Inactive	Available for selection
Explanation:				
-	Bit value is not meaningful.			
1	If the channel path is recognized as being used in active communication with a device, the channel-path condition is described as active. Otherwise, its condition is described as inactive.			
2	A PAM bit cannot have the value one when the corresponding PIM bit has the value zero.			
3	If a dedicated allegiance exists to the channel path at the subchannel, the state of the bit is ignored, and the channel path is considered to be available for selection.			
4	The channel path may appear to be active when a channel-path-terminal condition has been recognized.			
X	Condition is not meaningful			

Figure 15-4. Path condition and Path-Availability Status for PIM, PAM, and LPM Values

Control-Unit Type

In "Clear Function" on page 15-14, "Halt Function" on page 15-16, and "Start Function and Resume Function" on page 15-20, reference is made to type-1, type-2, and type-3 control units. For a description of these control-unit types, see the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974. For the purposes of this definition, all control units attaching to a serial-I/O interface are considered type-2 control units.

Clear Function

Subsequent to the execution of CLEAR SUBCHANNEL, the channel subsystem performs the clear function. Performance of the clear function consists in (1) performing a path-management operation, (2) modifying fields at the subchannel, (3) issuing the clear signal to the associated device, and (4) causing the subchannel to be made status pending, indicating the completion of the clear function.

Clear-Function Path Management

A path-management operation is performed as part of the clear function in order to examine channel-path conditions for the associated subchannel and to attempt to choose an available channel path on which the clear signal can be issued to the associated device.

Channel-path conditions are examined in the following order:

1. If the channel subsystem is actively communicating or attempting to establish active communication with the device to be signaled, the channel path that is in use is chosen.
2. If the channel subsystem is in the process of accepting a no-longer-busy indication (which will not cause an interruption condition to be recognized) from the device to be signaled, and the associated subchannel has no allegiance to any channel path, the channel path that is in use is chosen.

3. If the associated subchannel has a dedicated allegiance for a channel path, that channel path is chosen.
4. If the associated subchannel has a working allegiance for one or more channel paths, one of those channel paths is chosen.
5. If the associated subchannel has no allegiance for any channel path, if a last-used channel path is indicated, and if that channel path is available for selection, that channel path is chosen. If that channel path is not available for selection, either no channel path is chosen or a channel path is chosen from the set of channel paths, if any, that are available for selection (as though no last-used channel path were indicated).
6. If the associated subchannel has no allegiance for any channel path, if no last-used channel path is indicated, and if there exist one or more channel paths that are available for selection, one of those channel paths is chosen.

If none of the channel-path conditions listed above apply, no channel path is chosen.

For item 4, for item 5 under the specified conditions, and for item 6, the channel subsystem chooses a channel path from a set of channel paths. In these cases, the channel subsystem may attempt to choose a channel path, provided that the following conditions *do not* apply:

1. A channel-path-terminal condition exists for the channel path.
2. For a parallel or ESCON channel path: Another subchannel has an active allegiance for the channel path.

For a FICON channel path: The channel path is currently being used to actively communicate with the maximum number of subchannels that can have concurrent active communications.

3. The device to be signaled is attached to a type-1 control unit, and the subchannel for another device attached to the same control unit has an allegiance to the same channel path, unless the allegiance is a working allegiance and primary status has been accepted by that subchannel.
4. The device to be signaled is attached to a type-3 control unit, and the subchannel for another

device attached to the same control unit has a dedicated allegiance to the same channel path.

Clear-Function Subchannel Modification

Path-management-control indications at the subchannel are modified during performance of the clear function. Effectively, this modification occurs after the attempt to choose a channel path, but prior to the attempt to select the device to issue the clear signal. The path-management-control indications that are modified are as follows:

1. The state of all eight possible channel paths at the subchannel is set to operational for the subchannel.
2. The last-path-used indication is reset to indicate no last-used channel path.
3. Path-not-operational conditions, if any, are reset.

Clear-Function Signaling and Completion

Subsequent to the attempt to choose a channel path and the modification of the path-management-control fields, the channel subsystem, if conditions allow, attempts to select the device to issue the clear signal. (See “Clear Signal” on page 17-11.) Conditions associated with the subchannel and the chosen channel path, if any, affect (1) whether an attempt is made to issue the clear signal, and (2) whether the attempt to issue the clear signal is successful. Independent of these conditions, the subchannel is subsequently set status pending, and the performance of the clear function is complete. These conditions and their effect on the clear function are described as follows:

No Attempt Is Made to Issue the Clear Signal:

The channel subsystem does not attempt to issue the clear signal to the device if any of the following conditions exist:

1. No channel path was chosen. (See “Clear-Function Path Management” on page 15-14.)
2. The chosen channel path is no longer available for selection.
3. A channel-path-terminal condition exists for the chosen channel path.

4. For parallel and ESCON channel paths: The chosen channel path is currently being used to actively communicate with a different device.

For FICON channel paths: The chosen channel path is currently being used to actively communicate with the maximum number of devices that can have concurrent active communications.

5. The device to be signaled is attached to a type-1 control unit, and the subchannel for another device attached to the same control unit has an allegiance to the same channel path, unless the allegiance is a working allegiance and primary status has been accepted by that subchannel.
6. The device to be signaled is attached to a type-3 control unit, and the subchannel for another device attached to the same control unit has a dedicated allegiance to the same channel path.

If any of the conditions above exist, the subchannel remains clear pending and is set status pending, and the performance of the clear function is complete.

The Attempt to Issue the Clear Signal Is Not Successful: When the channel subsystem attempts to issue the clear signal to the device, the attempt may not be successful because of the following conditions:

1. The control unit or device signals a busy condition when the channel subsystem attempts to select the device to issue the clear signal.
2. A path-not-operational condition is recognized when the channel subsystem attempts to select the device to issue the clear signal.
3. An error condition is encountered when the channel subsystem attempts to issue the clear signal.

If any of the conditions above exists and the channel subsystem either determines that the attempt to issue the clear signal was not successful or cannot determine whether the attempt was successful, the subchannel remains clear pending and is set status pending, and the performance of the clear function is complete.

The Attempt to Issue the Clear Signal Is Successful: When the channel subsystem determines that the attempt to issue the clear signal was successful, the subchannel is no longer clear pending and is set

status pending, and the performance of the clear function is complete. When the subchannel becomes status pending, the I/O operation, if any, with the associated device has been terminated.

Programming Note: Subsequent to the performance of the clear function, any nonzero status, except control unit end alone, that is presented to the channel subsystem by the device is passed to the program as unsolicited alert status. Unsolicited status consisting of control unit end alone or zero status is not presented to the program.

Halt Function

Subsequent to the execution of HALT SUBCHANNEL, the channel subsystem performs the halt function. Performance of the halt function consists of (1) performing a path-management operation, (2) issuing the halt signal to the associated device, and (3) causing the subchannel to be made status pending, indicating the completion of the halt function.

Halt-Function Path Management

A path-management operation is performed as part of the halt function to examine channel-path conditions for the associated subchannel and to attempt to choose a channel path on which the halt signal can be issued to the associated device.

Channel-path conditions are examined in the following order:

1. If the channel subsystem is actively communicating or attempting to establish active communication with the device to be signaled, the channel path that is in use is chosen.
2. If the channel subsystem is in the process of accepting a no-longer-busy indication (which will not cause an interruption condition to be recognized) from the device to be signaled, and the associated subchannel has no allegiance to any channel path, the channel path that is in use is chosen.
3. If the associated subchannel has a dedicated allegiance for a channel path, that channel path is chosen.

4. If the associated subchannel has a working allegiance for one or more channel paths, one of those channel paths is chosen.
5. If the associated subchannel has no allegiance for any channel path, if a last-used channel path is indicated, and if that channel path is available for selection, that channel path is chosen. If that channel path is not available for selection, either no channel path is chosen or a channel path is chosen from the set of channel paths, if any, that are available for selection (as though no last-used channel path were indicated).
6. If the associated subchannel has no allegiance for any channel path, if no last-used channel path is indicated, and if there exist one or more channel paths that are available for selection, one of those channel paths is chosen.

If none of the channel-path conditions listed above apply, no channel path is chosen.

For item 4, for item 5 under the specified conditions, and for item 6, the channel subsystem chooses a channel path from a set of channel paths. In these cases, the channel subsystem may attempt to choose a channel path for which the following conditions *do not* apply:

1. A channel-path-terminal condition exists for the channel path.
2. For a parallel or ESCON channel path: Another subchannel has an active allegiance for the channel path.

For a FICON channel path: The channel path is currently being used to actively communicate with the maximum number of subchannels that can have concurrent active communications.

3. The device to be signaled is attached to a type-1 control unit, and the subchannel for another device attached to the same control unit has an allegiance to the same channel path, unless the allegiance is a working allegiance and primary status has been accepted by that subchannel.
4. The device to be signaled is attached to a type-3 control unit, and the subchannel for another device attached to the same control unit has a dedicated allegiance to the same channel path.

Halt-Function Signaling and Completion

Subsequent to the attempt to choose a channel path, the channel subsystem, if conditions allow, attempts to select the device to issue the halt signal. (See “Halt Signal” on page 17-11.)

Conditions associated with the subchannel and the chosen channel path, if any, affect (1) whether an attempt is made to issue the halt signal, (2) whether the attempt to issue the halt signal is successful, and (3) whether the subchannel is made status pending to complete the halt function. These conditions and their effect on the halt function are described as follows:

No Attempt Is Made to Issue the Halt Signal:

The channel subsystem does not attempt to issue the halt signal to the device if any of the following conditions exist:

1. No channel path was chosen. (See “Halt-Function Path Management” on page 15-16.)
2. The chosen channel path is no longer available for selection.
3. A channel-path-terminal condition exists for the chosen channel path.
4. The associated subchannel is status pending with other than intermediate status alone.
5. The device to be signaled is attached to a type-1 control unit, and the subchannel for another device attached to the same control unit has an allegiance to the same channel path, unless the allegiance is a working allegiance and primary status has been accepted by that subchannel.
6. The device to be signaled is attached to a type-3 control unit, and the subchannel for another device attached to the same control unit has a dedicated allegiance to the same channel path.

If the conditions described in items 3, 5, or 6 exist, the associated subchannel remains halt pending until those conditions no longer exist. When the conditions no longer exist (for the channel-path-terminal condition, when the condition no longer exists as a result of executing RESET CHANNEL PATH), the channel subsystem attempts to issue the halt signal to the device.

If any of the remaining conditions above exist, the subchannel remains halt pending and is set status pending, and the halt function is complete.

The Attempt to Issue the Halt Signal Is Not Successful: When the channel subsystem attempts to issue the halt signal to the device, the attempt may not be successful because of the following conditions:

1. The control unit or device signals a busy condition when the channel subsystem attempts to select the device to issue the halt signal.
2. A path-not-operational condition is recognized when the channel subsystem attempts to select the device to issue the halt signal.
3. An error condition is encountered when the channel subsystem attempts to issue the halt signal.

If the control unit or device signals a busy condition (item 1), the subchannel remains halt pending until the internal indication of busy is reset. When this event occurs, the channel subsystem again attempts to issue the halt signal to the device.

If any of the remaining conditions above exists and the channel subsystem either determines that the attempt to issue the halt signal was not successful or cannot determine whether the attempt was successful, then the subchannel remains halt pending and is set status pending, and the halt function is complete.

The Attempt to Issue the Halt Signal Is Successful: When the channel subsystem determines that the attempt to issue the halt signal was successful and ending status, if appropriate, has been received at the subchannel, the subchannel is no longer halt pending and is set status pending, and the halt function is complete. When the subchannel becomes status pending, the I/O operation, if any, with the associated device has been terminated. The conditions that affect the receipt of ending status at the subchannel, and the effect of the halt signal at the device are described in the following discussion.

When the subchannel is subchannel-and-device active or only device active during the performance of the halt function, the state continues until the subchannel is made status pending because (1) the device has provided ending status or (2) the channel

subsystem has determined that ending status is unavailable.

When the subchannel is operating in command mode and is idle, start pending, start pending and resume pending, suspended, or suspended and resume pending, or when the halt signal is issued during command chaining after the receipt of device end but before the next command is transferred to the device, no operation is in progress at the device, and therefore no status is generated by the device as a result of receiving the halt signal. When the subchannel is neither subchannel active, nor status pending with intermediate status, and no errors are detected during the attempt to issue the halt signal to the device, an interruption condition indicating status pending alone is generated after the halt signal is issued.

When the subchannel is operating in transport mode and is start pending, an operation may or may not be in progress at the device. Regardless of whether an operation is in progress, device status is not generated.

The effect of the halt signal at the device depends partially on the subchannel operation mode, type of device, and its state. The effect of the halt signal on a device that is not active or that is performing a mechanical operation in which data is not transferred across the channel path, such as rewinding tape or positioning a disk-access mechanism, depends upon the control-unit or device model. If the device is performing a type of operation that is unpredictable in duration or in which data is transferred across the channel path, the control unit interprets the signal as one to terminate the operation. Pending status conditions at the device are not reset. When the control unit recognizes the halt signal, it immediately ceases all communication with the channel subsystem until it has reached the normal ending point. The control unit then requests selection by the channel subsystem to present any generated status.

If the subchannel is involved in the data-transfer portion of an I/O operation, data transfer is terminated during the performance of the halt function, and the device is logically disconnected from the channel path. If the halt function is addressed to a subchannel operating in command mode, performing a chain of I/O operations, and the device has already provided channel end for the current I/O operation, the channel subsystem causes the device to be discon-

nected and command chaining or command retry to be suppressed.

If the subchannel is operating in command mode, performing a chain of I/O operations with the device, and the halt signal is issued during command chaining at a point after the receipt of device end for the previous I/O operation but before the next command is transferred to the device, the subchannel is made status pending with primary and secondary status immediately after the halt signal is issued. The device-status field of the SCSW contains zeros in this case. If the halt function is addressed to a subchannel that is operating in command mode, start pending, and the halt-pending condition is recognized before initiation of the start function, initiation of the start function is not attempted, and the subchannel becomes status pending after the device has been signaled.

If the halt function is addressed to a subchannel operating in transport mode that is start pending, the halt signal is issued as follows

- If the TCCB has been transported to the I/O device for execution, the halt signal is issued and the subchannel becomes status pending.
- If the TCCB has not been transported to the I/O device, the TCCB is not transported and the subchannel becomes status pending.

When the subchannel is not performing an I/O operation with the associated device, the device is selected, and an attempt is made to issue the halt signal as the device responds. If the subchannel is in the device-active state, the subchannel does not become status pending until it receives the device-end status from the halted device. If the subchannel is neither subchannel-and-device active nor device active, the subchannel becomes status pending immediately after selecting the device and issuing the halt signal. The SCSW for the latter case has the status-pending bit set to one (see “Status-Pending (Bit 31)” on page 16-19).

The termination of an I/O operation by performing the halt function may result in two distinct interruption conditions.

The first interruption condition occurs when the device generates the channel-end condition. The channel subsystem handles this condition as it would any other interruption condition from the device,

except that the command address in the associated SCSW designates the point at which the I/O operation is terminated as follows:

- When a command-mode IRB is stored, the SCSW contains the address of the last-executed CCW, plus 8, and the subchannel-status bits may reflect unusual conditions that were detected.
- When a transport-mode IRB is stored, the SCSW contains the address of the current TCW, and the subchannel-status, subchannel-extended-status, and FCX bits may reflect unusual conditions that were detected.

If the halt signal was issued before all data designated for the operation had been transferred, incorrect length is indicated, subject to the control of the SLI flag in the current when the subchannel is operating in command mode. The value in the count field of the associated SCSW is unpredictable. If the halt signal was issued before all data designated for the operation had been transferred, incorrect length is not indicated, when the subchannel is operating in transport mode.

The second interruption condition occurs if device-end status was not presented with the channel-end interruption condition. In this situation, the subchannel-key, command-address, and count fields of the associated SCSW are not meaningful.

When HALT SUBCHANNEL terminates an I/O operation, the method of termination differs from that used upon exhaustion of count or upon detection of programming errors to the extent that termination by HALT SUBCHANNEL is not contingent on the receipt of a service request from the associated device.

Programming Notes:

1. When, after an operation is terminated by HALT SUBCHANNEL, the subchannel is status pending with primary, primary and secondary, or secondary status, the extent of data transferred as described by the count field is unpredictable.
2. When the path that is chosen by the path-management operation has a channel-path-terminal condition associated with it, the halt function remains pending until the condition no longer exists. Until the condition is cleared, the associated subchannel cannot be used to perform I/O operations, even if other channel paths become

available for selection. CLEAR SUBCHANNEL can be executed to terminate the halt-pending condition and make the subchannel usable.

Start Function and Resume Function

The start and resume functions initiate I/O operations as described below. The start function applies to subchannels operating in either command mode or transport mode. The resume function applies only to subchannels operating in command mode.

Subsequent to the execution of START SUBCHANNEL and RESUME SUBCHANNEL, the channel subsystem performs the start and resume functions, respectively, to initiate an I/O operation with the associated device. Performance of a start or resume function consists of: (1) performing a path-management operation, (2) performing an I/O operation or chain of I/O operations with the associated device, and (3) causing the subchannel to be made status pending, indicating the completion of the start function. (Completion of a start function is described in “I/O Interruptions” on page 16-1.) The start function initiates the execution of a channel program that is designated in the ORB, which in turn is designated as the operand of START SUBCHANNEL, in contrast to the resume function that initiates the execution of a suspended channel program, if any, beginning at the CCW that caused suspension; otherwise, the resume function is performed as if it were a start function (see “Resume-Pending (Bit 20)” on page 16-14).

Start-Function and Resume-Function Path Management

A path-management operation is performed by the channel subsystem during the performance of either a start or a resume function to choose an available channel path that can be used for device selection to initiate an I/O operation with that device. The actions taken are as follows:

1. If the subchannel is currently start pending and device active, the start function remains pending at the subchannel until the secondary status for the previous start function has been accepted from the associated device and the subchannel is made start pending alone. When the status is

accepted and does not describe an alert interruption condition, the subchannel is not made status pending, and the performance of the pending start function is subsequently initiated. If the status describes an alert interruption condition, the subchannel becomes status pending with secondary and alert status, the pending start function is not initiated, deferred condition code 1 is set, and the start-pending bit remains one. If the subchannel is currently start pending alone, the performance of the start function is initiated as described below.

2. If a dedicated allegiance exists at the subchannel for a channel path, the channel subsystem chooses that path for device selection. If a busy condition is encountered while attempting to select the device and a dedicated allegiance exists at the subchannel, the start function remains pending until the internal indication of busy is reset for that channel path. When the internal indication of busy is reset, the performance of the pending start function is initiated on that channel path.
3. If no channel path is available for selection and no dedicated allegiance exists in the subchannel for a channel path, a channel path is not chosen.
4. If all channel paths that are available for selection have been tried and one or more of them are being used to actively communicate with other devices, or, alternatively, if the channel subsystem has encountered either a control-unit-busy or a device-busy condition on one or more of those channel paths, or a combination of those conditions on one or more of those channel paths, the start function remains pending at the subchannel until a channel path, control unit, or device, as appropriate, becomes available.
5. If (1) the start function is to be initiated on a channel path with a device attached to a type-1 control unit and (2) no other device is attached to the same control unit whose subchannel has either a dedicated allegiance to the same channel path or a working allegiance to the same channel path where primary status has not been received for that subchannel, then that channel path is chosen if it is available for selection; otherwise, that channel path is not chosen. If, however, another channel path to the device is available for selection and no allegiances exist as described above, that channel path is chosen. If no other channel path is available for selection,

the start or resume function, as appropriate, remains pending until a channel path becomes available.

6. If the device is attached to a type-3 control unit, and if at least one other device is attached to the same control unit whose subchannel has a dedicated allegiance to the same channel path, another channel path that is available for selection may be chosen, or the start function remains pending until the dedicated allegiance for the other device is cleared.
7. If a channel path has been chosen and a busy indication is received during device selection to initiate the execution of the first command of a pending CCW channel program or to transport the TCCB of a pending TCW channel program, the channel path over which the busy indication is received is not used again for that device or control unit (depending on the device-busy or control-unit-busy indication received) until the internal indication of busy is reset.
8. If, during an attempt to select the device in order to initiate the execution of the first command specified for the start or implied for the resume function for a CCW channel program, or to initiate the transportation of the TCCB for the start function for a TCW channel program, (as described in action 7 above), the channel subsystem receives a busy indication, it performs one of the following actions:
 - a. If the device is specified to be operating in the multipath mode and the busy indication received is device busy, then the start or resume function remains pending until the internal indication of busy is reset. (See "Multipath Mode (D)" on page 15-4.)
 - b. If the device is specified to be operating in the multipath mode and the busy indication received is control unit busy, or if the device is specified to be operating in the single-path mode, the channel subsystem attempts selection of the device by choosing an alternate channel path that is available for selection and continues the path-management operation until either the start or the resume function is initiated or selection of the device has been attempted on all channel paths that are available for selection. If the start or resume function has not been initiated by the channel subsystem after all channel paths available for selection have been chosen, the start or resume function remains pending until the internal indication of busy is reset.
 - c. If the subchannel has a dedicated allegiance, then action 2 on page 15-20 applies.
9. When, during the selection attempt to transfer the first command for a CCW channel program, or to transport the TCCB for a TCW channel program, the device appears not operational and the corresponding channel path is operational for the subchannel, a path-not-operational condition is recognized, and the state of the channel path changes at the subchannel from operational for the subchannel to not operational for the subchannel (see "Path-Not-Operational Mask (PNOM)" on page 15-5). The path-not-operational conditions at the subchannel, if any, are preserved until the subchannel next becomes clear pending, start pending, or resume pending (if the subchannel was suspended), at which time the path-not-operational conditions are cleared. If, however, the corresponding channel path is not operational for the subchannel, a path-not-operational condition is not recognized. When the device appears not operational during the selection attempt to transfer the first command or TCCB on a channel path that is available for selection, one of the following actions occurs:
 - a. If a dedicated allegiance exists for that channel path, then it is the only channel path that is available for selection; therefore, further attempts to initiate the start or resume function are abandoned, and an interruption condition is recognized.
 - b. If no dedicated allegiance exists and there are alternate channel paths available for selection that have not been tried, one of those channel paths is chosen to attempt device selection and transfer the first command for a CCW channel program, or the TCCB for a TCW channel program.
 - c. If no dedicated allegiance exists, no alternate channel paths are available for selection that have not been tried, and the device has appeared operational on at least one of the channel paths that were tried, the start or resume function remains pending at the subchannel until a channel path, a control unit,

or the device, as appropriate, becomes available.

- d. If no dedicated allegiance exists, no alternate channel paths are available for selection that have not been tried, and the device has appeared not operational on all channel paths that were tried, further attempts to initiate the start or resume function are abandoned, and an interruption condition is recognized.

10. When the subchannel is active and an I/O operation is to be initiated with a device, all device selections occur according to the LPUM indication if the multipath mode is not specified at the subchannel. For example, if command chaining is specified for a CCW channel program, the channel subsystem transfers the first and all subsequent commands describing a chain of I/O operations over the same channel path.

Interrogate Function

Subsequent to the execution of CANCEL SUBCHANNEL for a subchannel operating in transport mode, the channel subsystem performs the interrogate function. Performance of the interrogate function consists of (1) transporting the designated interrogate TCCB to the I/O device, (2) storing the interrogate TSB, and (3) causing the subchannel to become intermediate status pending with interrogate-complete status indicated.

All storage accesses used to perform the interrogate operation are made as if a storage-access key of zero is used.

Interrogate-Function Path Management

The same channel path that was used to transport the TCCB for the I/O operation being interrogated is used for the interrogate function.

Interrogate TCCB Transportation and Completion

Subsequent to determining the channel path, the channel subsystem, if conditions allow, attempts to transport the interrogate TCCB. Conditions associ-

ated with the subchannel, the interrogate TCCB, and the chosen channel path, if any, affect (1) whether an attempt is made to transport the interrogate TCCB, (2) whether the attempt to transport the interrogate TCCB is successful, and (3) whether the interrogate TSB is stored. These conditions and their effect on the interrogate function are described as follows:

No Attempt Is Made to Transport the Interrogate TCCB: The channel subsystem does not transport the interrogate TCCB if any of the following conditions exists:

- The subchannel is status pending with primary status indicated.
- The subchannel is idle or device active only.
- The TCCB for the I/O operation being interrogated has not been transported to the device.
- Another interrogate function is already in progress on the channel path for the I/O device.
- The interrogate-TCW-address field of the TCW for the I/O operation being interrogated contains zeros.
- The channel subsystem doesn't have the resources to process the interrogate.

The Attempt to Transport the Interrogate TCCB or Store the Interrogate TSB is Unsuccessful:

When the channel subsystem attempts to transport the interrogate TCCB, the attempt may not be successful if any of the following conditions exists:

- The interrogate-TCW address designated by the TCW of the I/O operation being interrogated designates an unavailable storage location or does not designate the interrogate TCW on a 64-byte boundary.
- The TCCB-TIDA flag is one in the flags field of the interrogate TCW and the TCCB-address field in the interrogate TCW does not designate the first TIDAW on a quadword boundary.
- The interrogate-TSB address designated by the TCW of the I/O operation being interrogated designates an unavailable storage location or does not designate the interrogate TSB on a doubleword boundary.

If any of the above conditions is true, the subchannel is made status pending with program check and

interrogate-failed status indicated in the subchannel-extended-status field of the IRB.

The attempt to transport the interrogate TCCB is also not successful when a device-detected program check is recognized. In this case, interrogate-failed status is also indicated in the subchannel-extended-status field of the IRB.

If the interrogate TCCB has been sent and the corresponding interrogate response is not received within a model-dependent amount of time, an interface-control check is recognized with interrogate-failed status indicated in the subchannel-extended-status field of the IRB.

The Attempt to Transport the Interrogate TCCB is Successful: When the channel subsystem determines that the attempt to transport the interrogate TCCB was successful and the interrogate data has been received at the subchannel before the completion of the I/O operation being interrogated, the interrogate data is stored into the interrogate TSB, and the subchannel is made status pending with intermediate status and interrogate-complete status indicated. The status related to the operation being interrogated remains unchanged.

When the channel subsystem determines that the operation being interrogated has completed before the interrogate data has been received at the subchannel, the channel subsystem waits until the interrogate data is received. When received, the interrogate data is stored into the interrogate TSB, and the subchannel is made status pending with primary status indicated. The status related to the operation being interrogated is unchanged.

Execution of I/O Operations

After a channel path is chosen, the channel subsystem, if conditions allow, initiates the execution of an I/O operation with the associated device. Execution of additional I/O operations may follow the initiation and execution of the first I/O operation.

For subchannels operating in command mode, the channel subsystem can execute seven types of commands: write, read, read backward, control, sense, sense ID, and transfer in channel. Each command, except transfer in channel, initiates a corresponding I/O operation. Except for periods when channel-pro-

gram execution is suspended at the subchannel (see “Suspension of CCW Channel-Program Execution” on page 15-68), the subchannel is active from the acceptance of the first command until the primary interruption condition is recognized at the subchannel. If the primary interruption condition is recognized before the acceptance of the first command, the subchannel does not become active. Normally, the primary interruption condition is caused by the channel-end signal or, in the case of command chaining, the channel-end signal for the last CCW of the chain. (See “Primary Interruption Condition” on page 16-4.) The device is active until the secondary interruption condition is recognized at the subchannel. Normally, the secondary interruption condition is caused by the device-end signal or, in the case of command chaining, the device-end signal for the last CCW of the chain. (See “Secondary Interruption Condition” on page 16-5.)

For subchannels operating in transport mode, the channel subsystem can transport six types of commands for execution: write, read, control, sense, sense ID, and interrogate. Each command initiates a corresponding device operation. When one or more commands are transported to the I/O device in a TCCB, the subchannel remains start pending until primary status is presented.

Programming Notes:

In the single-path mode, all transfers of commands, data, and status for the I/O operation or chain of I/O operations occur on the channel path over which the first command was transferred to the device.

When the device has the dynamic-reconnection feature installed, an I/O operation or chain of I/O operations may be performed in the multipath mode. To operate in the multipath mode, MODIFY SUBCHANNEL must have been previously executed for the subchannel with bit 13 of word 1 of the SCHIB specified as one. (See “Multipath Mode (D)” on page 15-4.) In addition, the device must be set up for the multipath mode by the execution of certain model-dependent commands appropriate to that type of device. The general procedures for handling multipath-mode operations are as follows:

1. Setup
 - a. A set-multipath-mode type of command must be successfully executed by the device on each channel path that is to be a member of

the multipath group being set up; otherwise, the multipath mode of operation may give unpredictable results at the subchannel. If, for any reason, one or more physically available channel paths to the device are not included in the multipath group, these channel paths must not be available for selection while the subchannel is operating in the multipath mode. A channel path can be made not available for selection by having the corresponding LPM bit set to zero either in the SCHIB prior to the execution of MODIFY SUBCHANNEL or in the ORB prior to the execution of START SUBCHANNEL.

- b. When a set-multipath-mode type of command is transferred to a device, only a single channel path must be logically available in order to avoid alternate channel-path selection for the execution of that start function; otherwise, device-busy conditions may be detected by the channel subsystem on more than one channel path, which may cause unpredictable results for subsequent multipath-mode operations. This type of setup procedure should be used whenever the membership of a multipath group is changed.

2. Leaving the Multipath Mode

To leave the multipath mode and continue processing in the single-path mode, either of the following two procedures may be used:

- a. A disband-multipath-mode type of command may be executed for any channel path of the multipath group. This command must be followed by either (1) the execution of MODIFY SUBCHANNEL with bit 13 of word 1 of the SCHIB specified as zero, or (2) the specification of only a single channel path as logically available in the LPM. A start function must not be performed at a subchannel operating in the multipath mode with multiple channel paths available for selection while the device is operating in single-path mode; otherwise, unpredictable results may occur at the subchannel for that function or subsequent start functions.
- b. A resign-multipath-mode type of command is executed on each channel path of the multipath group (the reverse of the setup described in item 1 on page 15-23). This

command must be followed by either (1) the execution of MODIFY SUBCHANNEL with bit 13 of word 1 of the SCHIB specified as zero, or (2) the specification of only a single channel path as logically available in the LPM. No start function may be performed at a subchannel operating in the multipath mode with multiple channel paths available for selection while the device is operating in single-path mode; otherwise, unpredictable results may occur at the subchannel for that or subsequent start functions.

Blocking of Data

Data recorded by an I/O device is divided into blocks. The length of a block depends on the device; for example, a block can be a card, a line of printing, or the information recorded between two consecutive gaps on magnetic tape.

The maximum amount of information that can be transferred in one I/O operation is one block. An I/O operation is terminated when the associated main-storage area is exhausted or the end of the block is reached, whichever occurs first. For some operations, such as writing on a magnetic-tape unit or at an inquiry station, blocks are not defined, and the amount of information transferred is controlled only by the program.

Operation-Request Block

The operation-request block (ORB) is the operand of START SUBCHANNEL. The ORB specifies the parameters to be used in controlling that particular start function. These parameters include the interruption parameter, the subchannel key, the address of first CCW or the TCW, operation-control bits, priority-control numbers, and a specification of the logical availability of channel paths to the designated device.

The contents of the ORB are placed at the designated subchannel during the execution of START SUBCHANNEL, prior to the setting of condition code 0. If the execution will result in a nonzero condition code, the contents of the ORB are not placed at the designated subchannel.

The two rightmost bits of the ORB address must be zeros, placing the ORB on a word boundary; otherwise, a specification exception is recognized.

When the fibre-channel-extensions (FCX) facility is installed, the channel-program-type control (B) (word 1, bit 13) of the ORB specifies the type of channel program that is designated by the ORB. When B is zero, the ORB designates a CCW channel program. When the B is one, the ORB designates a TCW channel program. Only I/O-devices that support FCX recognize TCW channel programs.

If the contents of an ORB that designates a CCW channel program are placed at the designated subchannel during the execution of START SUBCHANNEL, the subchannel remains in command mode. Thus, such an ORB is also known as a command-mode ORB. If the contents of an ORB that designates a TCW channel program are placed at the designated subchannel during execution of START SUBCHANNEL, the subchannel enters transport mode. Thus, such an ORB is also known as a transport-mode ORB.

Command-Mode ORB

This section describes the command-mode ORB. The term operation-request block (ORB), in this section, means an ORB in which bit 13 of word 1 (the channel-program-type control (B)) is zero.

The format of the command-mode ORB is as follows:

Word																								
0	Interruption Parameter																							
1	Key	S	C	M	Y	F	P	I	A	U	B	H	T	LPM			L	D	0	0	0	0	0	X
2	Channel-Program Address																							
3	CSS Priority				Reserved				CU Priority				Reserved											
4	Reserved																							
5	Reserved																							
6	Reserved																							
7	Reserved																							
	0	8				16				24				31										

Figure 15-5. Command-Mode Operation-Request Block

The fields in the command-mode ORB are defined as follows:

Interruption Parameter: Bits 0-31 of word 0 are preserved unmodified in the subchannel until replaced by a subsequent START SUBCHANNEL or MODIFY SUBCHANNEL instruction. These bits are placed in word 1 of the interruption code when an I/O interruption occurs and when an interruption request

is cleared by the execution of TEST PENDING INTERRUPTION.

Subchannel Key: Bits 0-3 of word 1 form the subchannel key for all fetching of CCWs, IDAWs, MIDAWS, and output data and for the storing of input data associated with the start function initiated by START SUBCHANNEL. This key is matched with a storage key during these storage references. For details, see the section “Key-Controlled Protection” on page 3-10.

Suspend Control (S): Bit 4 of word 1 controls the performance of the suspend function for the channel program designated in the ORB. The setting of the S bit applies to all CCWs of the channel program designated by the ORB (see “Commands and Flags for CCWs” on page 15-70).

When bit 4 is one, suspend control is specified, and channel-program suspension occurs when a suspend flag set to one is detected in a CCW. When bit 4 is zero, suspend control is not specified, and the presence of a suspend flag set to one in any CCW of the channel program causes a program-check condition to be recognized.

Streaming-Mode Control (C): Bit 5 of word 1 controls streaming-mode enablement for subchannels configured to FICON-converted-I/O-interface channel paths during performance of the specified start function. When bit 5 is zero, the streaming mode is enabled at the subchannel. When bit 5 is one, the streaming mode is disabled at the subchannel. Bit 5 is meaningful only for subchannels configured to FICON-converted-I/O-interface channel paths and is ignored for subchannels configured to other channel-path types.

When the streaming mode is enabled, the channel path considers the first command of the designated channel program to be in progress at the associated device when the channel path receives the indication that the command has been accepted at the device. In addition, the channel path's acceptance of status, under certain conditions, is recognized by the channel path without receiving acknowledgement of status acceptance from the device.

When the streaming mode is not enabled, the channel path does not consider the first command of the designated channel program to be in progress at the associated device until the appropriate channel-path response, indicating that the device-command

response or status has been accepted at the channel path, is sent to the device. In addition, when the device sends status to the channel path, the channel path's acceptance of that status is not recognized at the channel path until the channel path's confirmation of acceptance is received and acknowledged by the device.

Modification Control (M): Bit 6 of word 1 specifies whether modification control is required for the channel program. When bit 6 is zero, modification control is specified. When bit 6 is one, modification control is not specified.

When modification control is specified, the channel subsystem forces command synchronization with the addressed I/O device each time a command is executed and the previously executed command has the PCI and chain-command flags set to one and the chain-data and suspend flags set to zero. When this condition is recognized, the channel subsystem signals a synchronization request to the I/O device for the current command. The channel subsystem temporarily suspends command chaining and does not fetch (or refetch) the next command-chained CCW until after normal ending status is received for the synchronizing command.

When modification control is not specified, then command synchronization is not required, and the channel subsystem may transfer commands to the I/O device without waiting for status.

The M bit is meaningful only for subchannels configured to FICON-I/O-interface or FICON-converted-I/O-interface channel paths and is ignored for other subchannels configured to other channel-path types.

Programming Notes:

1. For FICON-I/O-interface and FICON-converted-I/O-interface channel paths, modification control provides the capability to optimize dynamically modified channel programs that use the PCI flag in the CCW to initiate channel-program modification. Specifically, it allows the program to delay the channel-subsystem fetching and transferring of commands until after status is received for the command following the command with the PCI bit set. This increases the likelihood that a program-controlled interruption will be accepted by a CPU and acted upon by the program that dynamically modifies one or more command-chained CCWs that follow the synchronizing command.

In order to increase the probability that any dynamically modified CCWs are fetched after their modification and not prior to their modification, the modifying program should be executed as soon as possible following the CPU's acceptance of the program-controlled interruption. Additionally, the program should minimize the periods during which the configured CPUs are disabled for I/O interruptions.

For channel paths other than FICON-I/O-interface or FICON-converted-I/O-interface channel paths, command synchronization is implicit in the signalling protocol between the channel subsystem and the I/O device; therefore no explicit programming action is required to force command synchronization. Regardless, the program should still attempt to accept and process program-controlled interruptions for these channel-path types in as timely a manner as possible for the same reason as stated above.

2. In order to allow the channel subsystem to optimize the execution of channel programs for FICON-I/O-interface or FICON-converted-I/O-interface channel paths, use of the modification-control facility is discouraged except for channel programs that require dynamic modification.

Synchronization Control (Y): Bit 7 of word 1 specifies whether synchronization control is required for the channel program. When bit 7 is zero and the prefetch-control bit, bit 9 of word 1, is one, synchronization control is specified. When bit 7 is one and bit 9 is one, synchronization control is not specified.

When synchronization control is specified, the channel subsystem forces command synchronization with the addressed I/O device whenever the current command in execution describes an input operation and the next CCW to be fetched describes an output operation. When this condition is recognized, the channel subsystem signals a synchronization request to the I/O device when the input command is transferred. The transfer of the output command is held pending at the subchannel until normal ending status, signaling the completion of the performance of the input operation by the I/O device, is received. Upon receipt of the ending status, the channel subsystem fetches (or refetches) the data associated with the output command and transfers it to the I/O device.

When synchronization control is not specified, the channel subsystem may transfer commands of the channel program without awaiting status that would signal the completion of the I/O operation for each command.

The Y bit is meaningful only when the subchannel is configured to FICON-I/O-interface or FICON-converted-I/O-interface channel paths and the prefetch-control bit, bit 9 of word 1, is one. The Y bit is ignored for subchannels configured to other channel-path types and when the prefetch-control bit is zero.

Format Control (F): Bit 8 of word 1 specifies the format of the channel-command words (CCWs) that make up the channel program designated by the channel-program-address field. When bit 8 of word 1 is zero, format-0 CCWs are specified. When bit 8 is one, format-1 CCWs are specified. (See “Channel-Command Word” on page 15-31 for the definition of the CCW formats.)

Prefetch Control (P): Bit 9 of word 1 specifies whether or not unlimited prefetching of CCWs is allowed for the channel program. When bit 9 is one, unlimited prefetching of CCWs is allowed. (Unlimited prefetching of data, IDAWs, and MIDAWs, associated with the current and prefetched CCWs is always allowed.) It is model dependent whether prefetching is actually performed.

When bit 9 of word 0 is zero, no prefetching is allowed, except in the case of data chaining on output, where the prefetching of one CCW describing a data area is allowed. When bit 9 of word 0 is zero, the synchronization-control bit, bit 7 of word 1, is ignored.

Additional controls may limit the scope of prefetching.

Initial-Status-Interrupt Control (I): Bit 10 of word 1 specifies whether or not the channel subsystem must verify to the program that the device has accepted the first command associated with a start or resume function. When the I bit is specified as one in the ORB, then, when the subchannel becomes active, indicating that the first command has been accepted for this start or resume function, the Z bit (see “Zero Condition Code (Z)” on page 16-12) is set to one at this subchannel, and the subchannel becomes status pending with intermediate status.

If the subchannel does not become active — for example, when the device signals channel end immediately upon receiving the first command, com-

mand chaining is not specified in the CCW, and command retry is not signaled — the command-accepted condition (Z bit set to one) is not generated; instead, the subchannel becomes status pending with primary status. Intermediate status may also be indicated in this case when the command is accepted if the first CCW contained the PCI flag set to one.

Address-Limit-Checking Control (A): When the address-limit-checking facility is installed, bit 11 of word 1 specifies whether or not address-limit checking is specified for the channel program. When this bit is zero, no address-limit checking is performed for the execution of the channel program, independent of the setting of the limit-mode bits in the subchannel (see “Limit Mode (LM)” on page 15-3). When this bit is one, address-limit checking is allowed for the channel program, subject to the setting of the limit-mode bits in the subchannel.

When the address-limit-checking facility is not installed, the address-limit-checking-control bit (A) must be zero in the ORB when START SUBCHANNEL is executed; otherwise, an operand exception is recognized.

Suppress-Suspended-Interrupt Control (U): Bit 12 of word 1, when one, specifies that the channel subsystem is to suppress the generation of an intermediate interruption condition due to suspension if the subchannel becomes suspended. When bit 12 is zero, the channel subsystem generates an intermediate interruption condition whenever the subchannel becomes suspended during the execution of the channel program.

Channel-Program Type Control (B): Bit 13 of word 1 specifies the type of channel program that is designated by the channel-program-address field. When bit 13 is zero, the channel-program-address field designates a CCW channel program. When bit 13 is one, the channel-program-address field designates a TCW channel program. If the FCX facility is not installed and this bit is one, an operand exception or program-check exception is recognized.

Bit 13 also designates the layout of the ORB. When bit 13 is zero, a command-mode ORB is specified. When bit 13 is one, a transport-mode ORB is specified.

Format-2-IDAW Control (H):

Bit 14 of word 1 specifies the format of IDAWs for CCWs that specify indirect data addressing. When

bit 14 of word 1 is one, format-2 (64-bit data address) IDAWs are provided for all CCWs that have the IDAW flag set to one in the CCW. When bit 14 of word 1 is zero, format-1 (31-bit data address) IDAWs are provided for all CCWs that have the IDAW flag set to one in the CCW.

Programming Note: Format-2 IDAWs and MIDAWs provide the only means by which data can be transferred directly between an I/O device and storage locations with addresses greater than 2G bytes.

2K-IDAW Control (T): Bit 15 of word 1 specifies the main-storage block size for format-2-IDAW data areas. Bit 15 is meaningful only when bit 14 (format-2 IDAW control) is one and is ignored when bit 14 is zero. When bit 15 of word 1 is one, all format-2 IDAWs designate 2K-byte storage blocks. When bit 15 of word 1 is zero, all format-2 IDAWs designate 4K-byte storage blocks.

Logical-Path Mask (LPM): Bits 16-23 of word 1 are preserved unmodified in the subchannel and specify to the channel subsystem which of the logical paths 0-7 are to be considered logically available, as viewed by the program. A bit setting of one means that the corresponding channel path is logically available; a zero specifies that the corresponding channel path is logically not available. If a channel path is specified by the program as being logically not available, the channel subsystem does not use that channel path to perform clear, halt, resume, or start functions when requested by the program, except when a dedicated-allegiance condition exists for that channel path. If a dedicated-allegiance condition exists, the setting of the LPM is ignored, and a resume, start, halt, or clear function is performed by using the channel path having the dedicated allegiance.

Incorrect-Length-Suppression Mode (L): When bit 8 of word 1 is one, then bit 24 of word 1, when one, specifies the incorrect-length-suppression mode. When the subchannel is in this mode when an immediate operation occurs (that is, when a device signals the channel-end condition during initiation of the command) and the current CCW contains a non-zero value in bit positions 16-31, indication of an incorrect-length condition is suppressed.

When bit 8 of word 1 is one, then bit 24 of word 1, when zero, specifies the incorrect-length-indication mode. When the subchannel is in this mode when an immediate operation occurs (that is, when a device

signals the channel-end condition during initiation of the command) and the current CCW contains a non-zero value in bit positions 16-31, indication of an incorrect-length condition is recognized. Command chaining is suppressed unless the SLI flag in the CCW is one and the chain-data flag is zero.

When bit 8 of word 1 is zero, the value of bit 24 is ignored by the channel subsystem, and the subchannel is in the incorrect-length-suppression mode.

Modified-CCW-Indirect-Data-Addressing Control (D): When the modified-CCW-indirect-data-addressing facility is installed, bit 25 of word 1 specifies whether the channel program may include CCWs that specify modified-CCW-indirect-data-address lists. When bit 25 of word 1 is one, the channel program may include CCWs that specify lists of quad-words each called a modified-CCW-indirect-data-address word (MIDAW). When bit 25 of word 1 is zero, the channel program may not include CCWs that specify modified-CCW-indirect-data-address lists.

When the modified-CCW-indirect-data-addressing facility is not installed, the modified-CCW-indirect-data-addressing control must be zero in the ORB when START SUBCHANNEL is executed; otherwise, an operand exception is recognized.

ORB-Extension Control (X): Bit 31 of word 1 specifies whether the ORB is extended. When bit 31 of word 1 is zero, the ORB consists of words 0-2, and words 3-7 are ignored. When bit 31 of word 1 is one, the ORB consists of words 0-7. Words 0 and 1 are described above. Words 2-7 are described below.

Reserved: Bits 26-30 of word 1 are reserved for future use and must be set to zeros. Bit 31 of word 1 must be zero if the ORB-extension facility is not installed. Otherwise, an operand exception or program-check condition is recognized.

Channel-Program Address: Bits 1-31 of word 2 specify the absolute address of the first CCW in main storage. Bit 0 of word 2 must be zero; otherwise, either an operand exception or a program-check condition is recognized. If format-0 CCWs are specified by bit 8 of word 1, then bits 1-7 of word 2 also must be zeros; otherwise, a program-check condition is recognized.

The three rightmost bits of the channel-program address must be zeros, designating the CCW on a

doubleword boundary; otherwise, a program-check condition is recognized.

If the channel-program address designates a location protected against fetching or designates a location outside the storage of the particular configuration, the start function is not initiated at the device. In this situation, the subchannel becomes status pending with primary, secondary, and alert status.

Channel-Subsystem (CSS) Priority: When bit 31 (X) of word 1 of the ORB is one, byte 0 of word 3 contains an unsigned binary integer, called the channel-subsystem-priority number, that is assigned to the designated subchannel and used to order the selection of subchannels when either a start function or a resume function is to be initiated for one or more subchannels that are start pending or resume pending.

The specified channel-subsystem-priority number can be any number in the range of 0 to 255. The numbers 0 and 255 designate the lowest and highest priorities, respectively.

Depending on the model and the configuration:

1. Fewer than 256 priority levels may be provided. For such models, the ORB-specified priority number may be ignored, and an alternative priority number may be implicitly assigned to the subchannel when the subchannel becomes start pending.
2. When bit 31 (X) of word 1 of the ORB is zero, an implicit priority number is assigned to the subchannel.

See "Channel-Subsystem-I/O-Priority Facility" on page 17-29 for details about how the priority number is assigned for both of these cases.

Control-Unit (CU) Priority: When bit 31 (X) of word 1 of the ORB is one, byte 2 of word 3 contains an unsigned binary integer, called the control-unit-priority number, that specifies, for an associated control unit attached by a FICON channel path, the priority level that is applied at the associated control unit for all I/O operations associated with the start function.

The specified control-unit-priority number can be any integer in the range of 1 to 255. The numbers 1 and 255 designate the lowest and highest priorities,

respectively. The number 0 designates that no priority is assigned to the I/O operations associated with the start function. The handling of I/O operations when the priority number is 0 depends on the control-unit model.

Also depending on the control-unit model, fewer than 255 priority levels may be supported by the control unit. See the control-unit's System Library publication for additional information regarding the range of priority numbers supported and how this priority number is used.

The specified control-unit-priority number is ignored if any of the following conditions exists:

1. Bit 31 (X) of word 1 is zero. In this case, a control-unit-priority number of 0 is transmitted in the associated outbound frames.
2. The designated subchannel is not associated with a control unit configured to a FICON channel path.
3. The associated control unit does not provide prioritized performance of I/O operations. In this case, the control-unit-priority number in the associated outbound frames is ignored at the control unit.
4. The channel-subsystem model does not provide for the transmission of the control-unit-priority number.
5. The channel-subsystem-I/O-priority facility is not operational due to an operator action.

Reserved: All fields in the ORB that are defined as either "0" or "Reserved" must contain zeros when START SUBCHANNEL is executed; otherwise, either an operand exception or a program-check condition is recognized.

Programming Notes:

1. Bit positions of the ORB that presently are specified to contain zeros may in the future be assigned for the control of new functions.
2. The interruption parameter may contain any information, but ordinarily the information is of significance to the program handling the I/O interruption.

Transport-Mode ORB

This section describes the transport-mode ORB. The term operation-request block (ORB), in this section, means an ORB in which bit 13 of word 1 (the channel-program-type control (B)) is one.

The format of the transport-mode ORB is as follows:

Word																					
0	Interruption Parameter																				
1	Key	0	0	0	0	0	0	0	0	0	B	0	0	LPM	0	0	0	0	0	0	X
2	Channel-Program Address																				
3	CSS Priority				Reserved				Rsv. for Pgm.				Reserved								
4	Reserved																				
5	Reserved																				
6	Reserved																				
7	Reserved																				
	0	8				16				24				31							

Figure 15-6. Transport-Mode Operation-Request Block

Interruption Parameter: This field has the same meaning as in the ORB that designates a CCW channel program. See “Interruption Parameter” on page 15-2.

Subchannel Key: Bits 0-3 of word 1 form the subchannel key for all fetching of TCWs, TCCBs, TIDAWS, and output data and for the storing of input data associated with the start function initiated by START SUBCHANNEL. This key is matched with a storage key during these storage references. For details, see the section “Key-Controlled Protection” on page 3-10.

All storage accesses to the TSB are made as if the subchannel key is zero.

Reserved: All fields in the ORB that are defined as either “0” or “Reserved” must contain zeros when START SUBCHANNEL is executed; otherwise, either an operand exception or a program-check condition is recognized.

Channel-Program Type Control (B): This bit has the same meaning as in the ORB that designates a CCW channel program. For the transport-mode ORB, the value of this bit is one. See “Channel-Program Type Control (B)” on page 15-27.

Logical-Path Mask (LPM): Bits 16-23 of word 1 are preserved unmodified in the subchannel and specify to the channel subsystem which of the logical paths 0-7 are to be considered logically available, as viewed by the program. A bit setting of one means that the corresponding channel path is logically available; a zero specifies that the corresponding channel path is logically not available. If a channel path is specified by the program as being logically not available, the channel subsystem does not use that channel path to perform clear, halt, or start functions when requested by the program.

If the LPM specifies a channel path that does not support FCX, either an operand-exception or a program check may be recognized.

ORB-Extension Control (X): This field has the same meaning as in the ORB that designates a CCW channel program. See “ORB-Extension Control (X)” on page 15-28.

Channel-Program Address: Bits 1-31 of word 2 specify the absolute address of the TCW in main storage. Bit 0 of word 2 must be zero; otherwise, either an operand exception or a program-check condition is recognized.

The six rightmost bits of the channel-program address must be zeros, designating the TCW on a 64-byte boundary; otherwise, a program-check condition is recognized.

If the channel-program address designates a location protected against fetching or designates a location outside the storage of the particular configuration, the start function is not initiated at the device. In this situation, the subchannel becomes status pending with primary, secondary, and alert status.

Channel-Subsystem (CSS) Priority: This field has the same meaning as in the ORB that designates a CCW channel program. See “Channel-Subsystem (CSS) Priority” on page 15-29

Reserved for Program Use (Rsv. for Pgm.): Byte 2 of word 3 is reserved for program use and is not checked or used by the channel subsystem.

Programming Note: See the programming notes on page 15-29.

Channel-Command Word

The channel-command word (CCW) specifies the command to be executed and, for commands initiating certain I/O operations, it designates the storage area associated with the operation, the action to be taken whenever transfer to or from the area is completed, and other options.

A channel program consists of one or more CCWs that are logically linked such that they are fetched and executed by the channel subsystem in either a sequential or a nonsequential order. Sequential (contiguous) CCWs are linked by the use of the chain-data and chain-command flags, and nonsequential (noncontiguous) CCWs are linked by a CCW specifying the transfer-in-channel command.

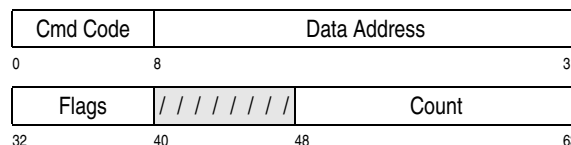
As each CCW is executed, it is recognized as the current CCW. A CCW becomes current (1) when it is the first CCW of a channel program and has been fetched, (2) when, during command chaining, the new CCW is logically fetched, or (3) when, during data chaining, the new CCW takes over control of the I/O operation (see “Data Chaining” on page 15-56). When chaining is not specified, a CCW is no longer current after TEST SUBCHANNEL clears the start-function bit in the subchannel.

The location of the first CCW of the channel program is designated in the ORB that is the operand of START SUBCHANNEL. The first CCW is fetched subsequent to the execution of the instruction. The format of the CCWs fetched by the channel subsystem is specified by bit 8 of word 1 of the ORB. Each additional CCW in the channel program is obtained when the CCW is needed. Fetching of the CCWs by the channel subsystem does not affect those locations in main storage.

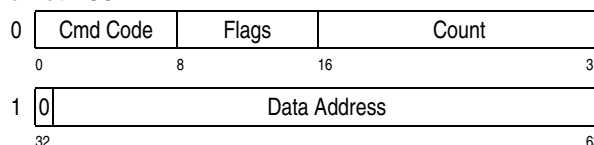
CCWs have either of two different formats, format 0 or format 1. The two formats do not differ in the information contained in the CCW, but they do differ in the size of the address and the arrangement of fields within the CCW.

The formats are defined as follows:

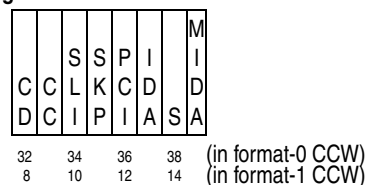
Format-0 CCW



Format-1 CCW



Flags



Format-0 CCWs can be located anywhere in the first 2^{24} (16M) bytes of absolute storage, and format-1 CCWs can be located anywhere in the first 2^{31} (2G) bytes of absolute storage.

If indirect data addressing is specified and the format-2-IDAW-control bit is zero in the ORB associated with the CCW, then:

1. Bits 30 and 31 (format 0) or 62 and 63 (format 1) of the CCW must be zeros, designating a word boundary,
2. Bit 0 of the first entry of the indirect-data-address list must be zero.
3. The MIDA flag must be zero.

If indirect data addressing is specified and the format-2-IDAW-control bit is one in the ORB associated with the CCW, bits 29-31 (format 0) or bits 61-63 (format 1) of the CCW must be zeros, designating a doubleword boundary.

If modified indirect data addressing is specified, then:

1. The modified-CCW-indirect-data-addressing-control bit in the ORB must be one.

2. Bits 28-31 (format 0) or bits 60-63 (format 1) of the CCW must be zeros, designating a quadword boundary.
3. The CCW SKP and IDA flags must be zero.
4. The MIDAW count field must be greater than zero.
5. The combination of the MIDAW count and data-address fields must not specify the transfer of data across a 4K-byte boundary, unless skipping is in effect for the MIDAW.
6. Reserved bits in the first entry of the modified-CCW-indirect-data-address list must be zero.

When any of these requirements is not met, a program-check condition may be recognized (see “CCW Indirect Data Addressing” on page 15-61 and “Modified CCW Indirect Data Addressing” on page 15-63). Detection of this condition during data chaining causes the I/O device to be signaled to conclude the operation. When a failure to meet these requirements is detected during command chaining or subsequent to the execution of START SUBCHANNEL, the new operation is not initiated, and an interruption condition is generated.

The contents of bit positions 40-47 of a format-0 CCW are ignored.

The fields in the CCWs are defined as follows:

Command Code: Bits 0-7 (both formats) specify the operation to be performed.

Data Address: Bits 8-31 (format 0) or bits 33-63 (format 1) designate a location in absolute storage. The designated location is the first location referred to in the area designated by the CCW. Bit 32 of a format-1 CCW must be zero; otherwise, a program-check condition is recognized. If a byte count of zero is specified, this field is not checked.

When the MIDA flag is one, the data-address field must designate an absolute storage location that is on a quadword boundary; otherwise, a program-check condition is recognized.

See “CCW Indirect Data Addressing” on page 15-61 and “Modified CCW Indirect Data Addressing” on page 15-63 for information about the specification of data addresses greater than 2G bytes.

Chain-Data (CD) Flag: Bit 32 (format 0) or bit 8 (format 1), when one, specifies chaining of data. The bit causes the storage area designated by the next CCW to be used with the current I/O operation. When the CD flag is one in a CCW, the chain-command and suppress-length-indication flags (see below) are ignored.

Chain-Command (CC) Flag: Bit 33 (format 0) or bit 9 (format 1), when one, and when the CD flag and S flag are both zeros, specifies chaining of commands. The bit causes the operation specified by the command code in the next CCW to be initiated upon the normal completion of the current operation.

Suppress-Length-Indication (SLI) Flag: Bit 34 (format 0) or bit 10 (format 1) controls whether an incorrect-length condition is to be indicated to the program. When this bit is one and the CD flag is zero, the incorrect-length indication is suppressed. When both the CC and SLI flags are ones and the CD flag is zero, command chaining takes place, regardless of the presence of an incorrect-length condition. This bit should be specified in *all* CCWs where suppression of the incorrect-length indication is desired.

Skip (SKP) Flag: Bit 35 (format 0) or bit 11 (format 1), when one, specifies the suppression of transfer of information to storage during a read, read-backward, sense ID, or sense operation. This bit must be zero when the MIDA flag is one; otherwise, a program-check condition is recognized.

Program-Controlled-Interruption (PCI) Flag: Bit 36 (format 0) or bit 12 (format 1), when one, causes the channel subsystem to generate an intermediate interruption condition when the CCW containing the bit takes control of the I/O operation. When the PCI flag bit is zero, normal operation takes place.

Indirect-Data-Address (IDA) Flag: Bit 37 (format 0) or bit 13 (format 1), when one, specifies indirect data addressing. This bit must be zero when the MIDA flag is one; otherwise, a program-check condition is recognized.

Suspend (S) Flag: Bit 38 (format 0) or bit 14 (format 1), when one, specifies suspension of channel-program execution. When valid, it causes channel-program execution to be suspended prior to the execution of the CCW containing the S flag. A one value of the S flag is valid when bit 4 of word 1 of the associated ORB is one.

Modified-Indirect-Data-Address (MIDA) Flag: Bit 39 (format 0) or bit 15 (format 1), when one, specifies modified indirect data addressing. This bit must be zero when the modified-CCW-indirect-data-addressing-control bit in the ORB is zero, the CCW IDA flag is one, the CCW SKP flag is one, or the channel paths associated with the subchannel do not support modified CCW indirect data addressing; otherwise, a program-check condition is recognized.

Count: Bits 48-63 (format 0) or bits 16-31 (format 1) specify the number of bytes in the storage area designated by the CCW.

Transport Control Word

The transport-control word (TCW) specifies a transport-command-control block (TCCB) whose contents are to be transported to the I/O device for processing. The TCCB contains one or more device-command words (DCWs) and associated options. For a DCW that specifies a command which initiates the transfer of data (with the exception of control data contained within the TCCB), the TCW designates one or more storage areas where the data is located.

When START SUBCHANNEL is issued specifying a transport-mode ORB, the TCW designated by the ORB becomes the current TCW, the subchannel becomes start pending, and the TCCB designated by the TCW is transported to the I/O device for execution. The TCW remains the current TCW until TEST SUBCHANNEL clears the start-function indication in the subchannel.

The TCW is a 64-byte control block that is designated on a 64-byte boundary. For all operations, except for the interrogate operation, accesses to the TCW by the channel subsystem are made using the subchannel storage-access key specified in the ORB. For interrogate operations, fetch accesses to the interrogate TCW by the channel subsystem are made as if a subchannel storage-access key of zero is used. The channel subsystem does not make any store accesses to a TCW

The format of the TCW is as follows:

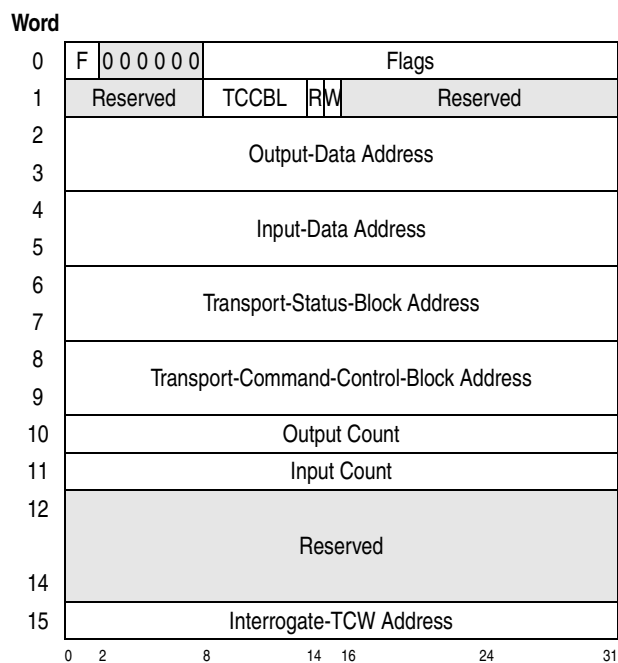


Figure 15-7. Transport-Control Word (TCW)

Format (F): Bits 0-1 of word 0 form the TCW-format that contains a 2-bit unsigned integer value that defines the layout of the TCW. The value of this field must be zero, otherwise a program-check condition is recognized.

Reserved: Bits 2-7 of word 0 are reserved and must be zeros, otherwise a program-check condition is recognized.

Bytes 0 and 2-3 of word 1 and words 12-14 are reserved and not checked. These fields are reserved for future extensions. Programs which place non-zero values in these fields may not operate compatibly on future models.

Flags: Bytes 1-3 of word 0 contain information about the TCW. The meaning of each flag bit is as follows:

Bit	Meaning
0-4	Reserved. Bits 0-4 are reserved and must be zero, otherwise a program-check condition is recognized.
5	Input transport-indirect-data addressing (TIDA): When bit 5 is zero and bit 14 of word 1 (the R-bit) is one, the input-data-

address field designates the absolute address of the input location. When bit 5 is one and the R-bit is one, the input-data-address field designates the absolute address of a transport-indirect-data-address word (TIDAW) or the first TIDAW in a list of TIDAWS that designate the input storage location or locations, respectively.

When the R-bit is zero, bit 5 has no meaning.

- 6 Transport-command-control-block TIDA: When bit 6 is zero, the transport-command-control-block-address field designates the absolute address of the transport-command-control block for the TCW. When bit 6 is one, the transport-command-control-block-address field designates the absolute address of a TIDAW or list of TIDAWs that designate the location or locations, respectively, of the transport-command-control block for the TCW.
- 7 Output TIDA: When bit 7 is zero and bit 15 of word 1 (the W-bit) is one, the output-data-address field designates an output location in absolute storage. When bit 7 is one and the W-bit is one, the output-data-address field designates the absolute address of a TIDAW or list of TIDAWs that designate the output storage location or locations, respectively.

When the W-bit is zero, bit 7 has no meaning.
- 8-9 TIDAW format: Bits 8-9 form a 2-bit encoded field that specifies the format of TIDAWs designated by the TCW. This field must contain zeros, otherwise a program-check condition is recognized.
- 10-23 Reserved. Bits 10-23 are reserved and must be zero, otherwise a program-check condition is recognized.

Transport-Command-Control-Block Length (TCCBL): Bits 8-13 of word 1, with two zeros appended on the right, specify the unsigned integer whose value, when added to 20, specifies the length of the TCCB in bytes. This field must, before having 20 added to it, specify a length that is between 12

and 244 inclusive, otherwise a program-check condition is recognized.

Note: The TCCBL is based on the FCP additional command length and does not include the TCA header (16 bytes), and the transport-count (4 bytes) in the TCA trailer.

Read Operations (R): When bit 14 of word 1 is one, the input count (word 11) is valid and contains a non-zero value, indicating the number of bytes to be transferred into main storage.

Write Operations (W): When bit 15 of word 1 is one, the output count (word 10) is valid and contains a non-zero value, indicating the number of bytes to be transferred from main storage.

If the R-bit and W-bit are both one, a program-check condition is recognized.

If the W-bit is one and the TCW is an interrogate TCW, a program-check condition is recognized.

Output-Data Address: When bit 15 of word 1 (the W-bit) is one and bit 7 of the flags field (the output-TIDA flag) is zero, words 2-3 designate the 64-bit output location in absolute storage. When the W-bit is one and the output-TIDA flag is one, words 2-3 designate the 64-bit location in absolute storage of a TIDAW or list of TIDAWs that designate the output storage location or locations.

When the W-bit is one and the output-TIDA flag is one, words 2-3 must designate a storage location that is on a quadword boundary, otherwise a program-check condition is recognized.

When the W-bit is zero, words 2-3 have no meaning and are ignored.

Input-Data Address: When the bit 14 of word 1 (the R-bit) is one and bit 5 of the flags field (the input-TIDA flag) is zero, words 4-5 designate the 64-bit input location in absolute storage. When the R-bit is one and the input-TIDA flag is one, words 4-5 designate the location in absolute storage of a TIDAW or list of TIDAWs that designate the input storage location(s).

When the R-bit is one and the input-TIDA flag is one words 4-5 must designate a storage location that is on a quadword boundary, otherwise a program-check condition is recognized.

When the R-bit is zero, words 4-5 have no meaning and are ignored.

Transport-Status-Block Address: Words 6-7 designate the 64-bit location in absolute storage of the transport-status block for the TCW.

Words 6-7 must designate a storage location on an doubleword boundary, otherwise a program-check condition is recognized.

Transport-Command-Control-Block Address: If the transport-command-control-block-TIDA bit (bit 6 of the flags field) is zero, words 8-9 designate 64-bit location in absolute storage of the transport-command-control block. When the transport-command-control-block-TIDA bit is zero, the transport-command-control block is specified to reside in a contiguous area of storage. If the transport-command-control-block-TIDA bit is one, words 8-9 designate the 64-bit location in absolute storage of a TIDAW or list of TIDAWs that designate the location in absolute storage of the transport-command-control block. When the transport-command-control-block-TIDA bit is one, the transport-command-control block may be specified to reside in noncontiguous areas of storage.

When the transport-command-control-block-TIDA flag is zero, words 8-9 should designate a storage location on a doubleword boundary, otherwise, degradation of performance is possible.

When the transport-command-control-block-TIDA flag is one:

- Words 8-9 must designate a storage location on a quadword boundary, otherwise a program-check condition is recognized.
- The transport-command-control-block-address and transport-command-control-block-length fields must not specify a list of TIDAWs that cross a 4K-byte boundary, otherwise a program-check condition is recognized.
- The TIDAW-transfer-in-channel (TTIC) bit should be zero in all specified TIDAWs, otherwise degradation of performance is possible.
- The first TIDAW should designate the TCCB on a doubleword boundary; otherwise, degradation of performance is possible.

Output Count: When bit 15 of word 1 (the W-bit) is one, word 10 contains the unsigned integer total count of output bytes for the TCW. The output-count value is determined as follows:

- When flags bit 7 (the output-TIDA bit) is zero, then word 10 contains the total count of data bytes for the TCW.
- When flags bit 7 is one, word 10 contains the sum of the following:
 - The total count of data bytes for the TCW. The total count of data bytes for the TCW is equal to the sum of count fields in all of the DCWs in the TCCB designated by the TCW.
 - Four bytes for the CBC information for every TIDAW designated by the output-address field for which the insert-CBC flag bit is one.
 - The size of the reserved area in the CBC-offset block (COB), if a COB is specified in the output data.
 - The count of padding bytes added for every TIDAW designated by the output-address field for which the insert-CBC flag bit is one. (See “Transport Indirect Data Addressing” on page 15-65.)

When the W-bit is one and word 10 contains zeros, a program-check condition is recognized.

When the W-bit is one:

- The output-count value, rounded upwards to the nearest multiple of 4, plus 4, should equal the value in the transport-count field in the associated TCCB, otherwise this may lead to a device-detected-program-check condition being recognized.
- The output-count value should equal the sum of count fields in all of the DCWs in the associated TCCB, plus the count of any inserted padding and CBC bytes, otherwise this may lead to a device-detected-program-check condition being recognized.

When the W-bit is zero, word 10 is ignored.

Input Count: When bit 14 of word 1 (the R-bit) is one, word 11 contains the unsigned integer total count of input bytes for the TCW.

When the R-bit is one and word 11 contains zeros, a program-check condition is recognized.

When the R-bit is one:

- The input-count value, rounded upwards to the nearest multiple of 4, plus 4, should equal the value in the transport-count field in the associated TCCB, otherwise this may lead to a device-detected-program-check condition being recognized.
- The input-count value should equal the sum of count fields in all of the DCWs in the associated TCCB, otherwise this may lead to a device-detected-program-check condition being recognized.

When the R-bit is zero, word 11 is ignored.

Interrogate-TCW Address: When a START SUBCHANNEL designates a TCW, word 15 of the TCW is not checked. When a CANCEL SUBCHANNEL designates a subchannel that is start pending for a TCW channel program and is not status pending, bits 1-31 of word 15 of the TCW designated by START SUBCHANNEL specify the 31-bit location in absolute storage of the interrogate-TCW that is used to initiate an interrogate operation for the subchannel. If word 15 contains zeros when CANCEL SUBCHANNEL is issued, an interrogate operation is not initiated.

When CANCEL SUBCHANNEL is issued, bit 0 of word 15 must be zero, otherwise a program-check condition is recognized with interrogate-failed status indicated. When CANCEL SUBCHANNEL is issued and bits 1-31 of word 15 do not contain zeros, bits 1-31 of word 15 must designate a storage location on a 64-byte boundary, otherwise a program-check condition is recognized with interrogate-failed status indicated.

Transport-Command-Control Block

The transport-command-control block (TCCB) is variable in length, contains header and trailer information, and from 1 to 30 device-command words (DCWs) that are logically linked such that they are executed by the control unit in a sequential manner.

A TCCB may reside as a single block of contiguous storage or it may reside as multiple blocks of noncontiguous storage. The TCCB-TIDA flag specifies

whether the TCCB resides in contiguous storage. When the flag is zero, the TCW designates a contiguous TCCB. When the flag is one, the TCW designates one or more TIDAWs that designate the location of the TCCB in noncontiguous storage.

The transport-command-control block is designated on an doubleword boundary and has a maximum length of 264 bytes. For non-interrogate operations, fetch accesses to the TCCB by the channel subsystem are made using the subchannel storage-access key specified in the ORB. For interrogate operations, fetch accesses to the interrogate TCCB by the channel subsystem are made as if a storage-access key of zero is used. The channel subsystem does not make any store accesses to a TCCB nor does it examine its contents.

The format of the TCCB is as follows:

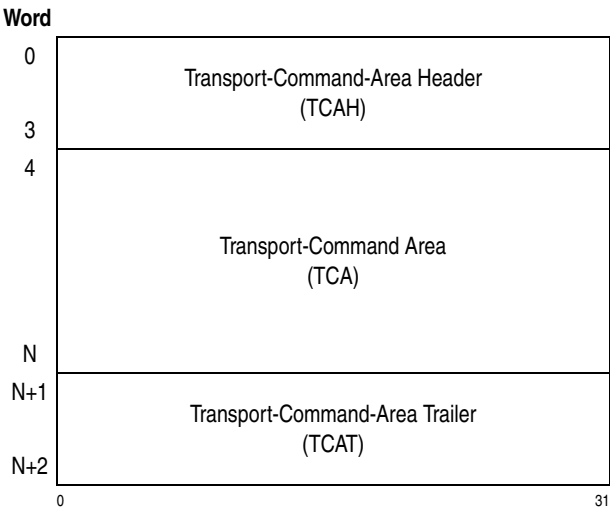


Figure 15-8. Transport-Command-Control Block (TCCB)

Transport-Command-Area Header

The transport-command-area header (TCAH) is a 16-byte control block that contains information about the transport-command area and the operations described therein. The TCAH has the following format:

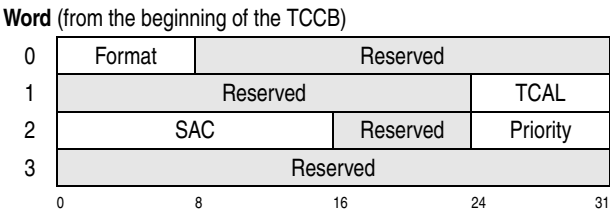


Figure 15-9. Transport-Command-Area Header (TCAH)

Format Control: Byte 0 of word 0 is the TCCB format control. The value of this field must be 7F hex, otherwise a device-detected-program-check is recognized.

Reserved: Byte 1 of word 0 is reserved and must contain zeros, otherwise a device-detected-program-check condition may be recognized. Bytes 2-3 of word 0, bytes 0-2 of word 1, byte 2 of word 2, and bytes 0-3 of word 3 are reserved and should contain zeros, otherwise the channel program may not operate compatibly in the future.

Transport-Command-Area Length (TCAL): Byte 3 of word 1 specifies an 8-bit unsigned integer whose value, when reduced by 12, specifies the length of the TCA in bytes. The specified value must be a non-zero multiple of 4 that is between 20 and 252, inclusive, otherwise a device-detected-program-check condition may be recognized.

Service-Action Code (SAC): Bytes 0-1 of word 2 contain an unsigned integer value that must be either 1FFE hex or 1FFF hex. The meanings of these values is device dependent.

Priority: Byte 3 of word 2 contains the 8-bit unsigned binary integer priority value for the TCCB. A value of zero indicates that no priority has been assigned. Valid priority values range from 1, the lowest priority, to 255, the highest priority.

Transport-Command Area

The transport-command area (TCA) is a variable-length area that contains from 1 to 30 device-command words (DCWs). The length of the TCA is an integral number of words.

For DCWs that specify control data, the TCA also contains the control data associated with the commands. Each DCW that specifies control data, reduces the maximum-DCW capacity by one or more DCWs, depending on the size of the command-associated data.

For DCWs that specify input or output data, the TCW specifying the TCCB designates the associated storage area or areas (see “Input-Data Address” on page 15-34 and “Output-Data Address” on page 15-34) and the DCW designates the count of bytes to transfer.

The maximum size of the TCA is 240 bytes. The TCA has the following format:

Word (from the beginning of the TCA)

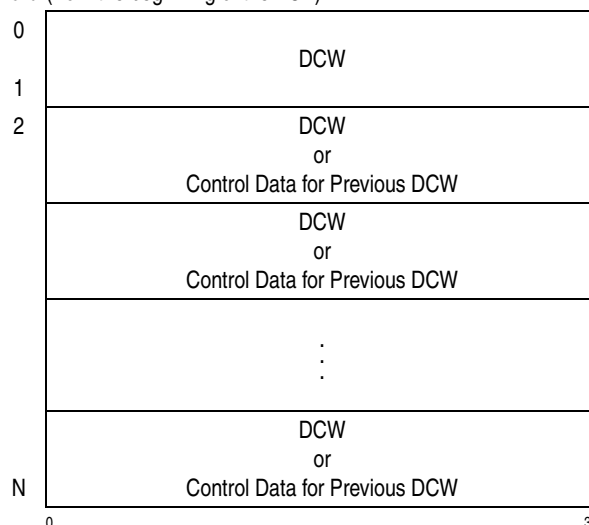


Figure 15-10. Transport-Command Area (TCA)

Device-Command Word

A device-command word (DCW) specifies a command to be executed. For commands initiating certain I/O operations, it designates the count of bytes on which the operation is performed, the action to be taken whenever transfer to or from storage is completed, and other options. The storage area or areas associated with a DCW data-transfer operation are designated, depending on the operation specified by the command, by the input-data-address field or the output-data-address field of the TCW that designates the transport-control-block containing the DCW. Whether the input-data-address field or the output-data-address field of the TCW designates the storage directly or indirectly, by use of a TIDAW list, is specified by the input-TIDA and output-TIDA flags in the TCW, respectively.

For commands initiating control operations, the associated data immediately follows the command. If this data is not a multiple of 4 bytes, the field immediately following the data begins on the next word boundary that immediately follows the last byte of specified data. This field will be either the next DCW, if additional DCWs are specified, or the transport-command-area trailer, if additional DCWs are not specified.

As each DCW is executed and while it is executing it is recognized as the current DCW. A DCW becomes

TCAT is determined as follows, otherwise a device-detected-program-check condition may be recognized:

- The count field values in the DCWs that each specify a read-type command are summed.
- The sum is rounded upwards to the nearest multiple of 4.
- The rounded sum is increased by 4 giving the transport-count value.

When a read operation is specified, the transport-count value should be equal to the value in the TCW input-count field, rounded upwards to the next multiple of 4, plus 4, otherwise a device-detected-program-check condition may be recognized.

When a write operation is specified (the W-bit in the TCW is one), the value in the transport-count field in the TCAT is determined as follows, otherwise a device-detected-program-check condition may be recognized:

- The count field values in the DCWs that each specify a write-type command are summed.
- If a CBC-offset block (COB) is specified in the output data, the count field value in the transfer-

CBC-offset-block (TCOB) DCW and the size of the reserved field in the COB are added to the sum.

- The total of the counts of any TIDAW-specified CBC bytes and padding bytes (see “Transport Indirect Data Addressing” on page 15-65) is added to the sum.
- The sum is rounded upwards to the nearest multiple of 4.
- The rounded sum is increased by 4 giving the transport-count value.

When a write operation is specified, the transport-count value should be equal to the value in TCW output-count field, rounded upwards to the next multiple of 4, plus 4, otherwise a device-detected-program-check condition may be recognized.

When neither a read nor a write operation is specified (both the W-bit and R-bit in the TCW are zero), the transport count value should be zero, otherwise this may lead to a device-detected-program-check condition being recognized.

Programming Note: The following table summarizes the determination of the TCW input-count, output-count, and the TCCB transport count values:

Operation	TCW Input Count	TCW Output Count	TCAT Transport Count
Input operation (TCW R-bit is 1)	Sum of DCW count values in the TCA.	n/a	Sum of DCW count values in TCA, rounded to four, ² plus four ²
Output operation (TCW W-bit is 1), TIDAWS not used (TCW output-TIDA flag is zero)	n/a	Sum of DCW count values in TCA.	Sum of DCW count values in TCA, rounded to four, ² plus four ²
Output operation (TCW W-bit is 1), TIDAWS used (TCW output-TIDA flag is one)	n/a	Sum of DCW count values in TCA plus the size of the reserved area in the COB when the COB is specified in output data, plus total of TIDAW-specified CBC and padding bytes ¹	Sum of DCW count values in TCA, plus the size of the reserved area in the COB when the COB is specified in output data, plus total of TIDAW-specified CBC and padding bytes, ¹ rounded to four, ² plus four ²
Explanation: ¹ See the description of the insert-CBC control in “Flags” on page 15-65. ² The channel subsystem adds a CBC word to the end of the last data transported. Up to 3 padding bytes are added after the last data transported to ensure the added CBC word is on a word boundary.			

Figure 15-13. Determination of the TCAT Transport Count, TCW Input Count, and TCW Output Count

Transfer-CBC-Offset Block DCW

For write operations, the transfer-CBC-offset-block (TCOB) DCW specifies that a CBC-offset block (COB) is transported to the device. The content of the TCOB DCW is as follows:

- The command code contains the transfer-CBC-offset-block command which is a value of 60 hex.
- The chain-command flag is one.
- When the CD-count is not zero, the COB immediately follows the TCOB DCW in the TCA and the CD-count specifies the number of CBC offsets in the COB multiplied by 4.
- When the CD-count is zero, the COB is in the output data, the location of the COB is specified by the output-data-address field in the TCW, and the count field specifies the number of CBC offsets in the COB multiplied by 4.

When a TCOB DCW is used, a device-detected program check is recognized when any of the following conditions exist:

- A TCOB DCW is specified and a write operation is not specified (that is, the W-bit in the TCA is zero).
- A TCOB DCW is the only DCW in the TCA. (That is the chain-command flag in the TCOB DCW is zero.)
- A TCOB DCW is not the first DCW in the TCA.
- More than one TCOB DCW is specified
- The control-data-count field and the count field in the TCOB DCW both contain the value zero or both contain a value that is non-zero.
- The control-data-count field contains the value zero, the count field contains a value that is non-zero, but the value in the count field is not a multiple of 4.
- The count field contains the value zero, the control-data-count field contains a value that is non-zero, but the value in the control-data-count field is not a multiple of 4.

When a TCOB is used that specifies the COB in the output data, the following must be true; otherwise, a device-detected program check may be recognized:

- The output-TIDA flag (flags bit 7) in the TCW must be one.
- The insert-CBC control must be set to one in the last or only TIDAW that is used to transfer the COB.
- When there is an odd number of CBC offsets in the COB, the total of the count fields in the TIDAWs used to the transfer the COB must specify the number of CBC offsets in the COB multiplied by 4. When there is an even number of CBC offsets in the COB, the total of the count fields in the TIDAWs used to the transfer the COB must specify the number of CBC offsets in the COB multiplied by 4, plus 4.

Whether a device recognizes the TCOB command is determined by device-dependent means.

CBC-Offset Block

The CBC-offset block (COB) is a variable-length control block that contains a list of 4-byte entries, each of which identifies the offset of a CBC specified by a TIDAW to be inserted in the output data. When the COB is specified in the output data, the offset of the CBC inserted by the TIDAW that specifies the transfer of the COB is not included in the COB.

Accesses to a CBC-offset block by the channel subsystem are treated as if a storage-access key of zero is used.

The COB has the following format:

4-Byte Entry

0	CBC Offset 0
1	CBC Offset 1
2	.
	.
	.
N	CBC Offset N
Y	Reserved (When the COB is in output data.)
0	31

Explanation:

- The size of the COB is $4(N+1)$ when the COB is specified in the TCA. (There is no reserved field for this case.)
- The size of the COB is $4(Y+1)$ when the COB is specified in the output data, where Y is an even number that is equal to either N or $N+1$.

Figure 15-14. CBC-Offset Block (COB)

CBC Offsets: Entries 0 through *N* contain CBC offsets. Each CBC offset is a 4-byte field containing a 32-bit unsigned integer value that specifies the relative offset, from the beginning of the output-data area, to a CBC that is specified to be inserted by a TIDAW. When a COB is in the TCA, the beginning of the output-data area is specified by the output-data-address field of the TCW. When a COB is in the output-data, the output-data area begins immediately after the COB.

Reserved: When a COB is specified in the TCA, or a COB is specified in the output data and there is an odd number of CBC offsets in the COB, there are no reserved bytes.

When a COB is specified in the output data and there is an even number of CBC offsets in the COB, the 4 bytes immediately following the last CBC offset are reserved and should contain zeros, otherwise unpredictable results may result.

Programming Note: When a device indicates that it supports the TCOB command and TIDAWs are used that specify insert-CBC, a TCOB and COB should be used; otherwise, unpredictable results or errors may result.

Interrogate TCCB

The interrogate TCCB is a TCCB that specifies a service-action code value of 1FFF hex, contains a TCA that is at least 8 bytes in length that specifies an interrogate DCW, and may specify a read operation.

Any data that immediately follows the interrogate DCW and is specified by a non-zero CD-count in the interrogate DCW is transported to the device and may be written to a device-dependent log. Any input data that is specified by the interrogate TCW is device-dependent data and is transferred from the I/O device.

Accesses to the interrogate TCCB by the channel subsystem are treated as if a storage-access key of zero is used.

The format of the interrogate TCA is as follows:

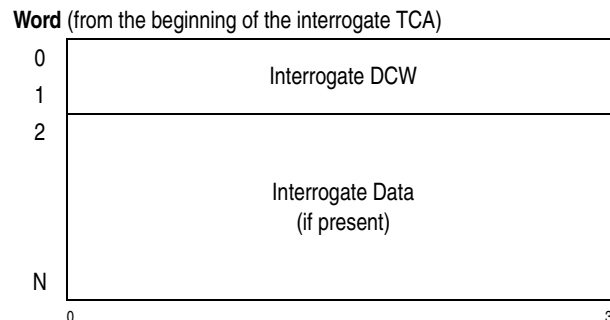


Figure 15-15. Interrogate Transport-Command Area (TCA)

Interrogate DCW

Words 0 and 1 of the interrogate TCA contain a DCW that specifies an interrogate operation. The content of the interrogate DCW is as follows:

- The command code contains the interrogate command which is a value of 40 hex.
- The CD-count field must contain a value that is not greater than 232, otherwise a device-detected-program-check condition is recognized with interrogate-failed status indicated in the subchannel-extended-status field of the IRB.
- With the exceptions of the command code, count and CD-count fields, all other fields in the DCW must contain zeros, otherwise a device-detected-program-check condition is recognized with interrogate-failed status indicated in the subchannel-extended-status field of the IRB.

Interrogate Data

If the CD-count of the interrogate DCW is greater than zero, interrogate data is specified. Since interrogate data is for device-dependent logging purposes only, interrogate data that is incorrectly specified

does not cause any exception condition to be recognized. The interrogate data has the following format:

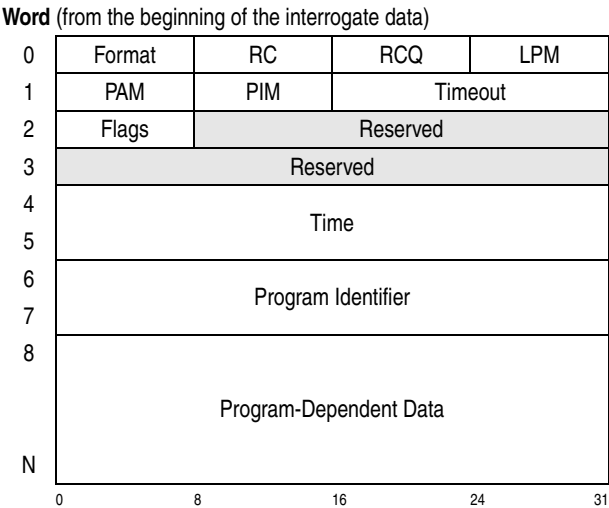


Figure 15-16. Interrogate Data

Format: Byte 0 of word 0 contains the unsigned integer value that defines the layout of the interrogate data. The value of this field should be zero, otherwise the I/O device may not recognize the layout of the interrogate data

Reason Code (RC): Byte 1 of word 0 contains the unsigned integer value that indicates the reason the interrogate operation was initiated. The following values may be specified. The meaning of RC values is as follows:

Value	Meaning
0	Interrogate reason not specified.
1	Timeout: Program-detected timeout for the operation being interrogated.
2-255	Reserved.

Reason-Code Qualifier (RCQ): Byte 2 of word 0 contains the unsigned integer value that indicates additional information about the reason the interrogate operation was initiated.

When the reason-code field contains the value one, the meaning of RCQ values are as follows:

Value	Meaning
0	Interrogate reason qualifier not specified.

- 1 Primary: The timeout was specified by the primary program.
- 2 Secondary: The timeout was specified by a secondary program.
- 3-255 Reserved.

When the reason-code field does not contain the value one, the reason-code-qualifier field has no meaning.

Logical-Path Mask (LPM): Byte 3 of word 0 contains the LPM that was used when the operation being interrogated was initiated by START SUB-CHANNEL.

Path-Available Mask (PAM): Byte 0 of word 1 contains the value of the PAM at the time the interrogate operation is initiated.

Path-Installed Mask (PIM): Byte 1 of word 1 contains the value of the PAM at the time the interrogate operation is initiated.

Timeout: When the RC field contains the value one or two, bytes 2-3 of word 1 contain the timeout interval used by the designated program, in unsigned integer seconds.

Flags: Byte 0 of word 2 contains information about the interrogate. The meaning of each flag bit is as follows:

Bit	Meaning
0	Multipath mode: This bit has the same setting as the multipath-mode bit (D) at the time the interrogate operation is initiated.
1	Program path recovery: The interrogate is issued during path recovery by the program.
2	Critical: The device is a critical device for the program.
3-7	Reserved.

Reserved: Bytes 1-3 of word 2 and word 3 are reserved.

Time: Words 4-5 contain the time the interrogate operation was initiated.

Program Identifier: Words 6-7 identify the program initiating the interrogate operation. The content of this field is program dependent.

Program-Dependent Data: Words 8-N contain program-dependent information.

Programming Notes:

1. It is assumed that the program will specify the value in the time field as UTC in the same form as the TOD clock.
2. If the interrogate operation does not complete within a model-dependent amount of time, an interface-control check condition is recognized, terminating the I/O operation being interrogated.
3. If a timeout value is specified and the timeout value is less than a device-dependent minimum, the I/O device may not write the interrogate data into its log.

Transport Status Block

When the TSB-valid bit in the IRB is one, the transport-status block (TSB) for the I/O operation may contain additional information about the completion of the associated TCW channel program. When the the interrogate-complete (Q) bit in the IRB is one, the interrogate TSB may contain interrogate data.

The TSB is 64 bytes in length and is allocated on an doubleword boundary. Accesses to the TSB by the channel subsystem are treated as if a storage-access key of zero is used.

The format of the TSB is as follows:

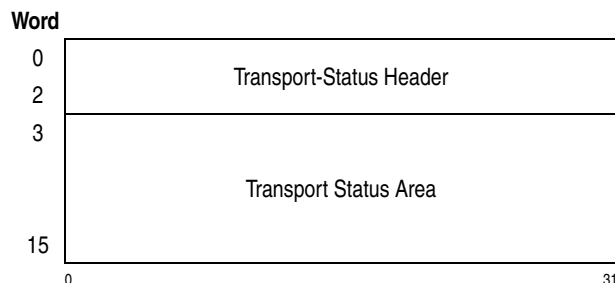


Figure 15-17. Transport-Status Block

Transport-Status Header (TSH)

The transport-status header is a 12-byte control block that contains information about the transport status. The format of the TSH is as follows:

Word (from the beginning of the TSB)

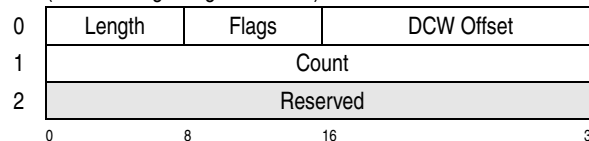


Figure 15-18. Transport-Status Header (TSH)

Length: Byte 0 of word 0 contains the unsigned binary integer length, in bytes, of the status information stored in the TSB. This length includes the TSH and the transport-status area (TSA), if present. If this value is less than 64, the difference between the length value and 64 specifies the number of bytes, at the end of the TSB, that are not used, have no meaning, and may or may not be stored.

Flags: Byte 1 of word 0 contains information about the TSB. The meaning of each flag bit is as follows.:

Bit	Meaning
0	DCW-offset field valid. When bit 0 is one, the DCW-offset field contains a DCW-off-set value. When bit 0 is zero, the DCW-off-set field has no meaning.
1	Count field valid. When bit 1 is one, the count field contains a count value. When bit 1 is zero, the count field has no meaning.
2	Cache miss. When flag bits 4-7 specify an I/O-status TSB and bit 2 is one, one or more I/O-device cache misses occurred during the I/O operation. When flag bits 4-7 specify an I/O-status TSB and bit 2 is zero, no cache misses occurred. When flag bits 4-7 do not specify an I/O-status TSB, bit 2 is reserved.
3	Time fields valid. When flag bits 4-7 specify an I/O-status TSB and bit 3 is one, the device-time, defer-time, queue-time, device-busy-time, and device-active-only time fields in the I/O-status TSA contain time information. When flag bits 4-7 specify an I/O-status TSB and bit

3 is zero, the contents of the device-time, defer-time, queue-time, device-busy-time, and device-active-only time fields in the I/O-status TSA have no meaning.

When flag bits 4-7 do not specify an I/O-status TSB, bit 3 is reserved.

- 4 Reserved.
- 5-7 TSA format. Bits 5-7 form a 3-bit unsigned integer that indicates the layout of the TSA as follows:

Value	Meaning
0	TSA contents have no meaning: The other fields of the TSA do not contain meaningful data.
1	I/O Status TSA: The TSA contains ending status in addition to ending status in the IRB for the I/O operation.
2	Device-detected-program-check TSA: The TSA contains information describing the reason for the device-detected program check
3	Interrogate TSA: The TSA contains interrogate response information
4-7	Reserved.

DCW Offset: When bit 0 of the flags field (the DCW-offset-field-valid bit) is one, bytes 2-3 of word 0 contain the byte offset, from the beginning of the TCA to the DCW either partially or completely executed when the TSB is stored. If the channel program cannot be completed, this offset identifies the DCW for which processing could not be completed. Additional information in the IRB and TSB may be used for recovery purposes. When bit 0 of the flags field is zero, bytes 2-3 of word 0 have no meaning.

Count: When bit 1 of the flags field (the count-field-valid bit) is one, word 1 contains the residual count for the DCW designated by the DCW-offset field.

Reserved: Word 2 is reserved and contains zeros.

Programming Note: If the program sets the length and flags fields to zeros before initiating an operation, the program can determine whether the TSB con-

tains meaningful information after the operation completes by checking that both the length and flags field contain non-zero values.

I/O-Status TSA

When the TSB-format field in the TSH contains one, the TSA has the following format:

Word (from the beginning of the TSA)

0	Device Time
1	Defer Time
2	Queue Time
3	Device-Busy Time
4	Device-Active-Only Time
5	Additional Data (if present)
6	
7	
8	
9	
10	
11	
12	
0	0
	31

Figure 15-19. I/O-Status Transport-Status Area (TSA)

Device Time: When bit 3 of the flags field in the TSH (the time-fields-valid flag) is one, word 0 contains the 32-bit unsigned integer total count of time intervals between when the I/O device received the information in the TCCB until the I/O device returned the information in the TSB. The resolution of the device time is such that bit 31 corresponds to one microsecond.

When bit 3 of the flags field in the TSH (the time-fields-valid flag) is zero, the contents of word 0 have no meaning.

Defer Time: When bit 3 of the flags field in the TSH (the time-fields-valid flag) is one, word 1 contains the 32-bit unsigned integer total count of time intervals that the I/O operation was temporarily delayed by the I/O device to perform device-dependent operations or other operations not associated with the current channel program. The resolution of the defer time is such that bit 31 corresponds to one microsecond.

When bit 3 of the flags field in the TSH (the time-fields-valid flag) is zero, the contents of word 1 have no meaning.

Queue Time: When bit 3 of the flags field in the TSH (the time-fields-valid flag) is one, word 2 contains the 32-bit unsigned integer total count of timer intervals that the I/O operation or portions of the

operation were queued at the control unit while waiting for operations not associated with the current operation to complete. The resolution of the queue time is such that bit 31 corresponds to one microsecond.

When bit 3 of the flags field in the TSH (the time-fields-valid flag) is zero, the contents of word 2 have no meaning.

Device-Busy Time: When bit 3 of the flags field in the TSH (the time-fields-valid flag) is one, word 3 contains the 32-bit unsigned integer total count of time intervals that the I/O device was busy attempting to initiate a command. The resolution of the device-busy time is such that bit 31 corresponds to one microsecond.

When bit 3 of the flags field in the TSH (the time-fields-valid flag) is zero, the contents of word 3 have no meaning.

Device-Active-Only Time: When bit 3 of the flags field in the TSH (the time-fields-valid flag) is one, word 4 contains the 32-bit unsigned integer total count of time between the time that the channel-end condition existed and the time that the channel-end and device-end conditions were recognized by the device. The resolution of the device-active time is such that bit 31 corresponds to one microsecond.

When bit 3 of the flags field in the TSH (the time-fields-valid flag) is zero, the contents of word 4 have no meaning.

Additional Data: When the length field in the TSH contains a value greater than 32, additional data has been provided by the I/O device. The count of meaningful additional-data bytes in this field is determined by subtracting 32 from the value in the length field.

When unit check is included in the device status, any additional data provided is sense data. When unit check is not included in the device status, any additional data provided is device dependent.

Device-Detected-Program-Check TSA

When the TSB-format field in the TSH contains two, the TSA has the following format:

Word (from the beginning of the TSA)

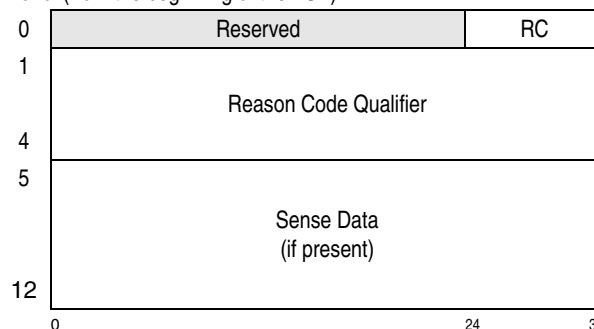


Figure 15-20. Device-Detected-Program-Check Transport-Status Area (TSA)

Reserved: Bytes 0-2 of word 0 are reserved and have no meaning.

Reason Code (RC): Byte 3 of word 0 contains an 8-bit unsigned integer code that indicates the primary reason for the device-detected program check. The meaning of each reason code value is as follows:

Value Meaning

- | | |
|---|--|
| 0 | No information: Byte 3 has no meaning. |
| 1 | TCCB transport failure: An invalid CBC was detected by the I/O device while transporting the TCCB. The reason-code-qualifier (RCQ) field contains additional information. |
| 2 | Invalid CBC detected on output data: An invalid CBC was detected while transferring output data. The RCQ field contains additional information. |
| 3 | Incorrect TCCB length specification: The RCQ field contains additional information. |
| 4 | TCAH specification error: The RCQ field contains additional information.

When a TCAH specification error is recognized, pre-existing allegiance conditions are not cleared. |
| 5 | DCW specification error: There is an error with the DCW designated by the |

- DCW-offset field in the TSH. The RCQ field contains additional information.
- 6 Transfer-direction specification error: The command specified by the DCW designated by the DCW-offset field in the TSH specifies a direction of data transfer that disagrees with the transfer direction specified in the TCW. The RCQ field contains additional information.
- 7 Transport-count specification error: The RCQ field contains additional information.
- 8 Two I/O operations active: While an I/O operation is active at the device a second non-interrogate TCCB has been transported to the device for execution. The RCQ field has no meaning.
- 9 CBC-offset specification error: One or more CBC offsets in the CBC-offset block indicate that a CBC is at a location that is not recognized by the device or is not recognized for the designated DCW. The RCQ field contains additional information.
- 10-255 Reserved.

For a summary of the meanings of reason-code values and their associated reason-code-qualifier values see Figure 15-21 on page 15-49.

Reason Code Qualifier (RCQ): Words 1-4 may contain additional information about the reason for the device-detected program check as follows.

When the RC field contains the value one, only byte 0 of the RCQ field has meaning. For this case, byte 0 of the RCQ field contains an 8-bit unsigned integer code that further describes the condition indicated by the RC field and the meaning of each RCQ value is as follows:

Value Meaning

- 0 No additional information.
- 1 TCCB transport size error: The value of the TCCBL field of the TCW that was sent to the I/O device is not equivalent to the count of bytes actually transported for the TCCB.

- 2 TCCB CBC error: An invalid CBC was detected while transferring the TCCB.
- 3-255 Reserved.

When the RC field contains the value two, only words 0-1 of the RCQ field have meaning as follows:

Value Meaning

- 0 Word 0 contains the 32-bit unsigned integer offset of the first output-data byte for which the invalid CBC was detected.
- 1 Word 1 contains the 32-bit unsigned integer offset of the last byte of the last output-data word for which the invalid CBC was detected.

When the RC field contains the value three, only byte 0 of the RCQ field has meaning. For this case, byte 0 of the RCQ field contains an 8-bit unsigned integer code that further describes the condition indicated by the RC field and the meaning of each RCQ value is as follows:

Value Meaning

- 0 No additional information.
- 1 The value specified by the TCAL field is not 8 greater than the value specified by the TCCBL field in the TCW for the operation.
- 2 The value specified by the TCAL field is less than 20 or greater than 252.
- 3-255 Reserved.

When the RC field contains the value four, only byte 0 of the RCQ field has meaning. For this case, byte 0 of the RCQ field contains an 8-bit unsigned integer code that further describes the condition indicated by the RC field and the meaning of each RCQ value is as follows:

Value Meaning

- 0 No additional information.
- 1 Format-field specification error: The format field contains an unrecognized value. See "Format Control" on page 15-37.

- 2 Reserved-field specification error: A reserved field in the TCAH does not contain zeros.
- 3 Service-action-code-field specification error: The service-action-code field in the TCAH contains an unrecognized value or a value that is incorrect for command specified by the DCW designated by the DCW-offset field in the TSH. See “Service-Action Code (SAC)” on page 15-37.

4-255 Reserved.

When the RC field contains the value five, only byte 0 of the RCQ field has meaning. For this case, byte 0 of the RCQ field contains an 8-bit unsigned integer code that further describes the condition indicated by the RC field and the meaning of each RCQ value is as follows:

Value	Meaning
0	No additional information.
1	Reserved-field specification error: A reserved field in the DCW does not contain zeros.
2	Flags-field command-chaining specification error: Either of the following is true: <ul style="list-style-type: none"> The command-chaining flag is one and the offset of the next DCW is such that all or part of the next DCW extends past the end of the TCA. The command-chaining flag is zero and more than 3 unused bytes remain in the TCA. See “Chain-Command (CC) Flag” on page 15-38.
3	Control-data-count-field specification error: The CD-count field specifies control data past the end of the TCA. See “Control-Data (CD) Count” on page 15-38.
4	TCOB location error: The first TCOB DCW is not the first DCW in the TCA
5	TCOB duplication error: More than one TCOB DCW is specified.

- 6 TCOB multiple-count error: The control-data-count field and the count field in the TCOB DCW both specify the value zero or both specify a non-zero value.
- 7 TCOB direction error: A TCOB DCW is specified and the write-operation (W) in the TCW is zero.
- 8 TCOB chaining error: The chain-command bit in the TCOB DCW is zero.
- 9 TCOB count-specification error: Either of the following is true:
 - The count field in the TCOB is zero, the control-data-count field in the TCOB DCW is non-zero, and the control-data-count field in the TCOB specifies a value that is not an even multiple of four
 - The control-data-count field in the TCOB is zero, the count field in the TCOB DCW is non-zero, and the count field in the TCOB specifies a value that is not an even multiple of four,

10-255 Reserved.

When the RC field contains the value six, only byte 0 of the RCQ field has meaning. For this case, byte 0 of the RCQ field contains an 8-bit unsigned integer code that further describes the condition indicated by the RC field and the meaning of each RCQ value is as follows:

Value	Meaning
0	No additional information.
1	Read-direction specification error: The DCW specifies an input operation, but the R-bit in the TCW used to transport the TCCB is zero.
2	Write-direction specification error: The DCW specifies an output operation, but the W-bit in the TCW used to transport the TCCB is zero.
3-255	Reserved.

When the RC field contains the value seven, only byte 0 of the RCQ field has meaning. For this case, byte 0 of the RCQ field contains an 8-bit unsigned

integer code that further describes the condition indicated by the RC field and the meaning of each RCQ value is as follows:

<i>Value</i>	<i>Meaning</i>
0	No additional information.
1	Read-count specification error: The TCAT transport-count specifies a value that is not equivalent to the TCW input-count value or is not equivalent to the total count of data bytes specified by the DCWs in the TCA. See “Transport Count” on page 15-38.
2	Write-count specification error: The TCAT transport-count specifies a value that is not equivalent to the TCW output-count value or is not equivalent to the total count of data bytes specified by the DCWs in the TCA. See “Transport Count” on page 15-38.
3-255	Reserved.

When the RC field contains the value nine, only words 1 of the RCQ field has meaning as follows:

<i>Value</i>	<i>Meaning</i>
0	Word 0 contains the 32-bit unsigned integer byte offset of the first CBC-offset entry in the COB that specifies a CBC offset that is not at a location that is recognized by the device or is not recognized for the DCW

Sense Data: When the length field in the TSH contains a value greater than 32, sense data has been provided by the I/O device. The count of meaningful sense-data bytes in this field is determined by subtracting 32 from the value in the length field.

Summary of Reason Codes and Associated Reason-Code Qualifiers

Figure 15-21 on page 15-49 provides a summary of the device-detected-program-check RC values and the meanings of their associated RCQ values.

Reason Code Value	Meaning	
0	No information.	
1	TCCB Transport Failure.	
	Reason-Code Qualifier Byte 0 Value	Meaning
	0	No additional information.
	1	TCCB transport size error
	2	TCCB CBC error.
2	Invalid CBC detected on output data.	
	Reason-Code Qualifier	Meaning
	Word 0 Value:	Offset of first output-data byte for which the invalid CBC was detected.
	Word 1 Value:	Offset of last output-data byte for which the invalid CBC was detected.
3	Incorrect TCCB length specification.	
	Reason-Code Qualifier Byte 0 Value	Meaning
	0	No additional information.
	1	TCCB TCAL field value not 8 greater than TCW TCCBL field value.
	2	TCCB TCAL field value is less than 20 or greater than 252.
4	TCAH specification error.	
	Reason-Code Qualifier Byte 0 Value	Meaning
	0	No additional information.
	1	TCCB format field specification error.
	2	TCCB reserved field specification error.
	3	TCCB service-action-code field specification error.
5	DCW specification error.	
	Reason-Code Qualifier Byte 0 Value	Meaning
	0	No additional information.
	1	DCW reserved field specification error.
	2	DCW flags field command-chaining specification error.
	3	DCW control-data-count field specification error.
	4	TCOB location error
	5	TCOB duplication error
	6	TCOB multiple-count error
	7	TCOB direction error
	8	TCOB chaining error
	9	TCOB count-specification error

Figure 15-21. Summary of Reason-Code (RC) and Reason-Code-Qualifier (RCQ) Values in the Device-Detected-Program-Check TSA (Part 1 of 2)

Reason Code Value	Meaning	
6	Transfer-direction specification error.	
	Reason-Code Qualifier Byte 0 Value	Meaning
	0	No additional information.
	1	Read-direction specification error.
	2	Write-direction specification error.
7	Transport-count specification error.	
	Reason-Code Qualifier Byte 0 Value	Meaning
	0	No additional information.
	1	Read-count specification error.
	2	Write-count specification error.
8	Two I/O operations active.	
	Reason-Code Qualifier	No additional information.
9	CBC-offset-specification error.	
	Reason-Code Qualifier	No additional information.
	Word 0 Value:	Offset of first CBC-offset entry that specifies a CBC offset that is not at a location that is recognized.
Explanation: <ul style="list-style-type: none"> Reason-code values and reason-code-qualifier values that have no meaning do not appear in this summary. Portions of the reason-code-qualifier field that have no meaning for a specific reason code do not appear in this summary. 		

Figure 15-21. Summary of Reason-Code (RC) and Reason-Code-Qualifier (RCQ) Values in the Device-Detected-Program-Check TSA (Part 2 of 2)

Interrogate TSA

When the TSB-format field in the TSH contains three, the TSA has the following format:

Word (from the beginning of the TSA)

0	Format	Flags	CS	DS	
1	OS	Reserved			
2	State-Dependent Information				
4					
5	Device-Level Identifier				
6	Device-Dependent Data				
12					
	0	8	16	24	31

Figure 15-22. Interrogate Transport-Status Area (TSA)

Format: Byte of word 0 contains the unsigned integer value that defines the layout of the interrogate TSA. If the value of this field is not one, the contents of the interrogate TSA are meaningless.

Flags: Byte 1 of word 0 contains information about the interrogate TSA. The meaning of each flag bit is as follows.:

Bit Meaning

- | | |
|---|---|
| 0 | Control-unit-state valid: When bit 0 is one, the control-unit-state field contains meaningful information. When bit 0 is zero, the control-unit-state field has no meaning. |
| 1 | Device-state valid: When bit 1 is one, the device-state field contains meaningful information. When bit 1 is zero, the device-state field has no meaning. |

- 2 Operation-state valid: When bit 2 is one, the operation-state field contains meaningful information. When bit 2 is zero, the operation-state field has no meaning.
- 3-7 Reserved.

Control-Unit State (CS): Byte 2 of word 0 contains an 8-bit unsigned integer that indicates a current state of the control unit for the I/O device. The meaning of each value is as follows:

Value	Meaning
0	Busy: The control unit is busy and the device-dependent-data field may contain additional information about the busy state.
1	Recovery: The control unit is performing a recovery process and the device-dependent-data field may contain additional information about the recovery state.
2	Interrogate maximum: The control unit is executing the maximum number of interrogate operations that it supports
3-127	Reserved.
128-255	Device-dependent meanings.

Device State (DS): Byte 3 of word 0 contains an 8-bit unsigned integer that indicates a current state of the I/O device. The meaning of this byte is

Value	Meaning
0	Path-group identification: The state-dependent-information field contains information identifying a path group.
1	Long busy: The device is in a long-busy state. The meaning of long busy is device dependent and the device-dependent field may contain additional information about the long-busy state.
2	Recovery: The device is performing a recovery process and the device-dependent-data field may contain additional information about the recovery state.
3-127	Reserved.

- 128-255 Device-dependent meanings.

Operation State (OS): Byte 0 of word 1 contains an 8-bit unsigned integer that indicates whether an I/O operation is present at the device and, when present, the state of the operation. The meaning of this byte is as follows:

Value	Meaning
0	No I/O operation is present.
1	An I/O operation is present and executing.
2	An I/O operation is present and waiting for completion of an I/O operation that was initiated by another configuration.
3	An I/O operation is present and waiting for completion of an I/O operation that was initiated for the same device extent.
4	An I/O operation is present and waiting to perform a device-dependent operation.
5-127	Reserved.
128-255	Device-dependent meanings.

State-Dependent Information: Words 2-4 may contain state-dependent information. Whether this field has meaning is designated by the CS, DS, and OS fields.

The contents of this field are device dependent. When the device places one or more bytes in this field, they are left justified.

Device-Level Identifier: Word 5 contains a device-dependent token that identifies the implementation level of the device.

Device-Dependent Data: Words 6-12 contain device dependent information. Whether this field has meaning is designated by the CS, DS, and OS fields.

When the device places one or more bytes in this field, they are left justified.

Note:

1. An interrogate TSA is stored only in a TSB designated by an interrogate TCW.
2. If a program-check condition is recognized for an interrogate TCCB, a device-detect-program-check TSA is stored in the TSB designated by the interrogate TCW.

Command Code

For CCW channel programs, the command code, bit positions 0-7 of the CCW, specifies to the channel subsystem and the I/O device the operation to be performed. For TCW channel programs, the command code, bit positions 0-7 of the DCW, specifies to the I/O device the operation to be performed.

With the exception of the interrogate command, the two rightmost bits or, when these bits are zeros, the four rightmost bits of the command code identify the operation to the channel subsystem. The channel subsystem distinguishes among the following operations:

- Control
- Output forward (write)
- Input forward (read, sense, sense ID)
- Input backward (read backward)
- Branching (transfer in channel)
- Interrogate

For CCW channel programs, the channel subsystem ignores the leftmost four bits of the command code in a format-0 CCW that specifies transfer-in-channel. In a format-1 CCW that specifies transfer in channel, all bits of the command code are decoded by the channel subsystem.

For CCW channel programs, commands that initiate I/O operations (write, read, read backward, control, sense, and sense ID) cause all eight bits of the command code to be transferred to the control unit.

In command codes that initiate I/O operations, the leftmost bit positions contain modifier bits. The modifier bits specify to the device how the command is to be executed. They may, for example, cause the device to compare data received during a write operation with data previously recorded, and they may specify such attributes as recording density and parity. For the control command, the modifier bits may contain the order code specifying the control function

to be performed. The meaning of the modifier bits depends on the type of I/O device and is specified in the System Library publication for the device.

The command-code assignment is listed in Figure 15-23 on page 15-52. The symbol x indicates that the bit position is ignored; the symbol m identifies a modifier bit.

Code		Command
x x x x	0 0 0 0	Invalid
m m m m	m m 0 1	Write ¹
m m m m	m m 1 0	Read ¹
m m m m	1 1 0 0	Read Backward ²
m m m m	m m 1 1	Control
m m m m	0 1 0 0	Sense
1 1 1 0	0 1 0 0	Sense ID
x x x x	1 0 0 0	Transfer in channel ³
0 0 0 0	1 0 0 0	Transfer in channel ⁴
m m m m	1 0 0 0	Invalid ⁵
0 1 m m	0 0 0 0	Transport ⁶

Explanation:

m Modifier bit

x Ignored

¹ May designate control data in DCWs.

² CCW only

³ Format-0 CCW

⁴ Format-1 CCW

⁵ Format-1 CCW with any of bits 0-3 nonzero

⁶ DCW only

Figure 15-23. Command-Code Assignment

Whenever the channel subsystem detects an invalid command code during the initiation of command execution for a CCW channel program, the program-check-interruption condition is generated, and channel-program execution is terminated. The command code is ignored during data chaining, unless it specifies transfer in channel.

Whenever an I/O device detects an invalid command code while executing the commands in a TCCB, a device-detected-program-check condition is recognized and channel-program execution is terminated.

Designation of Storage Area

For CCW channel programs, the main-storage area associated with an I/O operation is defined by one or more CCWs. A CCW defines an area by specifying the address of the first byte to be transferred and the

number of consecutive bytes contained in the area. The address of the first byte of data to be transferred is specified either directly in the data-address field of the CCW or indirectly in an indirect-data-address word (IDAW) or modified-indirect-data-address word (MIDAW) designated by the data-address field of the CCW. The number of bytes contained in the storage area is specified in the count field.

For TCW channel programs the main-storage area associated with an I/O operation is designated by a TCW. A TCW designates an area by specifying the address of the first byte to be transferred and the number of consecutive bytes contained in the area. The address of the first byte of data to be transferred for input operations is specified either directly in the input-data-address field of the TCW or indirectly by a transport-indirect-data-address word (TIDAW) designated by the input-data-address field of the TCW. The number of bytes contained in the storage area for input operations is specified in the input-count field. The address of the first byte of data to be transferred for output operations is specified either directly in the output-data-address field of the TCW or indirectly by a TIDAW designated by the output-data-address field of the TCW. The number of bytes contained in the storage area for output operations is specified in the output-count field. In this case, the actual number of bytes in the storage area is determined by subtracting the number of any inserted padding or CBC bytes from the output-count field value. The address of the first byte of data to be transferred into the TSB is specified in the TSB-address field of the TCW. The number of bytes contained in the storage area for the TSB is dependent on the status being returned and is a maximum of 64.

In general, for write, read, control, and sense operations, storage locations are used in ascending order of addresses. As information is transferred to or from main storage, the address from the appropriate address field is incremented, and the count from the associated count field is decremented. For command-mode operations, the read-backward operation places data in storage in a descending order of addresses, and both the count and the address are decremented. When the associated count reaches zero, the storage area defined by the CCW is exhausted.

When a subchannel is operating in transport mode, the channel subsystem may take advantage of performance optimizations and access storage locations in a non-sequential order within the storage area or

areas designated by the current TCW. When a transport-mode operation is concluded, the transfer of specified data, as indicated by information in the I/O-status TSB, is complete.

Any main-storage location available to the start function can be used in the transfer of data to or from an I/O device, provided that the location is not protected against that type of reference. Format-0 CCWs can be located in any available part of the first 2^{24} (16M) bytes of absolute storage, and format-1 CCWs and TCWs can be located in any available part of the first 2^{31} (2G) bytes of absolute storage, provided that, in both cases, the location is not protected against a fetch-type reference. When the channel subsystem attempts to refer to a protected location, the protection-check condition is generated, and the device is signaled to terminate the operation.

A main-storage location is available if it is available in the configuration and access to the location is not prevented by the address-limit-checking facility. CCWs, TCWs, ISDs, TSBs, and the data they address reside in main-storage locations. If a main-storage location is not available, it is said to have an invalid address

If the channel subsystem refers to a location that is not available, the program-check condition is generated. When the first CCW or TCW designated by the channel-program address is at an unavailable location, the start function is not initiated at the device, the status portion of the SCSW is updated with the program-check indication, the subchannel becomes status pending with primary, secondary, and alert status, and deferred condition code 1 is indicated. An invalid data address, as well as any invalid CCW address detected on chaining or subsequent to the execution of START SUBCHANNEL, causes the channel subsystem to signal the device to conclude the operation, if and when the device requests or offers a byte of data or status at the invalid address. In this situation, the subchannel is made status pending with program check indicated in the subchannel status, and the device status is a function of the status received from the device. The program-check condition causes command chaining and command retry to be suppressed.

During an output operation, the channel subsystem may fetch data from main storage before the time the I/O device requests the data. Any number of bytes specified by the current CCW or TCW may be prefetched and buffered.

When data chaining during an output operation for a CCW channel program, the channel subsystem may fetch one CCW describing a data area at any time during the execution of the current CCW. If unlimited prefetching is allowed by the setting of the prefetch-control bit in the ORB CCW channel program, any number of CCWs, IDAWs, and MIDAWs and the associated data may be prefetched by the channel subsystem. When the I/O operation uses data or CCWs from locations near the end of the available storage, such prefetching may cause the channel subsystem to refer to locations that do not exist. Invalid addresses detected during prefetching do not affect the performance of the I/O operation and do not cause error indications until the operation actually attempts to use the information. If the operation is concluded by the I/O device or by the execution of HALT SUBCHANNEL or CLEAR SUBCHANNEL before the invalid information is needed, the condition is not brought to the attention of the program.

The count field in the CCW can specify any number of bytes up to 65,535. In format-0 CCWs, the count field is always nonzero unless the command code specifies transfer in channel, in which case the count field is ignored. In format-1 CCWs, the count field may contain the value zero unless data chaining is specified or the CCW is fetched while data chaining. Whenever (1) the count field in a format-1 CCW is zero, (2) data chaining is either not specified or not in effect, and (3) data transfer is requested by the device, the device is signaled to stop, and the I/O operation is terminated. The channel subsystem sets the incorrect-length condition if the SLI flag is not one in the CCW. No data is transferred. If the device does not request data transfer, the operation proceeds to the normal ending point.

If a zero byte count is contained in a format-0 CCW that does not specify transfer in channel, or if a zero byte count is contained in a format-1 CCW that specifies data chaining or was fetched while data chaining, a program-check condition is recognized, and the subchannel is made status pending with combinations of primary, secondary, and alert status as a function of the state of the subchannel and the status received from the device.

The input-count and output-count fields in the TCW can specify any number of bytes up to $2^{32} - 8$. For input operations, the input-count field is always nonzero; otherwise, a program-check condition is recognized. For output operations, the output-count field is

always nonzero; otherwise, a program-check condition is recognized.

Note: For a description of the storage area associated with a CCW when indirect data addressing is used, see “CCW Indirect Data Addressing” on page 15-61. For a description of the storage area associated with a TCW when indirect data addressing is used, see “Transport Indirect Data Addressing” on page 15-65.

Programming Notes:

1. Since a format-1 CCW with a count of zero is valid, the program can use the CCW count field to specify that no data be transferred to the I/O device. If the device requests a data transfer, the device is signaled to terminate data transfer. If the SLI and chain-command flags are also specified as ones, and no unusual conditions are encountered subsequent to signaling the device to terminate data transfer, the new operation is initiated upon receipt of device end from the device.
2. If the subchannel is in the incorrect-length-suppression mode, the chain-data flag in the current CCW is zero, and the operation is performed as an immediate operation, then incorrect length is not indicated, regardless of the setting of the SLI flag.

If the subchannel is in the incorrect-length-indication mode, if the chain-data flag in the current CCW is zero, and if the operation is performed as an immediate operation, then incorrect length is indicated if the count field of the current CCW specifies a nonzero value, unless suppressed by the SLI flag of the CCW; incorrect length is not indicated, however, if the count field of the CCW specifies a value of zero.

If a new CCW that has a count field of zero is fetched during data chaining or if a CCW is fetched with the chain-data flag set to one and a count field of zero, a program-check condition is recognized by the channel subsystem.

3. Since the channel-subsystem may access storage in a non-sequential manner while a subchannel is operating in transport mode, the results may be unpredictable if the program accesses the input storage area or areas designated by a TCW before primary status is indicated.

CCW Channel Program Chaining

When the channel subsystem has completed the transfer of information specified by a CCW, it can continue performing the start function by fetching a new CCW. Such fetching of a new CCW is a form of chaining, and the CCWs belonging to such a sequence are said to be chained.

Chaining takes place between CCWs located in successive doubleword locations in storage. It proceeds in an ascending order of addresses; that is, the address of the new CCW is obtained by adding 8 to the address of the current CCW. Two chains of CCWs located in noncontiguous storage areas can be coupled for chaining purposes by a transfer-in-channel command. All CCWs in a chain apply to the I/O device that is associated with the subchannel designated by the original START SUBCHANNEL instruction.

Two types of chaining are provided for CCWs: Chaining is controlled by the chain-data (CD) and chain-command (CC) flags in conjunction with the suppress-length-indication (SLI) flag in the CCW. These flags specify the action to be taken by the channel subsystem upon the exhaustion of the current CCW and upon receipt of ending status from the device, as shown in Figure 15-24 on page 15-56.

The specification of chaining is effectively propagated through a transfer-in-channel command. When, in the process of chaining, a transfer-in-channel command is fetched, the CCW designated by the transfer-in-channel command is used for the type of chaining specified in the CCW preceding the transfer-in-channel command.

The CD and CC flags are ignored in a format-0 CCW specifying the transfer-in-channel command. In a format-1 CCW specifying the transfer-in-channel com-

mand, the CD and CC flags must be zeros; otherwise, a program-check condition is recognized.

Flags in Current CCW			Action at the Subchannel upon Exhaustion of Count or Receipt of Channel End							
			Immediate Operation				Non-immediate Operation			
			Incorrect-Length-Suppression Mode ¹		Incorrect-Length-Indication Mode		Count Exhausted		Count Not Exhausted and CE Received	
			CCW Count≠0	CCW Count=0	CCW Count≠0	CCW Count=0	Ce Not Received	Ce Received		
CD	CC	SLI								
0	0	0	End, NIL	End, NIL	End, IL	End, NIL	Stop, IL	End, NIL	End, IL	
0	0	1	End, NIL	End, NIL	End, NIL	End, NIL	Stop, NIL	End NIL,	End, NIL	
0	1	0	CC	CC	End, NIL	CC	Stop, IL	CC	End, IL	
0	1	1	CC	CC	CC	CC	Stop, CC	CC	CC	
1	-	-	End, NIL	PC	End, IL	PC	CD	*	End IL	

Explanation:

- The selected bit is ignored and may be either zero or one.

* These situations cannot validly occur. When data chaining is specified, the new CCW takes control of the operation after transferring the last byte of data designated by the current CCW, but before the next request for data or status transfer from the device. The new CCW (which cannot contain a count of zero unless a program-check condition is also recognized) is in control of the operation

¹ The count field must contain a nonzero value when format-0 CCWs are specified; otherwise, the operation is terminated with a program-check condition.

CC Command chaining is performed by the channel subsystem upon receipt of device end.

CD The chain-data flag causes the channel subsystem to immediately fetch a new CCW for the same operation. The operation continues unless the CCW thus fetched has a count field of zero, in which case the operation is terminated with a program-check condition.

CE Channel end from the device that indicates end of block.

End Operation is terminated.

IL Incorrect length is indicated with the subsequent interruption condition generated at the subchannel.

NIL Incorrect length is not indicated with the subsequent interruption condition generated at the subchannel.

PC These situations cannot validly occur. The channel subsystem recognizes a program-check condition when a CCW is fetched that has the chain-data flag set to one and a count field of zero.

STOP Device is signaled to terminate data transfer, but subchannel remains subchannel active until channel end is received.

Figure 15-24. Subchannel Chaining Action

Programming Note: When bit 9 of word 1 of the ORB is one, unlimited prefetching of chained CCWs (including CCWs linked by a transfer-in-channel command) by the channel subsystem is permitted. When prefetching is allowed by the ORB, no modification of the channel program should be performed after START SUBCHANNEL is executed and before the primary interruption condition for the operation has been received unless the subchannel is currently suspended and is not resume pending.

Data Chaining

During data chaining, the new CCW fetched by the channel subsystem defines a new storage area for the original I/O operation. If the channel path is of the parallel-I/O-interface type, the performance of the operation at the I/O device is not affected. If the channel path is of the serial-I/O-interface type, then

the performance of the operation at the I/O device either is not affected or, depending on the device model, may be terminated with unit-check status. When the operation at the I/O device is not affected and all data designated by the current CCW has been transferred to main storage or to the device, data chaining causes the operation to continue, using the storage area designated by the new CCW. The contents of the command-code field of the new CCW are ignored, unless they specify transfer in channel.

Data chaining is considered to occur immediately after the last byte of data designated by the current CCW has been transferred to main storage or to the device. When the last byte of the data transfer has been placed in main storage or accepted by the device, the new CCW takes over the control of the operation. If the device sends channel end after

exhausting the count of the current CCW but before transferring any data to or from the storage area designated by the new CCW, the SCSW associated with the concluded operation pertains to the new CCW.

If programming errors are detected in the new CCW or during its fetching, the error indication is generated, and the device is signaled to conclude the operation when it attempts to transfer data designated by the new CCW. If the device signals the channel-end condition before transferring any data designated by the new CCW, program check or protection check is indicated in the SCSW associated with the termination. The contents of the SCSW pertain to the new CCW unless the address of the new CCW is invalid, the location is protected against fetching, or programming errors are detected in an intervening transfer-in-channel command. A data address referring to a nonexistent or protected area causes an error indication only after the I/O device has attempted to transfer data to or from the invalid location.

Data chaining during an input operation causes the new CCW to be fetched when all data designated by the current CCW has been placed in main storage. On an output operation, the channel subsystem may fetch the new CCW from main storage before data chaining occurs. Any programming errors in the prefetched CCW, however, do not affect the performance of the operation until all data designated by the current CCW has been transferred to the I/O device. If the device concludes the operation before all data designated by the current CCW has been transferred, the conditions associated with the prefetched CCW are not indicated to the program. Unlimited prefetching is allowed under the control of the prefetch bit specified in the ORB. (See "Prefetch Control (P)" on page 15-27.) When unlimited prefetching is not allowed and an output operation is specified, only one CCW describing a data area may be prefetched. If a prefetched CCW specifies transfer in channel, only one more CCW may be fetched before the exhaustion of the current CCW.

Programming Notes:

1. If the ORB does not specify unlimited prefetching, no prefetching of CCWs is performed, except in the case of data chaining on an output operation where one CCW describing a data area may be prefetched at a time.

If the ORB for the I/O operation specifies that prefetching is allowed, any number of CCWs,

IDAWs, and MIDAWs and the associated data areas may be prefetched and buffered in the channel subsystem.

The same actions for signaling errors and terminating operations take place when unlimited prefetching is allowed by the ORB as when it is not allowed. However, when unlimited prefetching is specified and an error condition is detected, both the channel subsystem and the program must recognize that the points of termination at the channel subsystem and at the I/O device may be different in terms of the channel command in execution at the point of error. The channel subsystem indicates the point of termination at the channel subsystem by storing the appropriate CCW address in word 1 of the sub-channel-status word and the point of termination at the device by storing the secondary-CCW address in word 4 of the format-0 extended-status word.

When prefetching has been specified in the ORB, the result of modifications to CCWs after START SUBCHANNEL has been executed or after self-describing channel programs have been used is unpredictable. (See note 2 for the definition of self-describing channel programs.)

2. Data chaining may be used to rearrange information as it is transferred between main storage and an I/O device. Data chaining permits blocks of information to be transferred to or from non-contiguous areas of storage, and, when used in conjunction with the skipping function, data chaining allows the program to place in main storage specified portions of a block of data.

When, during an input operation, the program specifies data chaining to a location in which data has been placed under the control of the current CCW, the channel subsystem, in fetching the next CCW, fetches the new contents of the location. This is true even if the location contains the last byte transferred under the control of the current CCW. When a channel program data-chains to a CCW placed in storage by the CCW specifying data chaining, the input block is said to be self-describing. A self-describing block contains one or more CCWs that designate storage locations and counts for subsequent data in the same input block.

The use of self-describing blocks is equivalent to the use of unchecked data. An I/O data-transfer

malfunction that affects validity of a block of information is signaled only at the completion of data transfer. The error condition normally does not prematurely terminate or otherwise affect the performance of the operation. Thus, there is no assurance that a CCW read as data is valid until the operation is completed. If the CCW thus read is in error, use of the CCW in the current operation may cause subsequent data to be placed at wrong locations in main storage with resultant destruction of its contents, subject only to the control of the protection key and the address-limit-checking facility, if used.

3. When, during data chaining, a device transfers data by using the data-streaming feature, an overrun or chaining-check condition may be recognized when a small byte-count value is specified in the CCW. The minimum acceptable number of bytes that can be specified varies as a function of the system model and system activity.

Command Chaining

During command chaining, the new CCW fetched by the channel subsystem specifies a new I/O operation. The channel subsystem fetches the new CCW upon the receipt of the device-end signal for the current operation. If the new CCW does not have its S flag set to one and no unusual conditions are detected, the channel subsystem initiates the new operation. The presence of the S flag set to one or unusual conditions causes command chaining to be suppressed. When command chaining takes place, the completion of the current operation does not cause an I/O interruption, and the count indicating the amount of data transferred during the current operation is not made available to the program. For operations involving data transfer, the new command always applies to the next block of data at the device.

Command chaining takes place and the new operation is initiated only if no unusual conditions have been detected in the current operation. In particular, the channel subsystem initiates a new I/O operation by command chaining upon receipt of a status byte containing only the following bit combinations: (1) device end, (2) device end and status modifier, (3) device end and channel end, and (4) device end, channel end, and status modifier. In the first two cases, channel end is signaled before device end, with all other status bits zeros. If a condition such as attention, unit check, unit exception, incorrect length, program check, or protection check has occurred, the sequence of operations is concluded, and the status

associated with the current operation causes an interruption condition to be generated. The new CCW in this case is not fetched. The incorrect-length condition does not suppress command chaining if the current CCW has the SLI flag set to one.

An exception to sequential chaining of CCWs occurs when the I/O device presents the status-modifier condition with the device-end signal or channel-end and device-end signals. When command chaining is specified and no unusual conditions have been detected, or when command retry has been previously signaled and an immediate retry could not be performed, the combination of status-modifier and device-end bits causes the channel subsystem to alter the sequential execution of CCWs. If command chaining was specified, status modifier and device end cause the channel subsystem to fetch and chain to the CCW whose main-storage address is 16 higher than that of the CCW that specified chaining. If command retry was previously signaled and immediate retry could not be performed, the status causes the channel subsystem to command chain to the CCW whose storage address is 8 higher than that of the CCW for which retry was initially signaled.

When both command and data chaining are specified, the first CCW associated with the operation specifies the operation to be performed, and the last CCW specifies whether another operation follows.

Programming Note: Command chaining makes it possible for the program to initiate transfer of multiple blocks of data by issuing a single START SUBCHANNEL instruction. It also permits a subchannel to be set for execution of other commands, such as positioning the disk-access mechanism, and for data-transfer operations without interference by the program at the end of each operation. Command chaining, in conjunction with the status-modifier condition, permits the channel subsystem to modify the normal sequence of operations in response to signals provided by the I/O device.

TCW Channel Program Chaining

When the I/O device has completed a control operation or the transfer of information specified by a DCW, the device can continue by executing the next DCW that is specified. Such execution of a new DCW is a form of command chaining, and the DCWs belonging to such a sequence are said to be chained.

Command chaining is controlled by the DCW chain-command (CC) flag in the DCW. This flag specifies the action to be taken upon the exhaustion of the current DCW. Chaining takes place between successive DCWs within the TCA.

When a TCCB is transported to an I/O device and selected by the device for processing, the first DCW in the TCA becomes the current DCW and is executed by the device. When the current DCW specifies command chaining and no unusual conditions have been detected during the operation, the completion of the current DCW causes the next DCW to become the current DCW and be executed by the device. The TCA offset of the next DCW is determined by adding 8, plus the value of the control-data (CD) count field of the DCW, to the TCA offset of the current DCW and rounding upwards to the next word boundary. Thus, command chaining proceeds in ascending order of TCA offsets.

During command chaining, the new DCW executed by the device specifies a new I/O command. When command chaining takes place, the completion of the execution of the current command does not cause an I/O interruption, and the count indicating the amount of data transferred during the current operation is not made available to the program. For operations involving data transfer, the new command always applies to the next block of data at the device.

Command chaining takes place and the new command is initiated only if no unusual conditions have been detected in the execution of the current command. In particular, the I/O device initiates a new I/O operation by command chaining upon successful completion of the current command. If a condition such as attention, unit check, unit exception, or incorrect length has occurred, the sequence of operations is concluded, and the status associated with the current operation causes an interruption condition to be generated. The new DCW in this case is not executed.

Command chaining makes it possible for the program to initiate transfer of multiple blocks of data by issuing a single START SUBCHANNEL instruction. It also permits a subchannel to be set for execution of other commands, such as positioning the disk-access mechanism, and for data-transfer operations without interference by the program at the end of each operation. For command-mode operations, command chaining, in conjunction with the status-modifier condition, permits the channel subsystem to modify the

normal sequence of operations in response to signals provided by the I/O device

Skipping

Skipping causes the suppression of main-storage references during an I/O operation. It is defined only for read, read-backward, sense-ID, and sense operations. Skipping is controlled by the skip flag, which can be specified individually for each CCW, MIDAW, or TIDAW. When the skip flag is one, skipping occurs; when it is zero, normal operation takes place. The setting of the skip flag is ignored in all other operations.

Skipping affects only the handling of information by the channel subsystem. The operation at the I/O device proceeds normally, and information is transferred. The channel subsystem keeps updating the count but does not place the information in main storage. Chaining is not precluded by skipping. In the case of CCW data chaining, normal operation is resumed if the skip flag in the new CCW is zero.

No checking for invalid or protected data addresses takes place during skipping.

Programming Note: Skipping, when combined with CCW data chaining, MIDAWs, and TIDAWs, permits the program to place in main storage specified portions of a block of information from an I/O device.

Program-Controlled Interruption

The program-controlled-interruption (PCI) function permits the program to cause an I/O interruption during the performance of an I/O operation. The function is controlled by the PCI flag of the CCW. Neither the value of the PCI flag nor the associated interruption request affects the performance of the current operation.

The value of the PCI flag can be one either in the first CCW designated for the current start or resume function or in a CCW fetched during chaining. If the PCI flag is one in a CCW that has become current, the subchannel becomes status pending with intermediate status, and an I/O-interruption request is generated. The point at which the subchannel becomes status pending depends on the progress of the current start or resume function as follows:

1. If the PCI flag is one in the first CCW associated with a start function or a resume function, the subchannel becomes status pending with intermediate status only after the command has been accepted.
2. If the PCI flag is one in a CCW that has become current while data chaining, the subchannel becomes status pending with intermediate status after all data designated by the preceding CCW has been transferred.
3. If the PCI flag is one in a CCW that has become current while command chaining, the subchannel becomes status pending with intermediate status as that CCW becomes current.

In all cases, if the subchannel is enabled for I/O interruptions, the point of interruption depends on the current activity in the system and may be delayed. No predictable relationship exists between the point at which the interruption request is generated because of the PCI flag and the extent to which data transfer has been completed to or from the area designated by the CCW. However, all the fields within the SCSW pertain to the same instant.

An intermediate interruption condition that is made pending because of a PCI flag remains pending during chaining if not cleared by TEST SUBCHANNEL or CLEAR SUBCHANNEL. If another CCW containing a PCI flag that is one becomes current prior to the clearing of the intermediate interruption condition, only one interruption condition is preserved.

An intermediate interruption may occur while the subchannel is subchannel-and-device active with the operation specified by the CCW causing the intermediate interruption condition or with the operation specified by a CCW that has subsequently become current. If the intermediate interruption condition is not cleared prior to the conclusion of the operation or chain of operations, the condition is indicated together with the primary interruption condition at the conclusion of the operation or chain of operations. The intermediate interruption condition may be cleared by TEST SUBCHANNEL while the subchannel is subchannel active.

If the SCSW stored by TEST SUBCHANNEL indicates that the subchannel is status pending with intermediate status and the operation or chain of operations has not been concluded (that is, the activ-

ity-control field indicates subchannel-and-device active or suspended), then the CCW-address field contains an address that is 8 higher than the address of the most recent CCW to become current and have a PCI flag that is one, or the CCW-address field contains an address that is 8 higher than the address of a CCW that has subsequently become current. Unless the SCSW also contains the primary-status bit set to one, the device-status field contains zeros, and the count is unpredictable.

Subchannel-status conditions other than PCI may be indicated when the SCSW is stored. If the subchannel is not also status pending with primary status, these conditions may or may not be indicated again. If the subchannel-status condition is detected while prefetching and the operation or chain of operations is concluded before the condition affects an operation, the condition is reset and is not indicated when the subchannel subsequently becomes status pending with primary status. If the subchannel-status condition affects an operation, the condition is indicated when the subchannel becomes status pending with primary status.

If the program-controlled-interruption condition remains pending until the operation or chain of operations is concluded at the subchannel, a single interruption request exists. When TEST SUBCHANNEL is subsequently executed, the status-control field of the SCSW stored indicates both the primary interruption condition and the intermediate interruption condition, and the PCI bit of the subchannel-status field is one.

The value of the PCI flag is inspected in every CCW except for those CCWs that specify the transfer-in-channel command. The PCI flag is ignored during initial program loading.

Programming Notes:

1. The program-controlled interruption provides a means of alerting the program to the progress of chaining during an I/O operation. It permits programmed dynamic main-storage allocation.
2. A CCW with a PCI flag set to one may, if retried because of command retry, cause multiple PCI interruptions to occur. (See "Command Retry" on page 15-72.)

Indirect Storage Addressing

Indirect-storage addressing (ISA) allows for storage areas designated by a channel program to be specified in noncontiguous storage. Depending on the model, more than one form of ISA facility may be installed as follows:

- CCW-indirect-data (see “CCW Indirect Data Addressing” on page 15-61) addressing is installed in all models.
- Modified-CCW-indirect-data (see “Modified CCW Indirect Data Addressing” on page 15-63) addressing may also be installed in a model.
- TCW-indirect-data-addressing (see “Transport Indirect Data Addressing” on page 15-65) may also be installed in a model.

Indirect-Storage Designator (ISD)

Each ISA facility description includes the definition of a facility-unique indirect-storage designator (ISD) that is used to specify a block of storage to the channel subsystem. Thus, a list of ISDs may be used to designate noncontiguous storage blocks. How the location and size of a storage block are specified and whether the specification of the storage block has boundary restrictions, size restrictions, or both is dependent on the method of ISA used. The following ISDs may be used as follows:

- Format-1 and format-2 indirect-data-address word (IDAW). See “CCW Indirect Data Addressing” on page 15-61.
- Modified-indirect-data-address word (MIDAW), when the modified-CCW-indirect-data-addressing facility is installed. See “Modified CCW Indirect Data Addressing” on page 15-63.
- Transport-indirect-data-address word (TIDAW), when the fibre-channel-extensions (FCX) facility is installed. See “Transport Indirect Data Addressing” on page 15-65.

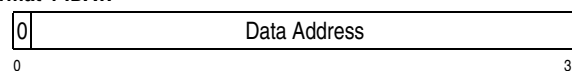
CCW Indirect Data Addressing

CCW indirect data addressing permits a single channel-command word to control the transfer of data that spans noncontiguous 2K-byte or 4K-byte blocks in main storage. The use of CCW indirect data addressing also allows the program to designate data

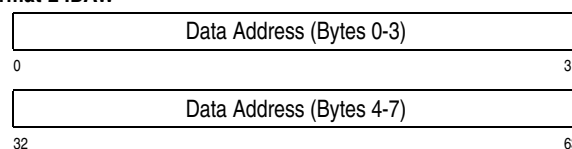
addresses above 16M bytes when using format-0 CCWs or above 2G bytes when using format-1 CCWs. CCW indirect data addressing is specified by a flag in the CCW which, when one, indicates that the data address is not used to directly address data. Instead, the address points to a list of words or doublewords, called indirect-data-address words (IDAWs), each of which contains an absolute address designating a data area in main storage.

IDAWs have either of two formats, called format 1 and format 2, as determined by the format-2-IDAW control, bit 14 of word 1 of the ORB associated with the channel program being executed. When the format-2-IDAW control is zero, the IDAW is format 1 and is a word containing a 31-bit address. When the control is one, the IDAW is format 2 and is a doubleword containing a 64-bit address. The IDAW formats are as follows:

Format-1 IDAW



Format-2 IDAW



Bit 0 (format 1) is reserved for future use and must be zero; otherwise, a program-check condition may be recognized, as described below.

A format-1 IDAW designates a data area within a 2K-byte block of main storage and is capable of addressing storage in the range of 0 to $2^{31} - 1$.

A format-2 IDAW designates a data area within a 2K-byte or 4K-byte block of main storage, as determined by the 2K-IDAW control, bit 15 of word 1 of the ORB associated with the channel program being executed, and is capable of addressing storage in the range of 0 to $2^{64} - 1$. When the 2K-IDAW-control bit is zero, each format-2 IDAW of the designated channel program designates a 4K-byte block of main storage. When the 2K-IDAW-control bit is one, each format-2 IDAW designates a 2K-byte data-area block. All IDAWs associated with the designated channel program must have the same IDAW format, and all of those IDAWs specify the same size of storage block.

When the indirect-data-addressing bit in the CCW is one, the data-address field of the CCW designates the location of the first IDAW to be used for data transfer for the command. Additional IDAWs, if needed for completing the data transfer for the CCW, are in successive locations in storage. The number of IDAWs required for a CCW is determined by the IDAW format as specified in the ORB, by the count field of the CCW, and by the data address in the initial IDAW. When, for example, (1) the ORB specifies format-2 IDAWs with 4K-byte blocks, (2) the CCW count field specifies 8K bytes, and (3) the first IDAW designates a location in the middle of a 4K-byte block, then three IDAWs are required.

The IDAW designated by the CCW can designate any location. Data is then transferred, for read, write, control, sense ID, and sense commands, to or from successively higher storage locations or, for a read-backward command, to successively lower storage locations, until a 2K-byte block boundary (format-1 or format-2 IDAW) or a 4K-byte block boundary (format-2 IDAW) is reached. The control of data transfer is then passed to the next IDAW. The second and any subsequent IDAWs must designate, depending on the command, the first byte, or the last byte for read backward, of a 2K-byte block (format-1 or format-2 IDAW) or a 4K-byte block (format-2 IDAW). Thus, for read, write, control, sense ID, and sense commands, such format-1 IDAWs must have zeros in bit positions 21-31, and such format-2 IDAWs must have zeros in bit positions 53-63 (2K-byte blocks) or 52-63 (4K-byte blocks). For a read-backward command, such format-1 IDAWs must have ones in bit positions 21-31, and such format-2 IDAWs must have ones in bit positions 53-63 (2K-byte blocks) or 52-63 (4K-byte blocks). If any of these rules is violated, a program-check condition is recognized.

Except for the unique restrictions on the designation of the data address by the IDAW, all other actions taken for the data address, such as for protected storage and invalid addresses, and the actions taken for data prefetching are the same as when indirect data addressing is not used.

IDAWs pertaining to the current CCW or a prefetched CCW may be prefetched. The number of IDAWs that can be prefetched cannot exceed that required to satisfy the count in the CCW that points to the IDAWs. An IDAW takes control of data transfer when the last byte has been transferred for the previous IDAW. The same actions take place as with data chaining regarding when an IDAW takes control of

data transfer during an I/O operation. That is, when the count for the CCW has not reached zero, a new IDAW takes control of the data transfer when the last byte has been transferred for the previous IDAW for that CCW, even in situations where (1) channel end, (2) channel end and device end, or (3) channel end, device end, and status modifier are received prior to the transfer of any data bytes pertaining to the new IDAW.

A prefetched IDAW does not take control of an I/O operation if the count in the CCW has reached zero with the transfer of the last byte of data for the previous IDAW for that CCW. Program or access errors detected in prefetched IDAWs are not indicated to the program until the IDAW takes control of data transfer. However, when the channel subsystem detects an invalid CBC on the contents of a prefetched IDAW or its associated key, the condition may be indicated to the program, when detected, before the IDAW takes control of data transfer. For a description of the indications provided when an invalid CBC is detected on the contents of an IDAW or its associated key, see "Channel-Control Check" on page 16-29.

Bits 1-31 (format 1) or bits 0-63 (format 2) designate the absolute storage location of the first byte to be used in the data transfer. When format-1 IDAWs are specified, the channel subsystem forms a 64-bit absolute main-storage address by appending 33 zero bits on the left of bit 1.

When the IDAW flag of the CCW is set to one and any of the following conditions occurs:

1. Format-1 IDAWs are specified in the ORB, and the address in the CCW does not designate the first IDAW on a word boundary,
2. Format-2 IDAWs are specified in the ORB, and the address in the CCW does not designate the first IDAW on a doubleword boundary,
3. The address in the CCW designates a storage location that is not physically available,
4. Access to the storage location specified by the address in the CCW is prohibited by protection, or
5. Bit 0 (format 1 only) of the first IDAW is not zero,

then, depending on the model, one of the following two actions is taken independent of the setting of the

skip flag (if condition 5 above is true, action 2 must be taken).

1. The above conditions are checked before initiating the operation at the device. If any of these conditions is recognized, initiation of the I/O operation does not occur, and the subchannel is made status pending with primary, secondary, and alert status.
2. The operation is initiated at the device prior to checking for these conditions. If the device attempts to transfer data, the device is signaled to terminate the I/O operation, and the subchannel is made status pending with primary, secondary, and alert status as a function of the subchannel state and the status presented by the device.

Modified CCW Indirect Data Addressing

Modified CCW indirect data addressing (MIDA) permits a single channel-command word to control the transfer of up to 65,535 bytes of data that spans non-contiguous blocks in main storage. Each block of main storage to be transferred may be specified on any boundary and length up to 4K bytes, provided the specified block does not cross a 4K-byte boundary. The use of modified CCW indirect data addressing requires that the program designate 64-bit data addresses.

Modified CCW indirect data addressing is controlled by a flag in the ORB and specified by a flag in the CCW which, when both are one, indicates that the CCW data address is not used to directly address data. Instead, the address points to a contiguous list of up to 256 quadwords called modified-indirect-data-address words (MIDAWs), each of which contains flags, a byte count, and a 64-bit address designating a data area in absolute storage.

Use of modified CCW indirect data addressing may be restricted to certain channel-path types. If the MIDA flag in the CCW is one, specifying the use of modified CCW indirect data addressing, and the subchannel is associated with channel paths that do not support modified CCW indirect data addressing, a program check is recognized.

The MIDAW begins on a quadword boundary and has the following format:

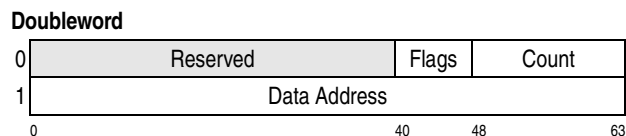


Figure 15-25. Modified-Indirect-Data-Address Word (MIDAW)

Reserved : Bits 0-39 of the MIDAW are reserved for future use and must be zero; otherwise, a program-check condition may be recognized.

Flags : Bits 40-47 of the MIDAW contain an 8-bit flag field. The meaning of each flag bit is as follows. Bits not shown are reserved.

Bit Meaning

40 Last MIDAW: Bit 40, when one, specifies that the MIDAW is the last in the contiguous list of MIDAWs.

41 Skip: Bit 41, when one, specifies the suppression of transfer of information to main storage during a read, read-backward, sense ID, or sense operation, thus specifying that skipping is in effect. When the operation is not read, read-backward, sense ID, or sense, bit 41 is ignored and skipping is not in effect

When skipping is in effect, no checking for invalid or protected data addresses takes place.

42 Data-transfer-interruption control: Bit 42, when one, specifies that a program-check be recognized when the device attempts to transfer data.

Count : Bits 48-63 of the MIDAW specify the number of bytes in the storage area designated by the data-address field. The count value must not be zero; otherwise a program-check condition is recognized. When skipping is in effect, the count value may be in the range of 1 - 65,535. When skipping is not in effect, the count value, in conjunction with the data address, must not specify the transfer of data that crosses a 4K-block boundary; otherwise, a program-check condition is recognized. When the count value causes the total data transfer count to exceed that specified in the CCW count field, a program-check condition is recognized.

Data Address : Bits 64-127 of the MIDAW designate the 64-bit address of a location in absolute storage which is the first byte of information to be transferred. If the count field specifies zero, this field is not checked.

When modified CCW indirect data addressing is specified, the data-address field of the CCW designates the location of the first MIDAW to be used for data transfer for the CCW command. Additional MIDAWs, if needed for completing the data transfer for the CCW, are fetched from successive locations in storage. The number of MIDAWs required for a CCW is determined by the count field of the CCW in relation to the count fields in the list of MIDAWs designated by the CCW. The list must not cross a 4K-byte boundary.

If a list of two or more MIDAWs is designated for a read operation, it is unpredictable in which order the MIDAW data addresses are used to transfer information into main storage. Thus, if two or more MIDAWs in a MIDAW list designate overlapping storage areas, the results are unpredictable.

MIDAWs that specify the transfer of data (MIDAWs for which skipping is not in effect) may specify a transfer count in the range of 1 - 4096 bytes, provided the MIDAW does not specify a data transfer that crosses a 4K-byte boundary; MIDAWs that specify the suppression of the transfer of data (MIDAWs for which skipping is in effect) may specify a skip count in the range of 1 - 65,535 bytes. The total number of bytes that can be transferred or skipped or both by a single MIDAW list is limited by the CCW count field, which is a maximum of 65,535 bytes. The sum of the count fields in all of the MIDAWs in the list designated by the CCW should equal the value in the count field of the CCW; otherwise, a program-check condition may be recognized.

The MIDAW designated by the CCW can designate any location. When the MIDAW skip flag is zero and the CCW specifies the read, write, control, sense ID, or sense command, data is then transferred to or from successively higher storage locations until the number of bytes specified by the MIDAW count field have been transferred. When the MIDAW skip flag is zero and the CCW specifies the read backwards command, data is then transferred to successively lower storage locations until the number of bytes specified by the MIDAW count field have been transferred. When the MIDAW skip flag is one and the CCW specifies the read, sense ID, or sense com-

mand, skipping occurs to successively higher storage locations until the number of bytes specified by the MIDAW count field have been skipped. When the MIDAW skip flag is one and the CCW specifies the read backwards command, skipping occurs to successively lower storage locations until the number of bytes specified by the MIDAW count field have been skipped.

When the specified number of bytes have been transferred or skipped and the CCW specifies a subsequent MIDAW, the control of data transfer is then passed to the next MIDAW in the list. Like the MIDAW designated by the CCW, subsequent MIDAWs may designate any location and any length, provided the MIDAW does not specify a data transfer that crosses a 4K-byte boundary; otherwise, a program-check condition is recognized.

Except for the unique restriction on designating a valid combination of data address and count by a MIDAW and the resulting behavior when the data-transfer-interruption control is one, all other actions taken for data addresses specified by a MIDAW, such as for protected storage and invalid addresses, and the actions taken for data prefetching, are the same as when modified CCW indirect data addressing is not used.

MIDAWs pertaining to the current CCW or a prefetched CCW may be prefetched. The number of MIDAWs that can be prefetched cannot exceed that required to satisfy the count in the CCW that designates the MIDAWs. Any MIDAWs that are prefetched for a CCW and are not used are discarded.

Similar to when CCW data chaining occurs, the action of transferring control from one MIDAW to the next is transparent to any attached device. A MIDAW takes control of data transfer when the last byte specified by the previous MIDAW has been transferred, even in situations where (1) channel end, (2) channel end and device end, or (3) channel end, device end, and status modifier are received prior to the transfer of any data bytes pertaining to the new MIDAW.

A prefetched MIDAW does not take control of an I/O operation if the count in the CCW has reached zero with the transfer of the last byte of data for the previous MIDAW. Program or access errors detected in prefetched MIDAWs are not indicated to the program until the MIDAW takes control of data transfer, even if an attempt had been made to prefetch that data. However, when the channel subsystem detects an

invalid CBC on the contents of a prefetched MIDAW or its associated key, the condition may be indicated to the program, when detected, before the MIDAW takes control of data transfer. For a description of the indications provided when an invalid CBC is detected on the contents of a MIDAW or its associated key, see “Channel-Control Check” on page 16-29.

When a CCW specifies a MIDAW list and either the address in the CCW designates a storage location that is not physically available or the address designates a storage location to which access is prohibited by protection, then, depending on the model, one of the following two actions is taken:

- The above conditions are checked before initiating the operation at the device. If any of these conditions is recognized, initiation of the I/O operation does not occur, and the subchannel is made status pending with primary, secondary, and alert status.
- The operation is initiated at the device prior to checking for these conditions. If the device attempts to transfer data, the device is signaled to terminate the I/O operation, and the subchannel is made status pending with primary, secondary, and alert status as a function of the subchannel state and the status presented by the device.

Transport Indirect Data Addressing

When a designated subchannel is operating in transport mode, transport indirect data addressing (TIDA) may be used. TIDA permits a TCW to specify the transfer of data from noncontiguous blocks in main storage or to specify the transfer-command-control block (TCCB) in noncontiguous blocks of storage or both.

The use of transport indirect data addressing is controlled by flags in the TCW as follows:

When the input-TIDA flag is one in the TCW, the TCW input-data address is not used to directly address data. Instead, the TCW input-data address points to a list of quadwords called transport-indirect-data-address words (TIDAWs), each of which contains flags, a byte count, and a 64-bit address designating a data area in absolute storage.

When the output-TIDA flag is one in the TCW, the TCW output-data address is not used to directly address data. Instead, the TCW output-data address points to a list of TIDAWs, each of which designates a data area in absolute storage.

When the transport-command-control-block-TIDA flag is one in the TCW, the TCW transport-command-control-block address is not used to directly address a TCCB. Instead, the TCW transport-command-control-block address points to a list of TIDAWs, each of which designates a portion of the TCCB in absolute storage.

A list of one or more TIDAWs is called a transport-indirect-data-addressing list (TIDAL).

Unless otherwise specified, TIDAWs may designate a block of main storage on any boundary and length up to 4K bytes, provided the specified block does not cross a 4K-byte boundary.

A TIDAW begins on a quadword boundary and has the following format:

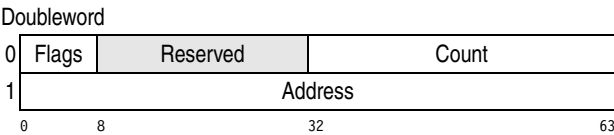


Figure 15-26. Transport-Indirect-Data-Address Word (TIDAW)

Flags : Bits 0-7 of word 0 contain an 8-bit flag field. The meaning of each flag bit is as follows. Bits not described below are reserved.

Bit	Meaning
0	Last TIDAW: Bit 0, when one, specifies that the TIDAW is the last in the contiguous list of TIDAWs.
1	Skip: Bit 1, when one, specifies the suppression of transfer of information to main storage during an input operation, thus specifying that skipping is in effect. When the operation is an output operation, bit 1 is ignored and skipping is not in effect. When the TIDAW is being used

to transport a TCCB, bit 1 is ignored and assumed to be zero.

When skipping is in effect, no checking for invalid or protected data addresses takes place.

2 Data-transfer-interruption control: Bit 2, when one, specifies that a program-check be recognized when the TIDAW is the current TIDAW.

3 TIDAW-transfer-in-channel (TTIC): Bit 3, when one, specifies that the TIDAW is not used to transfer data or TCCB storage. Instead, the address field designates the address of the next TIDAW to be used for the transfer of data. If bit 3 is one and the address field designates a TIDAW in which bit 3 is one, a program-check condition is recognized. If bit 3 is one and any other TIDAW flag bits are one, a program-check condition is recognized.

4 Insert-CBC control: Bit 4, when one, specifies that the following occurs when an output operation is specified and all of the data specified by the TIDAW has been transferred:

- If the count of data bytes transferred is not a multiple of 4, up to 3 padding bytes are transferred to make the total count of data bytes, plus the count of padding bytes, a multiple of 4.
- A word of CBC information is generated and transferred to the device.

Bit 4 has no meaning for the following cases:

- When the TIDAW is being used for an input data transfer.
- When the TIDAW is being used for an output data transfer and the last-TIDAW flag, bit 0, is one.
- When the TIDAW is in a TIDAW list designated by the TCCB-address field of the TCW.

5-7 Reserved: Bits 5-7 are reserved and must be zero, otherwise a program-check condition is recognized.

Reserved: Bits 8-31 are reserved for future use and must be zero; otherwise, a program-check condition may be recognized.

Count : Bits 32-63 specify the number of bytes to be transferred. When skipping is in effect, the count value may be in the range of 1 through $2^{32}-8$ for a TIDAW. When skipping is not in effect and the TTIC flag, bit 3, is zero, the count value, in conjunction with the data address, must not specify the transfer of data that crosses a 4K-byte block boundary; otherwise, a program-check condition is recognized.

When a TIDAW is in the TIDAW list designated by the input-address field in the TCW, the count value specified in the TIDAW in which the last-TIDAW flag is one and the input-count value specified in the TCW must both decrement to zero for the same byte transferred, otherwise a program-check condition is recognized.

When a TIDAW is in the TIDAW list designated by the output-address field in the TCW, the count value specified in the TIDAW in which the last-TIDAW flag is one and the output-count value specified in the TCW must both decrement to zero for the same byte transferred, otherwise a program-check condition is recognized.

When a TIDAW is in the TIDAW list designated by the TCCB-address field in the TCW, the following are true:

- The count value specified in the TIDAW in which the last-TIDAW flag is one and the count of bytes specified by the TCCB-length field in the TCW must both decrement to zero for the same byte transferred, otherwise a program-check condition is recognized.
- If the TIDAW specifies the TCCB to cross a 4K-byte boundary, a program-check condition is recognized.
- If the specified count value is zero or is not evenly divisible by four, a program-check condition is recognized.

When the TTIC flag is zero, the count field must not be zero, otherwise a program-check condition is recognized. When the TTIC flag is one, the count field must be zero, otherwise a program-check condition is recognized.

Address : Bits 64-127 of the TIDAW designate the 64-bit address of a location in absolute storage. When the TTIC flag is zero, this is the location of the first byte of information to be transferred. When the TTIC flag is one, this is the location of the next TIDAW to be used and must specify an address on a quadword boundary, otherwise a program-check condition is recognized.

Input TIDAL: When transport indirect data addressing is specified for input (the input-TIDA flag in the TCW is one), the input-address field of the TCW designates the location of the first TIDAW to be used for data transfer for the first DCW in the associated TCCB that specifies a data transfer command. Additional TIDAWs, if needed for completing the data transfer for the DCW or subsequent DCWs, are fetched from successive locations in storage. The number of TIDAWs required for an input TIDAL is variable and is a function of the following factors:

- The total number of bytes being transferred as specified by the input-count field in the TCW.
- The number of noncontiguous fragments that compose the data being transferred.
- The number of 4K-byte boundaries within those fragments.

The input TIDAL must not cross a 4K-byte boundary, otherwise a program-check condition is recognized.

If a list of two or more TIDAWs is designated for an input operation, it is unpredictable in which order the TIDAW data addresses are used to transfer information into main storage. Thus, if two or more TIDAWs in a TIDAW list designate overlapping storage areas, the results are unpredictable.

Output TIDAL: When transport indirect data addressing is specified for output (the output-TIDA flag in the TCW is one), the output-address field of the TCW designates the location of the first TIDAW to be used for data transfer for the first DCW in the associated TCCB that specifies a data transfer command. Additional TIDAWs, if needed for completing the data transfer for the DCW or subsequent DCWs, are fetched from successive locations in storage. The number of TIDAWs required for a TCW is determined by the output-count field of the TCW. The number of TIDAWs required for an output TIDAL is variable and is a function of the following factors:

- The total number of bytes being transferred as specified by the output-count field in the TCW.
- The number of noncontiguous fragments that compose the data being transferred.
- The number of 4K-byte boundaries within those fragments.

The output TIDAL must not cross a 4K-byte boundary, otherwise a program-check condition is recognized.

TCCB TIDAL: When transport indirect data addressing is specified for the TCCB (the transport-command-control-block-TIDA flag in the TCW is one), the transport-command-control-block-address field of the TCW designates the location of the first TIDAW to be used to designate the storage location of the TCCB. Additional TIDAWs, if needed for completing the TCCB, are fetched from successive locations in storage. The number of TIDAWs required for a TCCB TIDAL is variable and is a function of the following factors:

- The total number of bytes being transferred as specified by the TCCB-length field in the TCW.
- The number of noncontiguous fragments that compose the data being transferred.
- The number of 4K-byte boundaries within those fragments.

The TCCB TIDAL must not cross a 4K-byte boundary, otherwise a program-check condition is recognized.

Additional Information

TIDAWs that specify the transfer of data (TIDAWs for which skipping is not in effect) may specify a transfer count in the range of 1 through 4096 bytes, provided the TIDAW does not specify a data transfer that crosses a 4K-byte boundary.

TIDAWs that specify the suppression of the transfer of data may specify a skip count in the range of 1 through $2^{32}-8$.

The total number of bytes that can be transferred or skipped or both by a single TIDAL is limited by the TCW count field that applies to the TIDAL, which may be a maximum of $2^{32}-8$ bytes. If a TIDAL does not specify the same number of data bytes to be trans-

ferred as the count in the TCW for the TIDAL, or does not specify the same number of data bytes to transferred as that specified by the associated TCCB, a program-check condition may be recognized.

A TIDAW can designate any location. When the TIDAW skip flag is zero and a DCW specifies the read, write, control, sense ID, or sense command, data is then transferred to or from successively higher storage locations until the number of bytes specified by the TIDAW count field have been transferred. When the TIDAW skip flag is one and a DCW specifies the read, sense ID, or sense command, skipping occurs to successively higher storage locations until the number of bytes specified by the TIDAW count field have been skipped.

When the specified number of bytes have been transferred or skipped and the TCW and DCW specify a subsequent TIDAW, the control of data transfer is then passed to the next TIDAW in the list. Like the TIDAW designated by the TCW, subsequent TIDAWs may designate any location and any length, provided the TIDA does not specify a data transfer that crosses a 4K-byte boundary; otherwise, a program-check condition is recognized.

Except for the unique restriction on designating a valid combination of data address and count by a TIDAW, and the resulting behavior when the data-transfer-interruption control is one, all other actions taken for data addresses specified by a TIDAW, such as for protected storage and invalid addresses, and the actions taken for data prefetching, are the same as when transport indirect data addressing is not used.

TIDAWs pertaining to the current TCW may be prefetched. The number of TIDAWs that can be prefetched cannot exceed that required to satisfy the associated count in the TCW that designates the TIDAWs. Any TIDAWs that are prefetched for a TCW and are not used are discarded.

The action of transferring control from one TIDAW to the next is transparent to any attached device. A TIDAW takes control of data transfer when the last byte specified by the previous TIDAW has been transferred, this may occur even in situations where (1) channel end, (2) channel end and device end, or (3) channel end, device end, and status modifier are received prior to the transfer of any data bytes pertaining to the new TIDAW.

Program or access errors when fetching or prefetching TIDAWs are indicated to the program when detected. For output operations, this may preclude transporting the TCCB to the I/O device.

When the channel subsystem detects an invalid CBC on the contents of a prefetched TIDAW or its associated key, the condition may be indicated to the program, when detected, before the TIDAW takes control of data transfer. For a description of the indications provided when an invalid CBC is detected on the contents of a TIDAW or its associated key, see "Channel-Control Check" on page 16-29.

When a TCW specifies a TIDAL and either the address in the TCW designates a storage location that is not physically available or the address designates a storage location to which access is prohibited by protection, then, depending on the model, one of the following two actions is taken:

- The above conditions are checked before initiating the operation at the device. If any of these conditions is recognized, initiation of the I/O operation does not occur, and the subchannel is made status pending with primary, secondary, and alert status.
- The operation is initiated at the device prior to checking for these conditions. If the device attempts to transfer data, the device is signaled to terminate the I/O operation, and the subchannel is made status pending with primary, secondary, and alert status as a function of the subchannel state and any status presented by the device.

Suspension of CCW Channel-Program Execution

The suspend function, when used in conjunction with RESUME SUBCHANNEL, provides the program with a means to stop and restart the execution of a channel program. The initiation of the suspend function is controlled by the setting of the suspend control, bit 4 of word 1 of the command-mode ORB. The suspend function is signaled when suspend control has been specified for the subchannel in the ORB and a CCW containing an S flag set to one becomes the current CCW. The flag can be indicated either in the first CCW of the channel program or in a CCW fetched while command chaining. The S flag is not valid and causes a program-check condition to be recognized

if (1) the ORB contains the suspend-control bit set to zero, or (2) the CCW is fetched while data chaining (see “Data Chaining” on page 15-56, concerning the handling of programming errors detected during data chaining).

Upon recognition of the suspend function, suspension of channel-program execution occurs when the CCW becomes current (see “Channel-Command Word” on page 15-31, for a definition of when a CCW becomes current). If suspension occurs during command chaining, the device is signaled that command chaining is no longer in effect.

RESUME SUBCHANNEL signals that the CCW that caused channel-program suspension may have been modified, that the CCW must be refetched, and that the contents of the CCW must be examined to determine the settings of the flags. If the S flag is one, execution of that CCW does not occur. If the CCW is valid and the S flag in the CCW is zero, execution is initiated (see “RESUME SUBCHANNEL” on page 14-10 and “Start Function and Resume Function” on page 15-20).

When a valid CCW that contains an S flag validly set to one becomes the current CCW during command chaining and the resume-pending condition is not recognized, the suspend function is performed and causes the following actions to occur in the order given:

1. The device is signaled that the chain of operations has been concluded.
2. Channel-program execution is suspended at the subchannel; all prefetched IDAWs, MIDAWs, CCWs, and data are discarded; and the subchannel is set up such that the resume function can be performed when the subchannel is next recognized to be resume pending.
3. If the measurement-block-update mode is active and the subchannel is enabled for the mode, the accrued values of the measurement data, including the start-subchannel and sample count, are added to the accumulated values in the measurement block for the subchannel. The start-subchannel count is the only measurement data that is updated in the measurement block if the channel-subsystem-timing facility is not available for the subchannel. (See “Channel-Subsystem Monitoring” on page 17-1 for more information.)

If a measurement-check condition is detected during the measurement-block update, the channel program is terminated at the subchannel. The subchannel is made status pending with primary, secondary, and alert status, the device-status and subchannel-status fields are set to zero, and one of the measurement-check conditions is indicated in the extended-status flags of the format-0 ESW. The subchannel is not placed in the suspended state. (See “Subchannel-Control Field” on page 16-12.)

4. The subchannel is placed in the suspended state.
5. If the subchannel is not resume pending at this point, the intermediate interruption condition due to suspension is recognized if the suppress-suspended-interruption bit of the ORB is zero; otherwise, the resume function is performed.

When a valid CCW that contains an S flag validly set to one becomes the current CCW during command chaining and the resume-pending condition is recognized, the resume function is performed instead of the suspend function.

When the first CCW of a channel program contains an S flag validly set to one and the resume-pending condition is not recognized, the suspend function is performed and causes the following actions to occur in the order given:

1. Channel-program execution is suspended prior to the selection of the device.
2. The subchannel is set up such that the resume function can be performed when the subchannel is next recognized to be resume pending.
3. If the measurement-block-update mode is active and the subchannel is enabled for the mode, the SSCH+RSCH count is incremented, and the accrued function-pending time (a function of the setting of the timing-facility bit) is added to the accumulated value in the measurement block for the subchannel.

If a measurement-check condition is detected during the measurement-block update, the channel program is not started at the subchannel. The subchannel is made status pending with primary, secondary, and alert status. Deferred condition code one is set, and the start-pending bit remains set to one. The device-status and sub-

channel-status fields are set to zero, and one of the measurement-check conditions is indicated in the extended-status flags of the format-0 ESW. The subchannel is not placed in the suspended state. (See “Subchannel-Control Field” on page 16-12.)

4. The subchannel is placed in the suspended state.
5. If the subchannel is not resume pending at this point, the subchannel is made status pending with intermediate status due to suspension if the suppress-suspended-interruption-control bit of the ORB is zero; otherwise, the resume function is performed.

When the first CCW of a channel program contains an S flag validly set to one and the resume-pending condition is recognized, the resume function is performed instead of the suspend function.

Programming Notes:

1. The execution of MODIFY SUBCHANNEL and START SUBCHANNEL completes with condition code 2 set if the designated subchannel is suspended. The start function is indicated at the subchannel while the subchannel is in the suspended state.
2. In certain situations, normal resumption of the execution of a channel program that has been suspended may not be desired. Normal termination of the suspended channel-program execution may be accomplished by:
 - a. Executing HALT SUBCHANNEL and designating the subchannel.
 - b. Modifying the CCWs in storage such that, when channel-program execution is resumed, the command transferred to the device is a control command with all modifier bits specified as zeros (no-operation) and with the chain-command flag specified as zero; and then executing RESUME SUBCHANNEL.
 - c. When an IRB indicates measurement check along with zero device status, zero subchannel status, and status pending with primary, secondary, and alert status, it may indicate that the measurement check was detected during an attempt to place the subchannel into the suspended state.

3. If the suspended interruption is suppressed, the N condition and DCTI values applicable to the preceding subchannel-active period are not made available to the program. The execution of RESUME SUBCHANNEL when the subchannel is in the suspended state causes path-not-operational conditions and the N condition to be reset to zeros. Path-not-operational conditions and the N condition are not reset when RESUME SUBCHANNEL is executed and the designated subchannel is not in the suspended state.

Commands and Flags for CCWs

Figure 15-27 on page 15-70 lists the command codes for the seven CCW commands and indicates which flags are defined for each command. Except for a format-1 CCW specifying transfer in channel, the flags are ignored for all commands for which they are not defined. The flags are reserved in a format-1 CCW specifying transfer in channel and must be zeros.

Name	Code	Flags
Write	M M M M M M 0 1	CD CC SLI PCI IDA MIDA
Read	M M M M M M 1 0	CD CC SLI SK PCI IDA MIDA
Read backward	M M M M 1 1 0 0	CD CC SLI SK PCI IDA MIDA
Control	M M M M M M 1 1	CD CC SLI PCI IDA MIDA
Sense	M M M M 0 1 0 0	CD CC SLI SK PCI IDA MIDA
Sense ID	1 1 1 0 0 1 0 0	CD CC SLI SK PCI IDA MIDA
Transfer in Channel	X X X X 1 0 0 0	(See note below)

Explanation:

CC Chain command

CD Chain data

IDA Indirect data addressing

M Modifier bit

MIDA Modified indirect data addressing

PCI Program-controlled interruption

S Suspend

SK Skip

SLI Suppress-length indication

X Ignored in a format-0 CCW; must be zero in a format-1 CCW

Note: Flags are ignored in a format-0 transfer-in-channel CCW and must be zeros in a format-1 transfer-in-channel CCW.

Figure 15-27. Command Codes and Flags for CCWs

All flags have individual significance, except for the following cases:

- The CC and SLI flags are ignored when the CD flag is set to one, and, for output forward operations the SK flag is ignored.
- The presence of the SLI flag is ignored for immediate operations involving format-0 CCWs, in which case the incorrect-length indication is suppressed regardless of the setting of the flag.
- The incorrect-length indication may be suppressed for immediate operations when executing a format-1 CCW, depending on the incorrect-length-suppression mode.
- The PCI flag is ignored during initial program loading. All flags, except the PCI flag, are ignored when the S flag is one.
- The MIDA flag is mutually exclusive from both the IDA and SKP flags. If the IDA or SKP flag is specified and the MIDA flag is specified, a program-check condition is recognized.

Programming Notes:

1. A malfunction that affects the validity of data transferred in an I/O operation is signaled at the end of the operation by means of unit check or channel-data check, depending on whether the device (control unit) or the channel subsystem detected the error. In order to make use of the checking facilities provided in the system, data read in an input operation should not be used until the end of the operation has been reached and the validity of the data has been checked. Similarly, on writing, the copy of data in main storage should not be destroyed until the program has verified that no malfunction affecting the transfer and recording of data was detected.
2. An error condition may be recognized and the I/O operation terminated when 256 or more chained commands are executed with a device and none of the executed commands result in the transfer

of any data. When this condition is recognized, program check is indicated.

3. All CCWs that require suppression of incorrect-length indications must use the SLI flag.

Commands and Flags for DCWs

Figure 15-28 on page 15-71 lists the command codes for the six TCW commands and indicates which flags are defined for each command.

Name	Code	DCW Flags
Write	m m m m m m 0 1	CC
Read	m m m m m m 1 0	CC
Control	m m m m m m 1 1	CC
Sense	m m m m 0 1 0 0	CC
Sense ID	1 1 1 0 0 1 0 0	CC
Interrogate	0 1 0 0 0 0 0 0	
Transport COB	0 1 1 0 0 0 0 0	CC
Explanation:		
CC	Chain command	
m	Modifier bit.	

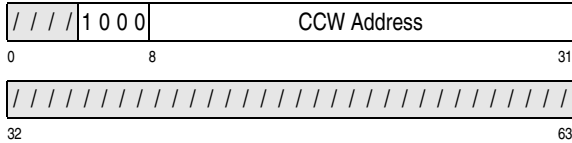
Figure 15-28. Command Codes and Flags for DCWs

Branching in CCW Channel Programs

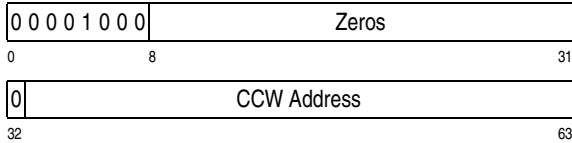
The channel subsystem provides two methods to modify the normal sequential execution of the CCWs in a channel program. One is the transfer-in-channel (TIC) command (described in "Transfer in Channel" on page 15-72), which can be used to loop back to a previously executed CCW, or to connect discontinuous segments of the channel program. The other method, which uses the status-modifier device-status bit (described in the publication *ESA/390 Common I/O-Device Commands*, SA22-7204), allows conditions at the device to cause the channel to bypass the next CCW in the channel program.

Transfer in Channel

Format-0 TIC CCW



Format-1 TIC CCW



The next CCW is fetched from the location in absolute main storage designated by the data-address field of the CCW specifying transfer in channel. The transfer-in-channel command does not initiate any I/O operation, and the I/O device is not signaled of the execution of the command. The purpose of the transfer-in-channel command is to provide chaining between CCWs not located in adjacent doubleword locations in an ascending order of addresses. The command can occur in both data and command chaining.

Bits 29-31 (format 0) or bits 61-63 (format 1) of a CCW that specifies the transfer-in-channel command must be zeros, designating a CCW on a doubleword boundary. Furthermore, a CCW specifying transfer in channel may not be fetched from a location designated by an immediately preceding transfer in channel. When either of these errors is detected or when an invalid address is designated in the transfer-in-channel command, the program-check condition is generated. When a CCW that specifies the transfer-in-channel command designates a CCW at a location protected against fetching, the protection-check condition is generated. Detection of these errors during data chaining causes the operation at the I/O device to be terminated and an interruption condition to be generated, while during command chaining it causes only an interruption condition to be generated.

The contents of the second half of the format-0 CCW, bit positions 32-63, are ignored. Similarly, the contents of bit positions 0-3 of the format-0 CCW are ignored.

Bit positions 0-3 and 8-32 of the format-1 CCW must contain zeros; otherwise, a program-check condition is generated.

Command Retry

The channel subsystem has the capability to perform command retry, a procedure that causes a command to be retried without requiring an I/O interruption. This retry is initiated by the control unit presenting either of two status-bit combinations by means of a special sequence. When immediate retry can be performed, it presents a channel-end, unit-check, and status-modifier status-bit combination, together with device end. When immediate retry cannot be performed, the presentation of device end is delayed until the control unit is prepared. When device end is presented alone, the previous command is transferred again. If device end is accompanied by status modifier, command retry is not performed, and the channel subsystem command-chains to the CCW following the one for which command retry was signaled (for information on status modifier, see the publication *ESA/390 Common I/O-Device Commands*, SA22-7204). When the channel subsystem is not capable of performing command retry due to an error condition, or when any status bit other than device end or device end and status modifier accompanies the requested command-retry initiation, the retry is suppressed, and the subchannel becomes status pending. The SCSW stored by TEST SUBCHANNEL contains the status provided by the I/O device.

Programming Note: The following possible results of a command retry must be anticipated by the program:

1. A CCW containing a PCI may, if retried because of command retry, cause multiple PCI interruptions to occur.
2. If a CCW used in an operation is changed before that operation has been successfully completed, the results are unpredictable.

Concluding I/O Operations before Initiation

Subsequent to the execution of START SUBCHANNEL or RESUME SUBCHANNEL for a command-mode operation and before the first command is accepted, the start function can be ended at the subchannel by CANCEL SUBCHANNEL, CLEAR SUBCHANNEL, or HALT SUBCHANNEL.

Subsequent to the execution of START SUBCHANNEL for a transport-mode operation, the start function may be ended at the subchannel by CANCEL SUBCHANNEL, CLEAR SUBCHANNEL, or HALT SUBCHANNEL before the TCCB is transported.

For both command-mode and transport-mode operations, if the I/O operation is ended by CANCEL SUBCHANNEL, there is no subsequent interruption condition from the I/O operation, and the subchannel is available for the initiation of another start function. However, the device may have signaled a busy condition while the canceled operation was start pending. In this case, the device owes a no-longer-busy signal to the channel subsystem. This may result in unsolicited device-end status before the next operation is initiated at the device. (See also “Clear Function” on page 15-14 and “Halt Function” on page 15-16.)

Concluding I/O Operations during Initiation

After the designated subchannel has been determined to be in a state such that START SUBCHANNEL can be executed, certain tests are performed on the validity of the information specified by the program and on the logical availability of the associated device. This testing occurs during or subsequent to the execution of START SUBCHANNEL and during command chaining and command retry.

A data-transfer operation is initiated at the subchannel and device only when no programming or equipment errors are detected by the channel subsystem and when the device responds with zero status during the initiation sequence. When the channel subsystem detects or the device signals any unusual condition during the initiation of a command-mode I/O operation, the command is said to be not accepted. When the channel subsystem detects or the device signals any unusual condition during the initiation of a transport-mode I/O operation, the I/O-operation initiation is said to be not accepted. For these cases, the subchannel becomes status pending with primary, secondary, and alert status. Deferred condition code 1 is set, and the start-pending bit remains set to one.

Conditions that preclude the initiation of an I/O operation are detailed in the SCSW stored by TEST SUB-

CHANNEL. In this situation, the device is not started, no interruption conditions are generated subsequent to TEST SUBCHANNEL, and the subchannel is idle. The device is immediately available for the initiation of another operation, provided the command was not rejected because of the busy or not-operational condition.

When an unusual condition causes a command to be not accepted during the initiation of a command-mode I/O operation by command chaining or command retry, an interruption condition is generated, and the subchannel becomes status pending with combinations of primary, secondary, and alert status as a function of the status signaled by the device. The status describing the condition remains at the subchannel until cleared by TEST SUBCHANNEL. The conditions are indicated to the program by means of the corresponding status bits in the SCSW. A path-not-operational condition recognized during command chaining is signaled to the program by means of an interface-control-check indication. The new I/O operation at the device is not started.

START SUBCHANNEL is executed independent of its associated device. Tests on most program-specified information, on device availability and unit status, and on most error conditions are performed subsequent to the execution of START SUBCHANNEL. When any conditions are detected that preclude the performance of the start function, an interruption condition is generated by the channel subsystem and placed at the subchannel, causing it to become status pending.

Immediate Conclusion of Command-Mode I/O Operations

During the initiation of an I/O operation, the device can accept the command and signal the channel-end condition immediately upon receipt of the command code. An I/O operation causing the channel-end condition to be signaled during the initiation sequence is called an *immediate operation*. Status generated by the device for the immediate command, when command chaining is not specified and command retry is not signaled, causes the subchannel to become status pending with combinations of primary, secondary, intermediate, and alert status as a result of information specified in the ORB and CCW and status presented by the device. If the immediate operation is

the first operation of the channel program, deferred condition code 1 is set and accompanies the status indications. If intermediate status is indicated, the indication can occur only as a result of the CCW having the PCI flag set to one (see “Program-Controlled Interruption” on page 15-59).

Whenever command chaining is specified after an immediate operation and no unusual conditions have been detected during the operation, or when command retry occurs for an immediate operation, an interruption condition is not generated. The subsequent commands in the chain are handled normally, and, usually, the channel-end condition for the last CCW generates a primary interruption condition. If device end is signaled with channel end, a secondary interruption condition is also generated.

Whenever immediate completion of an I/O operation is signaled, no data has been transferred to or from the device, and the data address in the CCW is not checked for validity. If the subchannel is in the incorrect-length-suppression mode, incorrect length is not indicated to the program, and command chaining is performed when specified. If the subchannel is in the incorrect-length-indication mode, incorrect length and command chaining are under control of the SLI and chain-command flags. The conditions that cause the incorrect-length indication to be suppressed are summarized in Figure 15-24 on page 15-56.

Programming Note: I/O operations for which the entire operation is specified in the command code may be performed as immediate operations. Whether the command is executed as an immediate operation depends on the operation and type of device.

Concluding I/O Operations During Data Transfer

When the subchannel has been passed the contents of an ORB, the subchannel is said to be start pending. When a command-mode I/O operation has been initiated and the command has been accepted, the subchannel becomes subchannel-and-device active. When a transport-mode I/O operation has been initiated, the subchannel remains start pending while the TCCB is transported and executed. The subchannel remains in the respective state unless (1) the channel subsystem detects an equipment malfunction, (2) the operation is concluded by the execution of

CLEAR SUBCHANNEL or HALT SUBCHANNEL, or (3) status that causes a primary interruption condition to be recognized (usually channel end) is accepted from the device.

When CCW command chaining and command retry are not specified or when chaining is suppressed because of unusual conditions, the status that is recognized as primary status causes the operation at the subchannel to be concluded and an interruption condition to be generated. The status bits in the associated SCSW indicate primary status and the unusual conditions, if any. The device can present status that is recognized as primary status at any time after the initiation of the I/O operation, and the presentation of status may occur before any data has been transferred.

For operations not involving data transfer, the device normally controls the timing of the channel-end condition. The duration of data-transfer operations may be variable and may be controlled by the device or the channel subsystem.

Excluding equipment errors and the execution of the CLEAR SUBCHANNEL, HALT SUBCHANNEL, and RESET CHANNEL PATH instructions, the channel subsystem signals the device to conclude the performance of an I/O operation during data transfer whenever any of the following conditions occurs:

- The storage areas designated for the operation are exhausted or filled.
- A program-check condition is detected.
- A protection-check condition is detected.
- A CCW chaining-check condition is detected.
- A channel-control-check condition is detected that does not affect the control of the I/O operation.

The first of these conditions occurs when the channel subsystem has decremented the count to zero in the last CCW or current TCW associated with the operation. A count of zero indicates that the channel subsystem has transferred all information specified by the I/O operation. The other four conditions are due to errors and cause premature conclusion of data transfer. In either case, the conclusion is signaled in response to a service request from the device and causes data transfer to cease. If the device has no blocks defined for the operation (such as writing on

magnetic tape), it concludes the operation and presents channel-end status.

For command-mode operations, the device can control the duration of an operation and the timing of channel end by blocking of data. On certain operations for which blocks are defined (such as reading on magnetic tape), the device does not present channel-end status until the end of the block is reached, regardless of whether the device has been previously signaled to conclude data transfer.

Checking for the validity of the CCW data address is performed only as data is transferred to or from main storage. When the initial data address in the CCW is invalid, no data is transferred during the operation, and the device is signaled to conclude the operation in response to the first service request. On writing, devices such as magnetic-tape units request the first byte of data before any mechanical motion is started, and, if the initial data address is invalid, the operation is terminated by the channel subsystem before the recording medium has been advanced. However, since the operation has been initiated at the device, the device presents channel-end status, causing the channel subsystem to recognize a primary interruption condition. Subsequently, the device also presents device-end status, causing the channel subsystem to recognize a secondary interruption condition. Whether a block at the device is advanced when no data is transferred depends on the type of device.

Checking for the validity of main-storage addresses of the TCCB is performed before attempting to transport the TCCB to the I/O device. When an applicable address is invalid, a program check condition is recognized and no data is transferred during the operation nor is there any communication with the I/O device. After the TCCB has been transported to the device, the operation is considered initiated at the device and subsequent checks for the validity of main-storage addresses are performed either as information is transferred to or from main storage or as information is prefetched, as specified by the applicable TCW reference to storage (directly or indirectly by a TIDAW). If a main-storage address is found to be invalid after the operation has been initiated at the device, the device presents channel-end status, causing the channel subsystem to recognize a primary interruption condition. Subsequently, the device may also present device-end status, causing the channel subsystem to recognize a secondary interruption condition. Whether a block at the device

is advanced when no data is transferred depends on the type of device.

When CCW command chaining takes place, the subchannel is in the subchannel-and-device-active state from the time the first I/O operation is initiated at the device until the device presents channel-end status for the last I/O operation of the chain. The subchannel remains in the device-active state until the device presents the device-end status for the last I/O operation of the chain.

Any unusual conditions cause CCW command chaining to be suppressed and a primary interruption condition to be generated. The unusual conditions can be detected by either the channel subsystem or the device, and the device can provide the indications with channel end, control unit end, or device end. When the channel subsystem is aware of the unusual condition by the time the channel-end status for the operation is accepted, the chain is ended as if the operation during which the condition occurred were the last operation of the chain. The device-end status is recognized as a secondary interruption condition whether presented together with the channel-end status or separately. If the device presents unit check or unit exception together with either control unit end or device end as status that causes the channel subsystem to recognize the primary interruption condition, then the subchannel-and-device-active state of the subchannel is terminated, and the subchannel is made status pending with primary, secondary, and alert status. Intermediate status may also be indicated if an intermediate interruption condition previously existed at the subchannel for the initial-status-interruption condition or the PCI condition and that condition still remains pending at the subchannel. The channel-end status that was presented to the channel subsystem previously when command chaining was signaled is not made available to the program.

Channel-Path-Reset Function

Subsequent to the execution of RESET CHANNEL PATH, the channel-path-reset function is performed. The performance of the function consists of: (1) issuing the reset signal on the designated channel path and (2) causing a channel report to be made pending, indicating the completion of the channel-path-reset function.

Channel-Path-Reset-Function Signaling

The channel subsystem issues the reset signal on the designated channel path. As part of this operation, the following actions are taken:

1. All internal indications associated with control-unit-busy, device-busy, and allegiance conditions for the designated channel path are reset. These indications are reset at all subchannels that have access to the designated channel path. The reset function has no other effect on subchannels, including those having I/O operations in progress.
2. If the channel path fails to respond properly to the reset signal (see “I/O-System Reset” on page 17-13 for a detailed description) or, because of a malfunction, the reset signal could not be issued, the channel path is made physically not available at each applicable subchannel.
3. If an I/O operation is in progress at the device and the device is actively communicating on the channel path in the performance of that I/O operation when the reset signal is received on that channel path, the I/O operation is reset, and the control unit and device immediately terminate current communication with the channel subsystem. (To avoid possible misinterpretation of unsolicited device-end status, programming measures can be taken as described in programming note 2 on page 15-77.)
4. If an I/O operation is in progress in the multipath mode at the device and the device is not currently communicating over the channel path in the performance of that I/O operation when the reset signal is received, then the I/O operation may or may not be reset depending on whether another channel path is available for selection in the same multipath group for the device. If there is at least one other channel path in the multipath group for the device that is available for selection, the I/O operation is not reset. However, the channel path on which the system reset is received is removed from the current set of channel paths that form the multipath group. If the channel path on which the reset signal is received is the only channel path of a multipath group, or if the device is operating in the single-path mode, the I/O operation is reset.

5. The channel-path-reset function causes I/O operations to be terminated at the device as described above; however, I/O operations are *never* terminated at the subchannel by the channel-path-reset function.

If an I/O operation is in progress at the subchannel and the channel path designated for the performance of the channel-path-reset function is being used for that I/O operation, the subchannel may or may not accurately reflect the progress of the I/O operation up to that instant. The subchannel remains in the state that exists at the time the channel-path-reset function is performed until the state is changed because of some action taken by the program or by the device.

Channel-Path-Reset-Function-Completion Signaling

After the reset signal has been issued and an attempt has been made to issue the reset signal, or after it has been determined that the reset signal cannot be issued, the channel-path-reset function is completed. (See “Reset Signal” on page 17-12.)

As a result of the channel-path-reset function being performed, a channel report is made pending (see “Channel-Subsystem Recovery” on page 17-24)

to report the results. If the channel path responds properly to the system-reset signal, the channel report indicates that the channel path has been initialized and is physically available for use. If the reset signal was issued but either the channel path failed to respond properly or the channel path was already not physically available at each subchannel having access to the channel path, the channel report indicates that the channel path has been initialized but is not physically available for use. If, because of a malfunction or because the designated channel path is not in the configuration, the reset signal could not be issued, the channel report indicates that the channel path has not been initialized and is not physically available for use.

Programming Notes:

1. If an I/O operation is in progress in the multipath mode when the channel-path-reset function is performed on a channel path of the multipath group, it is possible for the I/O operation to be

continued on a remaining channel path of the group.

2. When the performance of the channel-path-reset function causes the I/O operation at the device to be reset, unsolicited device-end status presented by the device, if any, may be erroneously interpreted by the channel subsystem to be chaining status and thus cause the channel subsystem to continue the chain of commands. If this situation occurs, then the device-end status is not made available to the program, and the device is selected again by the channel subsystem; however, the device may interpret the initiation sequence as the beginning of a new channel program instead of as command chaining. This pos-

sibility can be avoided by issuing CLEAR SUBCHANNEL or HALT SUBCHANNEL, designating the affected subchannels, prior to issuing RESET CHANNEL PATH.

3. The performance of the channel-path-reset function may, on some models, cause overruns to occur on other channel paths.
4. Even though reset is signaled on the designated channel path, by one or more devices may not have been reset because of a malfunction at a control unit or a malfunction at the physical channel path to the control unit.

Chapter 16. I/O Interruptions

Interruption Conditions	16-2	Count Field	16-31
Intermediate Interruption Condition	16-4	Transport-Mode SCSW	16-34
Primary Interruption Condition	16-4	Subchannel Key	16-35
Secondary Interruption Condition	16-5	Reserved	16-35
Alert Interruption Condition	16-5	Extended-Status-Word Format (L)	16-35
Priority of Interruptions	16-5	Deferred Condition Code (CC)	16-35
Interruption Action	16-6	Format (FMT)	16-35
Interruption-Response Block	16-6	IRB-Format Control (X)	16-35
IRB Format	16-6	Interrogate Complete (Q)	16-35
Subchannel-Status Word	16-7	Subchannel-Control Field	16-35
Command-Mode SCSW	16-8	Extended Control (E)	16-35
Subchannel Key	16-9	Path Not Operational (N)	16-35
Suspend Control (S)	16-9	Function Control (FC)	16-35
Extended-Status-Word Format (L)	16-9	Activity Control (AC)	16-36
Deferred Condition Code (CC)	16-9	Status Control (SC)	16-36
CCW Format (F)	16-11	TCW Address Field	16-36
Prefetch (P)	16-11	Device-Status Field	16-40
Initial-Status-Interruption Control (I)	16-12	Subchannel-Status Field	16-40
Address-Limit-Checking Control (A)	16-12	Program Check	16-40
IRB-Format Control (X)	16-12	Protection Check	16-42
Suppress-Suspended Interruption (U)	16-12	Channel-Data Check	16-43
Subchannel-Control Field	16-12	Channel-Control Check	16-43
Zero Condition Code (Z)	16-12	Interface-Control Check	16-44
Extended Control (E)	16-12	Channel-Subsystem Retry Failed	16-44
Path Not Operational (N)	16-13	FCX-Status Field	16-44
Function Control (FC)	16-13	Subchannel-Extended-Status Field	16-45
Activity Control (AC)	16-14	Extended-Status Word	16-47
Status Control (SC)	16-17	Extended-Status Format 0	16-47
CCW-Address Field	16-19	Subchannel Logout	16-47
Device-Status Field	16-24	Extended-Report Word	16-51
Subchannel-Status Field	16-24	Failing-Storage Address	16-53
Program-Controlled Interruption	16-24	Extended-Subchannel-Logout Descriptor	
Incorrect Length	16-24	(ESLD)	16-53
Program Check	16-25	Secondary-CCW Address	16-53
Protection Check	16-28	Extended-Status Format 1	16-53
Channel-Data Check	16-28	Extended-Status Format 2	16-54
Channel-Control Check	16-29	Extended-Status Format 3	16-55
Interface-Control Check	16-29	Extended-Control Word	16-56
Chaining Check	16-30	Extended-Measurement Word	16-56

When an I/O operation or sequence of I/O operations initiated by the execution of START SUBCHANNEL is ended, the channel subsystem and the device generate status conditions. The generation of these conditions can be brought to the attention of the program by means of an I/O interruption or by means of the execution of the TEST PENDING INTERRUPTION instruction. (During certain abnormal situations, these conditions can be brought to the attention of

the program by means of a machine-check interruption. See “Channel-Subsystem Recovery” on page 17-24 for details.)

The status conditions, as well as an address and a count indicating the extent of the operation sequence, are presented to the program in the form of a subchannel-status word (SCSW). The SCSW is stored in an interruption-response block (IRB) during

the execution of TEST SUBCHANNEL. When a transport-mode IRB is stored, additional information describing the status of the operation is also stored in the transport-status block.

Normally an I/O operation is being performed until the device signals primary interruption status. Primary interruption status can be signaled during initiation of an I/O operation, or later. An I/O operation can be terminated by the channel subsystem performing a clear or halt function when it detects an equipment malfunction, a program check, a chaining check, a protection check, or an incorrect-length condition, or by performing a clear, halt, or channel-path-reset function as a result of the execution of CLEAR SUBCHANNEL, HALT SUBCHANNEL, or RESET CHANNEL PATH, respectively.

I/O interruptions provide a means for the CPU to change its state in response to conditions that occur at I/O devices or subchannels. These conditions can be caused by the program, by the channel subsystem, or by an external event at the device.

Interruption Conditions

The conditions causing requests for I/O interruptions to be initiated are called I/O-interruption conditions. When an interruption condition is recognized by the channel subsystem, it is indicated at the appropriate subchannel. The subchannel is then said to be status pending. The subchannel becoming status pending causes the channel subsystem to generate an I/O-interruption request. An I/O-interruption request can be brought to the attention of the program only once.

An I/O-interruption request remains pending until it is accepted by a CPU in the configuration, is withdrawn by the channel subsystem, or is cleared by means of the execution of TEST PENDING INTERRUPTION, TEST SUBCHANNEL, or CLEAR SUBCHANNEL, or by means of subsystem reset. When a CPU accepts an interruption request and stores the associated interruption code, the interruption request is cleared. When the pending interruption is cleared by the execution of TEST PENDING INTERRUPTION, the subchannel remains status pending until the associated interruption condition is cleared when TEST SUBCHANNEL or CLEAR SUBCHANNEL is executed or when the subchannel is reset.

An I/O-interruption condition is normally cleared by means of the execution of TEST SUBCHANNEL. If TEST SUBCHANNEL is executed, designating a subchannel that has an I/O-interruption request pending, both the interruption request and the interruption condition at the subchannel are cleared. The interruption request and the interruption condition can also be cleared by CLEAR SUBCHANNEL.

A device-end status condition generated by the I/O device and presented following the conclusion of the last I/O operation of a start function is reset at the subchannel by the channel subsystem without generating an I/O-interruption condition or I/O-interruption request if the subchannel is currently start pending and if the status contains device end either alone or accompanied by control unit end. If any other status bits accompany the device-end status bit, then the channel subsystem generates an I/O-interruption request with deferred condition code 1 indicated.

When an I/O operation is terminated because of an unusual condition detected by the channel subsystem during the command-initiation sequence, status describing the interruption condition is placed at the subchannel, causing it to become status pending. If the unusual condition is detected by the device, the device-status field of the associated SCSW identifies the condition.

When command chaining takes place, the generation of status by the device does not cause an interruption, and the status is not made available to the program.

When the channel subsystem detects any of the following interruption conditions, it initiates a request for an I/O interruption without necessarily communicating with, or having received the status byte from, the device:

- A programming error associated with the contents of the ORB passed to the subchannel by the previous execution of START SUBCHANNEL
- A suspend flag set to one in the first CCW fetched that initiates a CCW channel program execution for either START SUBCHANNEL or RESUME SUBCHANNEL, and suppress suspended interruption not specified in the command-mode ORB.

- A programming error associated with the first CCW, TCW, TIDAW, TCCB fetch, data fetch, IDAW, or MIDAW.

These interruption conditions from the subchannel, except for the suspended condition, can be accompanied by other subchannel-status indications, but the device-status indications are all stored as zeros.

The channel subsystem issues the clear signal to the device when status containing unit check is presented to a subchannel that is disabled or when the device is not associated with any subchannel. However, if the presented status does not contain unit check, the status is accepted by the channel subsystem and discarded without causing the subchannel to become status pending.

An interruption condition caused by the device may be accompanied by multiple device-status conditions. Furthermore, more than one interruption condition associated with the same device can be accepted by the channel subsystem without an intervening I/O interruption. As an example, when the channel-end condition is not cleared at the device by the time device end is generated, both conditions may be cleared at the device concurrently and indicated in the SCSW together. Alternatively, channel-end status may have been previously accepted at the subchannel, and an I/O interruption may have occurred; however, the associated status-pending condition may not have been cleared by TEST SUBCHANNEL by the time device-end status was accepted at the subchannel. In this situation, the device-end status may be merged with the channel-end status without causing an additional I/O interruption. Whether an interruption condition may be merged at the subchannel with other existing interruption conditions depends

upon whether the interruption condition is unsolicited or solicited.

Unsolicited Interruption Condition: An unsolicited interruption condition is any interruption condition that is unrelated to the performance of a clear, halt, resume, interrogate, or start function. An unsolicited interruption condition is identified at the subchannel as alert status. An unsolicited interruption condition can be generated only when the subchannel is not device active.

The subchannel and device status associated with an unsolicited interruption condition is never merged with that of any currently existing interruption condition. If the subchannel is currently status pending, the unsolicited interruption condition is held in abeyance in either the channel subsystem or the device, as appropriate, until the status-pending condition has been cleared. Whenever the subchannel is idle and zero status is presented by the device, the status is discarded.

Solicited Interruption Condition: A solicited interruption condition is any interruption condition generated as a direct consequence of performing or attempting to perform a clear, halt, resume, interrogate, or start function. Solicited interruption conditions include any interruption condition generated while the subchannel is either subchannel-and-device active or device active. When the subchannel is operating in transport mode, solicited interruption conditions can be generated when the subchannel is start pending. The subchannel and device status associated with a solicited interruption condition may be merged at the subchannel with that of another currently existing solicited interruption condition. Figure 16-1 on page 16-4 describes the interruption

condition that results from any combination of bits in the status-control field of the SCSW.

Status-Control Field	Status-Control-Bit Combinations															
Alert	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Primary	0	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0
Secondary	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	0
Intermediate	0	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0
Status pending	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Resulting interruption condition	E	S	S	S	S	S	-	S	S	S	S	S	S	-	S	S
Explanation: - Combination does not occur. E Unsolicited or solicited interruption condition. S Solicited interruption condition. 0 Indicates the bit stored as zero. 1 Indicates the bit stored as one.																

Figure 16-1. Interruption Condition for Status-Control-Bit Combinations

Intermediate Interruption Condition

An intermediate interruption condition is a solicited interruption condition that indicates that an event has occurred for which the program had previously requested notification. An intermediate interruption condition is described by any of, or any combination of the following:

- solicited subchannel status
- the Z bit
- the subchannel-suspended condition
- the Q bit

An intermediate interruption condition can occur only after it has been requested by the program through the use of flags in the ORB, CCW, or a CANCEL SUBCHANNEL for a subchannel operating in transport mode that requests an interrogate operation. Depending on the state of the subchannel, the performance or suspension of the I/O operation continues, unaffected by the setting of the intermediate-status bit.

An intermediate interruption condition can be indicated only together with one of the following indications:

1. Subchannel active
2. Status pending with primary status alone

3. Status pending with primary status together with alert status or secondary status or both
4. Suspended
5. Interrogate-complete (Q-bit)

If only the intermediate-status bit and the status-pending bit of the status-control field are ones during the execution of TEST SUBCHANNEL, the device-status field is zero.

Primary Interruption Condition

A primary interruption condition is a solicited interruption condition that indicates the performance of the start function is completed at the subchannel. A primary interruption condition is described by the SCSW stored as a result of the execution of TEST SUBCHANNEL while the subchannel is status pending with primary status. Once the primary interruption condition is indicated at the subchannel, the channel subsystem is no longer actively participating in the I/O operation by transferring commands or data. When a subchannel is status pending with a primary interruption condition, the execution of any of the following instructions results in the setting of a nonzero condition code: HALT SUBCHANNEL, MODIFY SUBCHANNEL, RESUME SUBCHANNEL, and START SUBCHANNEL. Once the primary interruption condition is cleared by the execution of TEST SUBCHANNEL, the subchannel accepts the START

SUBCHANNEL instruction. (See “START SUBCHANNEL” on page 14-15)

Secondary Interruption Condition

A secondary interruption condition is a solicited interruption condition that normally indicates the completion of an I/O operation at the device. A secondary interruption condition is also generated by the channel subsystem if the start function is terminated because a solicited alert interruption condition is recognized prior to initiating the first I/O operation at the device. A secondary interruption condition is described by the SCSW stored as a result of the execution of TEST SUBCHANNEL while the subchannel is status pending with secondary status. Once the channel subsystem has accepted status from the device that causes a secondary interruption condition to be recognized, the start function is completed at the device.

Alert Interruption Condition

An alert interruption condition is either a solicited interruption condition that indicates the occurrence of an unusual condition in a halt, resume, or start function or an unsolicited interruption condition that describes a condition unrelated to the performance of a halt, resume, or start function. An alert interruption condition is described by the SCSW stored as a result of the execution of TEST SUBCHANNEL while the subchannel is status pending with alert status. An alert interruption condition may be generated by either the channel subsystem or the device. Nonzero alert status is always brought to the attention of the program.

Priority of Interruptions

All requests for an I/O interruption are asynchronous to any activity in any CPU, and interruption requests associated with more than one subchannel can exist at the same time. The priority of interruptions is controlled by two types of mechanisms — one establishes within the channel subsystem the priority among interruption requests from subchannels associated with the same I/O-interruption subclass, and another establishes within a given CPU the priority among requests from subchannels of different I/O-interruption subclasses. The channel subsystem

requests an I/O interruption only after it has established priority among requests from its subchannels. The conditions responsible for the I/O-interruption requests associated with subchannels are preserved at the subchannels until cleared by a CPU's execution of TEST SUBCHANNEL or CLEAR SUBCHANNEL or I/O-system reset is performed.

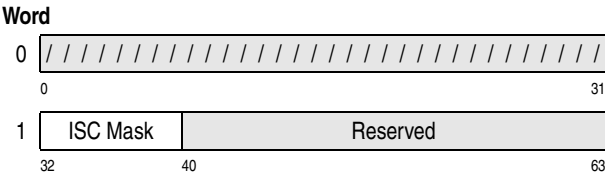
The assignment of priority among requests for interruption from subchannels of the same I/O-interruption subclass is in the order that the need for interruption is recognized by the channel subsystem. The order of recognition by the channel subsystem is a function of the type of interruption condition and the type of channel path. For the parallel-I/O-interface type of channel path, the order depends on the electrical position of the device on the channel path to which it is attached. (A device's electrical position on the parallel-I/O interface is not related to its device address.)

The assignment of priority among requests for interruption from subchannels of different I/O-interruption subclasses is made by the CPU according to the numerical value of the I/O-interruption subclass codes (with zero having highest priority), in conjunction with the I/O-interruption-subclass mask in control register 6. The numerical value of the I/O-interruption-subclass code directly corresponds to the bit position in the I/O-interruption-subclass mask in control register 6 of a CPU. If, in any CPU, an I/O-interruption-subclass-mask bit is zero, then all subchannels having an I/O-interruption-subclass code numerically equal to the associated position in the mask register are said to be masked off in the respective CPU. Therefore, a CPU accepts the highest-priority I/O-interruption request from a subchannel that has the lowest-numbered I/O-interruption subclass code that is not masked off by a corresponding bit in control register 6 of that CPU. When the highest-priority interruption request is accepted by a CPU, it is cleared so that the interruption request is not accepted by any other CPU in the configuration.

The priority of interruption handling can be modified by the execution of either TEST SUBCHANNEL or CLEAR SUBCHANNEL. When either of these instructions is executed and the designated subchannel has an interruption request pending, that interruption request is cleared, without regard to any previous established priority. The relative priority of the remaining interruption requests is unchanged.

Programming Notes:

- 1. The I/O-interruption subclass mask is in control register 6, which has the following format:



- 2. Control register 6 is set to all zeros during initial CPU reset.

Interrupt Action

An I/O interruption can occur only when the I/O-interruption-subclass-mask bit associated with the subchannel is one and the CPU is enabled for I/O interruptions.

The interruption occurs at the completion of a unit of operation (see “Point of Interruption” on page 5-20). If the channel subsystem establishes the priority among requests for interruption from subchannels while the CPU is disabled for I/O interruptions, the interruption occurs immediately after the completion of the instruction enabling the CPU and before the next instruction is executed, provided that the I/O-interruption subclass-mask bit associated with the subchannel is one. Alternatively, if the channel subsystem establishes the priority among requests for interruption from subchannels while the I/O-interruption-subclass-mask bit is zero for each subchannel that is status pending, the interruption occurs immediately after the completion of the instruction that sets at least one of the I/O-interruption-subclass-mask bits to one, provided that the CPU is also enabled for I/O interruptions. This interruption is associated with the highest-priority I/O-interruption request, as established by the CPU.

If the channel subsystem has not established the priority among requests for interruption from the subchannels by the time the interruption is allowed, the interruption does not necessarily occur immediately after the completion of the instruction enabling the CPU. A delay can occur regardless of how long the interruption condition has existed at the subchannel.

The interruption causes the current PSW to be stored as the input/output old PSW at real locations 368-383 and causes the I/O-interruption code associated with the interruption to be stored at real locations 184-195 of the CPU allowing the interruption. Subsequently, a new input/output PSW is loaded from real locations 496-511 and processing resumes in the CPU state indicated by that PSW. The subchannel causing the interruption is identified by the interruption code.

The I/O-interruption code has the following format when it is stored. The code is described in “TEST PENDING INTERRUPTION” on page 14-19.

Hex.

B8	184	Subsystem-Identification Word
BC	188	I/O-Interruption Parameter
C0	192	I/O-Interruption-Identification Word

Programming Note: The I/O-interruption subclass code for all subchannels is set to zero by I/O-system reset. It may be set to any of the values 0-7 by the execution of MODIFY SUBCHANNEL. (The operation of the instruction is described in “MODIFY SUBCHANNEL” on page 14-8.)

Interrupt-Response Block

The interruption-response block (IRB) is the operand of TEST SUBCHANNEL. The two rightmost bits of the IRB address are zeros, designating the IRB on a word boundary. The IRB contains three major fields: the subchannel-status word, the extended-status word, and the extended-control word. When the extended-I/O-measurement-word mode is enabled at the subchannel, the IRB contains a fourth major field, the extended-measurement word.

IRB Format

The format of the IRB stored depends upon the subchannel status. The status present at the subchannel when a TEST SUBCHANNEL instruction is executed determines whether a command-mode IRB or transport-mode IRB is stored. Bit 11 of word 0 of the IRB

is the IRB-format control bit (X). When an IRB is stored with X set to zero, a command-mode IRB is stored. (See “Command-Mode SCSW” on page 16-8.) When an IRB is stored with X set to one, a 3-bit format field is defined. (See “Format (FMT)” on page 16-35.) The format of the IRB is specified according to the table below:

Subchannel Mode	Subchannel Status	Bit (X)	FMT	IRB Mode Stored
Command mode	Any subchannel status or combination of status	0	N	Command mode
Transport mode	Status-pending alone with halt function	1	Y-0	Transport mode
	Status-pending alone with clear function	0	N	Command mode
	Primary status with or without any other status combination	1	Y-0	Transport mode
	Secondary status alone	0	N	Command mode
	Intermediate status	1	Y-0	Transport mode
	Alert status with secondary, without primary or intermediate status	0	N	Command mode
Explanation: N: Format field is not defined. Y-0: Format field is defined and its value is 0.				

Figure 16-2. Summary of IRB Format as Function of Sub-channel Status

The general layout of the IRB is as follows:

Word	
0	Subchannel-Status Word
1	
2	
3	Extended-Status Word
:	
7	
8	Extended-Control Word
15	
16	
:	Extended-Measurement Word
23	
0	31

The length of the subchannel-status and extended-status words is 12 bytes and 20 bytes, respectively. The length of the extended-control word is 32 bytes. When the extended-control bit, bit 14 of word 0 of the SCSW, is zero, words 8-15 of the interruption-response block may or may not be stored. The length of the extended-measurement word is 32 bytes. When the conditions for storing the extended-measurement word are not met (see “Extended-Measurement Word” on page 16-56), words 16-23 of the interruption-response block may or may not be stored.

Subchannel-Status Word

The subchannel-status word (SCSW) provides to the program indications describing the status of a subchannel and its associated device. When a transport-mode IRB is stored, additional information describing the status of the operation and the associated device may also be stored in the transport-status block associated with the operation. If performance of a halt, resume, or start function has occurred, the SCSW may describe the conditions under which the operation was concluded.

The SCSW is stored when TEST SUBCHANNEL is executed and the designated subchannel is operational. The SCSW is placed in words 0-2 of the IRB that is designated as the TEST SUBCHANNEL operand. When STORE SUBCHANNEL is executed, the SCSW is stored in words 7-9 of the subchannel-information block (described in “Subchannel-Information Block” on page 15-2).

Figure 16-3 on page 16-8 shows the summary and contents of the command-mode SCSW.

Command-Mode SCSW

The format of a command-mode SCSW is as follows:

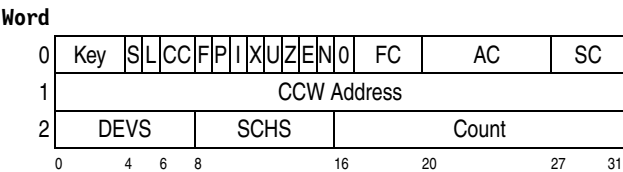


Figure 16-3. Command-Mode SCSW Format

Bits	Name
Word 0	
0-3	Subchannel Key
4	Suspend control (S)
5	ESW format (L)
6-7	Deferred condition code (CC)
8	CCW Format (F)
9	Prefetch (P)
10	Initial-status interruption control (I)
11	IRB-format control (X)
12	Suppressed-suspended interruption (U)
13	Zero condition (Z)
14	Extended control (E)
15	Path not operational (N)
16	Reserved
17-19 Function Control (FC)	
17	Start function
18	Halt function
19	Clear function
20-26 Activity Control (AC)	
20	Resume pending
21	Start pending
22	Halt pending
23	Clear pending
24	Subchannel active
25	Device active
26	Suspended
27-31 Status Control (SC)	
27	Alert status
28	Intermediate status
29	Primary status
30	Secondary status
31	Status pending
Word 1	
0-31	CCW Address
Word 2	
0-7 Device Status (DEVS)	
0	Attention
1	Status modifier
2	Control-unit end
3	Busy
4	Channel end
5	Device end
6	Unit check
7	Unit exception
8-15 Subchannel Status (SCHS)	
8	Program-controlled interruption
9	Incorrect length
10	Program check
11	Protection check
12	Channel-data check
13	Channel-control check
14	Interface-control check
15	Chaining check
16-31 Count	

Figure 16-4. Command-Mode SCSW

The contents of the subchannel-status word (SCSW) depend on the state of the subchannel when the SCSW is stored. Depending on the state of the subchannel and the device, the specific fields of the SCSW may contain (1) information pertaining to the last operation, (2) information unrelated to the performance of an operation, (3) zeros, or (4) meaningless values. The following descriptions indicate when an SCSW field contains meaningful information for a command-mode IRB.

Subchannel Key

When the start-function bit, bit 17 of word 0, is one, bit positions 0-3 of word 0 contain the access key used during performance of the associated start function. These bits are identical with the key specified in bit positions 0-3 of word 1 of the ORB. The subchannel key is meaningful only when the start-function bit, bit 17 of word 0, is one.

Suspend Control (S)

When the start-function bit, bit 17 of word 0, is one, bit 4 of word 0, when one, indicates that the suspend function can be initiated at the subchannel. Bit 4 is meaningful only when bit 17 is one. If bit 17 is one and bit 4 is one, channel-program execution can be suspended if the channel subsystem recognizes an S flag set to one in a CCW. If bit 4 is zero, channel-program execution cannot be suspended, and, if an S flag set to one in a CCW is encountered, a program-check condition is recognized.

Extended-Status-Word Format (L)

When the status-pending bit, bit 31 of word 0, is one, bit 5 of word 0, when one, indicates that a format-0 ESW has been stored. A format-0 ESW is stored when an interruption condition containing any of the following indications is cleared by TEST SUBCHANNEL:

- Channel-data check
- Channel-control check
- Interface-control check
- Measurement-block-program check
- Measurement-block-data check
- Measurement-block-protection check
- Path verification required
- Authorization check
- Extended subchannel logout pending

The extended-status-word-format bit is meaningful whenever the subchannel is status pending. The

extended-status information that is used to form a format-0 ESW is cleared at the subchannel by TEST SUBCHANNEL or CLEAR SUBCHANNEL.

Deferred Condition Code (CC)

When the start-function bit, bit 17 of word 0, is one and the status-pending bit, bit 31 of word 0, is also one, bits 6 and 7 of word 0 indicate the general reason that the subchannel was status pending when TEST SUBCHANNEL or STORE SUBCHANNEL was executed. The deferred condition code is meaningful when the subchannel is status pending with any combination of status and only when the start-function bit of the function-control field in the SCSW is one. The meaning of the deferred condition code for each value when the subchannel is status pending is given in Figure 16-5 on page 16-11.

The deferred condition code, if not zero, is used to indicate whether conditions have been encountered that preclude the subchannel becoming subchannel-and-device active while the subchannel is either start pending or suspended.

Deferred Condition Code 0: A normal I/O interruption has taken place.

Deferred Condition Code 1: Status is present in the SCSW that was presented by the associated device or generated by the channel subsystem subsequent to the setting of condition code 0 for START SUBCHANNEL or RESUME SUBCHANNEL. If only the alert-status bit and the status-pending bit of the status-control field of the SCSW are ones, the status present is not related to the execution of a channel program. If the intermediate-status bit, the primary-status bit, or both are ones, then the status is related to the execution of the channel program specified by the most recently executed START SUBCHANNEL instruction or implied by the most recently executed RESUME SUBCHANNEL instruction. (See "Immediate Conclusion of Command-Mode I/O Operations" on page 15-73.) If the secondary-status bit is one and the primary-status bit is zero, the status present is related to the channel program specified by the START SUBCHANNEL instruction or implied by the RESUME SUBCHANNEL instruction that preceded the most recently executed START SUBCHANNEL instruction.

Deferred Condition Code 2: This code does not occur and is reserved for future use.

Deferred Condition Code 3: An attempted device selection has occurred, and the device appeared not operational on all of the channel paths that were available for selection of the device.

A device appears not operational when it does not respond to a selection attempt by the channel subsystem. This occurs when the control unit is not provided in the system, when power is off in the control unit, or when the control unit has been logically switched off the channel path. The not-operational state is also indicated when the control unit is provided and is capable of attaching the device, but the device has not been installed and the control unit is not designed to recognize the device being selected as one of its attached devices. (See also “I/O Addressing” on page 13-5.)

A deferred condition code 3 also can be set by the channel subsystem if no channel paths to the device are available for selection. (See Figure 16-5 on page 16-11.)

Programming Notes:

1. If, during performance of a start function, the I/O device being selected is not installed or has been logically removed from the control unit, but the associated control unit is operational and the control unit recognizes the I/O device being selected as one of its I/O devices, the control unit, depending upon the model, either fails to recognize the address of the I/O device or considers the I/O device to be not ready. In the former case, a path-not-operational condition is recognized, subject to the setting of the path-operational mask. (See “Path-Operational Mask (POM)” on page 15-7.) In the latter case, the not-ready condition is indicated when the control unit responds to the selection and indicates unit check whenever the not-ready state precludes successful initiation of the operation at the I/O device. In this case, unit-check status is indicated in the SCSW, the subchannel becomes status pending with primary, secondary, and alert status, and with deferred condition code 1 indicated. (See the publication *ESA/390 Common I/O-Device Commands*, SA22-7204, for a description of unit-check status.) Refer to the System Library publication for the control unit to determine how the condition is indicated.
2. The deferred condition code is 1, and the status-control field contains the status-pending and intermediate-status bits or the status-pending, intermediate-status, and alert-status bits as ones when HALT SUBCHANNEL has been executed and the designated subchannel is suspended and status pending with intermediate status. If the alert-status bit is one, then subchannel-logout information was generated as a result of attempting to issue the halt signal to the device.

Bit 6	Bit 7	Status Control ¹	Meaning
0	0	A I P S X A I P – X A – P S X A – P – X – I P S X – I P – X – I – – X – – P S X – – P – X	Normal I/O Interruption
0	1	A I P S X A I P – X A I – – X ² A – P S X A – P – X A – – S X A – – – X – I P S X – I P – X – I – – X ² – – P S X – – P – X – – – S X ³ – – – – X ^{3 2}	Either an immediate operation, with chaining not specified, has ended normally, or the setting of some status condition precluded the initiation or resumption of a requested I/O operation at the device.
1	0	Reserved	Reserved
1	1	– – P S X – I P S X	The device is not operational on any available path or, if a dedicate-allegiance condition exists, the device is not operational on the path to which the dedicated-allegiance is owed.
Explanation: – Bit is zero. ¹ The allowed combinations of status-control-bit settings when the start-function bit is one in the function-control field. ² The condition is encountered after the execution of HALT SUBCHANNEL when the subchannel is currently suspended ³ The condition is encountered after the execution of HALT SUBCHANNEL when the subchannel is currently start pending. A Alert status. I Intermediate status. P Primary status. S Secondary status. X Status pending.			

Figure 16-5. Deferred-Condition-Code Meaning for Status-Pending Subchannel

CCW Format (F)

When the start-function bit, bit 17 of word 0, is one, bit 8 of word 0 indicates the format of the CCWs associated with an I/O operation. The format bit is meaningful only when bit 17 is one. If bit 8 of word 0 is zero, format-0 CCWs are indicated. If it is one, format-1 CCWs are indicated. (See “Channel-Com-

mand Word” on page 15-31 for the description of the two CCW formats.)

Prefetch (P)

When the start-function bit, bit 17 of word 0, is one, bit 9 of word 0 indicates whether or not unlimited prefetching of CCWs, IDAWs, MIDAWS, and associ-

ated data is allowed. The prefetch bit is meaningful only when bit 17 is one. If bit 9 is zero, prefetching of one CCW describing a data area is allowed during output-data-chaining operations and is not allowed during any other operations. If bit 9 is one, unlimited prefetching of CCWs, IDAWs, MIDAWS, and associated data is allowed. It is model dependent whether prefetching is actually performed for any or all of the CCWs, IDAWs, MIDAWS, and associated data that comprise the CCW channel program.

Initial-Status-Interrupt Control (I)

When the start-function bit, bit 17 of word 0, is one, bit 10 of word 0, when one, indicates that the channel subsystem is to generate an intermediate interruption condition if the subchannel becomes subchannel active (see “Initial-Status-Interrupt Control (I)” on page 15-27). Bit 10 of word 0, when zero, indicates that the subchannel becoming subchannel active is not to cause an intermediate interruption condition to be generated.

The program requests the intermediate interruption condition by means of the command-mode ORB. An I/O interruption that results from that request may be due to the channel subsystem performing either a start function or a resume function. (See “Zero Condition Code (Z)” on page 16-12 for details of the indication given by the channel subsystem when the intermediate interruption condition is cleared by TEST SUBCHANNEL.)

Address-Limit-Checking Control (A)

When the address-limit-checking facility is installed and the start-function bit, bit 17 of word 0, is one, bit 11 of word 0, when one, indicates that the channel subsystem has been requested by the program to perform address-limit checking, subject to the setting of the limit mode at the subchannel (see “Address-Limit-Checking Control (A)” on page 15-27). The address-limit-checking-control bit is meaningful only when bit 17 is one.

IRB-Format Control (X)

Bit 11 of word 0 is the IRB-format control bit (X). The value of this bit is zero for the command-mode IRB. See Figure 16-2 on page 16-7.

Suppress-Suspended Interruption (U)

When the start-function bit, bit 17 of word 0, is one, bit 12 of word 0, when one, indicates that the channel subsystem has been requested by the program to

suppress the generation of a subchannel-suspended interruption condition when the subchannel is suspended (see “Suppress-Suspended-Interruption Control (U)” on page 15-27). When bit 12 is zero, the channel subsystem generates an intermediate interruption condition whenever the subchannel is suspended during the execution of the associated channel program. The suppress-suspended-interruption bit is meaningful only when bit 17 is one.

Subchannel-Control Field

The following subchannel-control-information descriptions apply to the subchannel-control field, bits 13-31 of word 0 of the SCSW.

Zero Condition Code (Z)

Bit 13 of word 0, when one, indicates that the subchannel has become subchannel active and the channel subsystem has recognized an initial-status-interruption condition at the subchannel. The Z bit is meaningful only when the intermediate-status bit, bit 28 of word 0, and the start-function bit, bit 17 of word 0, are both ones.

If the initial-status-interruption-control bit, bit 10 of word 1 of the command-mode ORB, is one when START SUBCHANNEL is executed, then the subchannel becoming subchannel active causes the subchannel to be made status pending with intermediate status indicating the initial-status-interruption condition. The initial-status-interruption condition remains at the subchannel until the intermediate interruption condition is cleared by the execution of TEST SUBCHANNEL or CLEAR SUBCHANNEL. If the initial-status-interruption-control bit of the command-mode ORB is zero when START SUBCHANNEL is executed, then the subchannel becoming subchannel active does not cause an intermediate interruption condition to be generated, and the initial-status-interruption condition is not recognized.

Extended Control (E)

Bit 14 of word 0, when one, indicates that model-dependent information or concurrent-sense information is stored in the extended-control word (ECW). When bit 14 is zero, the contents of words 0-7 of the ECW, if stored, are unpredictable. The E bit is meaningful whenever the subchannel is status pending with alert status either alone or together with primary status, secondary status, or both.

Programming Note: During the execution of TEST SUBCHANNEL, the storing of words 0-7 of the ECW is a model-dependent function subject to the setting of bit 14 as described above. Therefore, the program should always provide sufficient storage to accommodate the storing of at least a 64-byte IRB or, when the extended-I/O-measurement-word mode is enabled at the subchannel, a 96-byte IRB.

Path Not Operational (N)

Bit 15 of word 0, when one, indicates that the N condition has been recognized by the channel subsystem. The N condition, in turn, indicates that one or more path-not-operational conditions have been recognized. The channel subsystem recognizes a path-not-operational condition when, during an attempted device selection in order to perform a clear, halt, resume, or start function, the device associated with the subchannel appears not operational on a channel path that is operational for the subchannel. A channel path is operational for the subchannel if the associated device appeared operational on that channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function. A channel path is not operational for the subchannel if the associated device appeared not operational on that channel path the last time the channel subsystem attempted device selection in order to perform a clear, halt, resume, or start function. A device appears to be operational on a channel path when the device responds to an attempted device selection.

The N bit is meaningful whenever the status-control field contains any of the indications listed below and at least one basic I/O function is also indicated at the subchannel:

- Status pending with any combination of primary, secondary, or alert status
- Status pending alone
- Status pending with intermediate status when the subchannel is also suspended

The N condition is reset whenever the execution of TEST SUBCHANNEL results in the setting of condition code 0 and the N bit is meaningful as described above.

Notes:

1. A path-not-operational condition does not imply a malfunctioning channel path. A malfunctioning channel path causes the generation of an error indication, such as interface-control check.
2. When a path-not-operational condition has been recognized and the subchannel subsequently becomes status pending with only intermediate status, the path-not-operational condition (a) continues to be recognized until the subchannel becomes status pending with primary status or becomes suspended and (b) is indicated by storing the path-not-operational bit as a one during the execution of TEST SUBCHANNEL. When a path-not-operational condition has been recognized and the channel-program execution subsequently becomes suspended, the path-not-operational condition does not remain pending if channel-program execution is subsequently resumed. Instead, the old indication is lost, and the path-not-operational indication, if any, pertains to the attempt by the channel subsystem to resume channel-program execution.

Function Control (FC)

The function-control field indicates the basic I/O functions that are indicated at the subchannel. This field may indicate the acceptance of as many as two functions. The function-control field is contained in bit positions 17-19 of the first word of the SCSW. The function-control field is meaningful at an installed subchannel whenever the subchannel is valid (see "Device Number Valid (V)" on page 15-4). The function-control field contains all zeros whenever both the activity- and status-control fields contain all zeros. The meaning of the individual bits is as follows:

Start Function (Bit 17): When one, bit 17 indicates that a start function has been requested and is either pending or in progress at the subchannel. A start function is requested by the execution of START SUBCHANNEL. A start function is indicated at the subchannel when condition code 0 is set during the execution of START SUBCHANNEL. The start-function indication is cleared at the subchannel when TEST SUBCHANNEL is executed and the subchannel is either status pending alone or status pending with any combination of alert, primary, or secondary status. The start-function indication is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

Halt Function (Bit 18): When one, bit 18 indicates that a halt function has been requested and is either pending or in progress at the subchannel. A halt function is requested by the execution of HALT SUBCHANNEL. A halt function is indicated at the subchannel when condition code 0 is set for HALT SUBCHANNEL. The halt-function indication is cleared at the subchannel when the next status-pending condition that occurs is cleared by the execution of TEST SUBCHANNEL. The next status-pending condition depends on the state of the subchannel when HALT SUBCHANNEL is executed. If the subchannel is subchannel active when HALT SUBCHANNEL is executed, then the next status-pending condition is status pending with at least primary status indicated. If the subchannel is device active when HALT SUBCHANNEL is executed, then the next status-pending condition is status pending with at least secondary status indicated. If the subchannel is suspended and status pending with intermediate status when HALT SUBCHANNEL is executed, then the next status-pending condition is status pending with intermediate status. If the subchannel is idle when HALT SUBCHANNEL is executed, then the next status-pending condition is status pending alone. The halt-function indication is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL. In normal operations, this function is indicated together with bit 17; that is, there is a start function either pending or in progress that is to be halted.

Clear Function (Bit 19): When one, bit 19 indicates that a clear function has been requested and is either pending or in progress at the subchannel. A clear function is requested by the execution of CLEAR SUBCHANNEL. A clear function is indicated at the subchannel when condition code 0 is set for CLEAR SUBCHANNEL (see “CLEAR SUBCHANNEL” on page 14-5). The clear-function indication is cleared at the subchannel when the resulting status-pending condition is cleared by TEST SUBCHANNEL.

Activity Control (AC)

The activity-control field is contained in bit positions 20-26 of the first word of the SCSW. This field indicates the current progress of a basic I/O function previously accepted at the subchannel. By using the contents of this field, the program can determine the degree of completion of the basic I/O function. The activity-control field is meaningful at an installed subchannel whenever the subchannel is valid (see

“Device Number Valid (V)” on page 15-4). However, if an IFCC or CCC condition is detected during the performance of a basic I/O function and that function is indicated as pending, I/O operations may or may not have been performed at the device. The activity-control bits are defined as follows:

Bit	Meaning
20	Resume pending
21	Start pending
22	Halt pending
23	Clear pending
24	Subchannel active
25	Device active
26	Suspended

When an SCSW is stored that has the status-pending bit of the status-control field zero and all zeros in the activity-control field, the subchannel is said to be idle or in the idle state.

Note: All conditions that are represented by the bits in the function-control field and by the resume-pending, start-pending, halt-pending, clear-pending, subchannel-active, and suspended bits in the activity-control field are reset at the subchannel when TEST SUBCHANNEL is executed and the subchannel is (1) status pending alone, (2) status pending with primary status, (3) status pending with alert status, or (4) status pending with intermediate status and is also suspended.

Resume-Pending (Bit 20): When one, bit 20 indicates that the subchannel is resume pending. The channel subsystem may or may not be in the process of performing the start function. The subchannel becomes resume pending when condition code 0 is set for RESUME SUBCHANNEL. The point at which the subchannel is no longer resume pending is a function of the subchannel state existing when the resume-pending condition is recognized and the state of the device if channel-program execution is resumed.

If the subchannel is in the suspended state when the resume-pending condition is recognized, the CCW that caused the suspension is refetched, the setting of the suspend flag is examined, and one of the following actions is taken by the channel subsystem:

1. If the CCW suspend flag is one, the device is not selected, the subchannel is no longer resume pending, and the channel-program execution remains suspended.

2. If the CCW suspend flag is zero, the channel subsystem attempts to resume channel-program execution by performing a modified start function. The resumption of channel-program execution appears to the device as the initiation of a new channel-program execution. The resume function causes the channel subsystem to perform the path-management operation as if a new start function were being initiated, using the ORB parameters previously passed to the subchannel by START SUBCHANNEL with the exception that the channel-program address is the address of the CCW that caused the suspension of the channel-program execution.

The subchannel remains resume pending when, during the performance of the start function, the channel subsystem (1) determines that it is not possible to attempt to initiate the I/O operation for the first command, (2) determines that an attempt to initiate the I/O operation for the first command does not result in the command being accepted, or (3) detects an IFCC or CCC condition and is unable to determine whether the first command has been accepted. (See “Start Function and Resume Function” on page 15-20.)

The subchannel is no longer resume pending when any of the following events occurs:

- a. While performing the start function, the subchannel becomes subchannel-and-device active or device active only, or the first command is accepted with channel-end and device-end initial status and the CCW does not specify command chaining.
- b. CLEAR SUBCHANNEL is executed.
- c. TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.
- d. TEST SUBCHANNEL clears intermediate status while the subchannel is suspended.
- e. CANCEL SUBCHANNEL is executed with a resulting condition code 0.

If the subchannel is not in the suspended state when the resume-pending condition is recognized, the CCW suspend flag of the most recently fetched CCW, if any, is examined, and one of the following actions is taken by the channel subsystem:

1. If a CCW has not been fetched or the suspend flag of the most recently fetched CCW is zero, the subchannel is no longer resume pending, and the resume function is not performed.
2. If the suspend flag of the most recently fetched CCW is one, the subchannel is no longer resume pending, and the CCW is refetched. The subchannel proceeds with channel-program execution if the suspend flag of the refetched CCW is zero. The subchannel suspends channel-program execution if the suspend flag of the refetched CCW is one.

Some models recognize a resume-pending condition only after a CCW having an S flag validly set to one is fetched. Therefore, if a subchannel is resume pending and, during the execution of the channel program, no CCW is fetched that has an S flag validly set to one, the subchannel remains resume pending until the primary interruption condition is cleared by TEST SUBCHANNEL.

Start-Pending (Bit 21): When one, bit 21 indicates that the subchannel is start pending. The channel subsystem may or may not be in the process of performing the start function. The subchannel becomes start pending when condition code 0 is set for START SUBCHANNEL. The subchannel remains start pending when, during the performance of the start function, the channel subsystem (1) determines that it is not possible to attempt to initiate the I/O operation for the first command, (2) determines that an attempt to initiate the I/O operation for the first command does not result in the command being accepted, or (3) detects an IFCC or CCC condition and is unable to determine whether the first command has been accepted. (See “Start Function and Resume Function” on page 15-20.)

The subchannel becomes no longer start pending when any of the following occurs:

1. While performing the start function, the subchannel becomes subchannel-and-device active or device active only, or the first command is accepted with channel-end and device-end initial status and the CCW does not specify command chaining.
2. The subchannel becomes suspended because of a suspend flag validly set to one in the first CCW.
3. CLEAR SUBCHANNEL is executed.

4. TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.
5. CANCEL SUBCHANNEL is executed with a resulting condition code 0.

Halt-Pending (Bit 22): When one, bit 22 indicates that the subchannel is halt pending. The channel subsystem may or may not be in the process of performing the halt function. The subchannel becomes halt pending when condition code 0 is set for HALT SUBCHANNEL. The subchannel remains halt pending when, during the performance of the halt function, the channel subsystem (1) determines that it is not possible to attempt to issue the halt signal to the device, (2) determines that the attempt to issue the halt signal to the device is not successful, or (3) detects an IFCC or CCC condition and is unable to determine whether the halt signal is issued to the device. (See “Halt Function” on page 15-16.)

The subchannel is no longer halt pending when any of the following occurs:

1. While performing the halt function, the channel subsystem determines that the halt signal has been issued to the device.
2. CLEAR SUBCHANNEL is executed.
3. TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.
4. TEST SUBCHANNEL clears intermediate status while the subchannel is suspended.

Clear-Pending (Bit 23): When one, bit 23 indicates that the subchannel is clear pending. The channel subsystem may or may not be in the process of performing the clear function. The subchannel becomes clear pending when condition code 0 is set for CLEAR SUBCHANNEL. The subchannel remains clear pending when, during performance of the clear function, the channel subsystem (1) determines that it is not possible to attempt to issue the clear signal to the device, (2) determines that the attempt to issue the clear signal to the device is not successful, or (3) detects an IFCC or CCC condition and is unable to determine whether the clear signal is issued to the device. (See “Clear Function” on page 15-14.)

The subchannel is no longer clear pending when either of the following occurs:

1. While performing the clear function, the channel subsystem determines that the clear signal has been issued to the device.
2. TEST SUBCHANNEL clears the status-pending condition alone.

Subchannel Active (Bit 24): When one, bit 24 indicates that the subchannel is subchannel active. A subchannel is said to be subchannel active when an I/O operation is currently being performed at the subchannel. The subchannel becomes subchannel active when the first command is accepted and the start function or resume function is not immediately concluded at the subchannel. (See “Immediate Conclusion of Command-Mode I/O Operations” on page 15-73.)

The subchannel is no longer subchannel active when any of the following occurs:

1. The subchannel becomes suspended.
2. The subchannel becomes status pending with primary status.
3. CLEAR SUBCHANNEL is executed.
4. The device appears not operational during performance of a halt function.

The subchannel does not become subchannel active during performance of the function specified by either a HALT SUBCHANNEL or a CLEAR SUBCHANNEL instruction.

Device Active (Bit 25): When one, bit 25 indicates that the subchannel is device active. A subchannel is said to be device active when an I/O operation is currently in progress at the associated device. The subchannel becomes device active when the first command is accepted.

The subchannel is no longer device active when any of the following occurs:

1. The subchannel becomes suspended.
2. The subchannel becomes status pending with secondary status.
3. CLEAR SUBCHANNEL is executed.
4. The device appears not operational during performance of a halt function.

If the subchannel is not start pending or if the status accepted from the device also describes an alert condition, the subchannel becomes status pending with secondary status. After the status has been accepted from the device, the device is capable of accepting a command for performing a new I/O operation. If the subchannel is start pending and the status is device end or device end with control unit end, then the channel subsystem discards the status and performs the start function for the new channel program. (See “Start Function and Resume Function” on page 15-20) In this situation, the subchannel does not become status pending with the secondary interruption condition, and the status is not made available to the program.

The subchannel does not become device active during performance of the functions specified by either a HALT SUBCHANNEL or a CLEAR SUBCHANNEL instruction.

Suspended (Bit 26): When one, bit 26 indicates that the subchannel is suspended. A subchannel is said to be suspended when channel-program execution is currently suspended. The subchannel becomes suspended as part of the suspend function. (See “Suspension of CCW Channel-Program Execution” on page 15-68.)

The subchannel is no longer suspended when any of the following occurs:

1. As part of the resume function following the execution of RESUME SUBCHANNEL when the subchannel becomes subchannel-and-device active or device active only, or the first command is accepted for channel-end and device-end initial status, with or without status modifier, and the CCW does not specify command chaining.
2. CLEAR SUBCHANNEL is executed.
3. TEST SUBCHANNEL clears any combination of primary, secondary, and alert status or clears the status-pending condition alone.
4. TEST SUBCHANNEL clears intermediate status while the halt function is specified.
5. CANCEL SUBCHANNEL is executed with a resulting condition code 0.

Programming Note: When an SCSW is stored by STORE SUBCHANNEL or TEST SUBCHANNEL following CLEAR SUBCHANNEL but prior to the sub-

channel becoming status pending, and the subchannel-active bit, bit 24 of word 0, is stored as zero, this does not mean that data transfer has stopped for the device. The program cannot determine whether data transfer has stopped until the subchannel becomes status pending as a result of performing the clear function.

Status Control (SC)

The status-control field is contained in bit positions 27-31 of the first word of the SCSW. This field provides the program with a summary-level indication of the interruption condition described by either subchannel or device status, the Z bit, or, in the case of the subchannel-suspended interruption, the suspended bit, bit 26. More than one summary indication may be signaled as a result of existing conditions at the subchannel. Whenever the subchannel is enabled (see “Enabled (E)” on page 15-3) and at least bit 31 is one, the subchannel is said to be status pending. Whenever the subchannel is disabled, the subchannel is not made status pending. Bit 31 of SCSW word 0 is meaningful at an installed subchannel whenever the subchannel is valid (see “Device Number Valid (V)” on page 15-4); bits 27-30 are meaningful when bit 31 is one. The status-control bits are defined as follows:

Alert Status (Bit 27): When one (and when the status-pending bit is also one), bit 27 indicates an alert interruption condition exists. In such a case, the subchannel is said to be status pending with alert status. An alert interruption condition is recognized when alert status is present at the subchannel. Alert status may be subchannel status or device status. Alert status is status generated by either the channel subsystem or the device under any of the following conditions:

- The subchannel is idle (activity-control bits 20-26 and status-control bit 31 are zeros).
- The subchannel is start pending, and the status condition precludes initiation of the I/O operation.
- The subchannel is subchannel-and-device active, and the status condition has suppressed command chaining or would have suppressed command chaining if chaining had been specified (see “CCW Channel Program Chaining” on page 15-55).
- The subchannel is subchannel-and-device active, command chaining is not specified, the

execution of the channel program has just been concluded, and the status presented by the device is attempting to alter the sequential execution of commands (see the publication *ESA/390 Common I/O-Device Commands*, SA22-7204, for more information on the use of status modifier to alter the sequential execution of commands).

- The subchannel is device active only, and the status presented by the device is other than device end, control unit end, or device end and control unit end.
- The subchannel is suspended (bit 26 is one).

If the subchannel is start pending when an alert interruption condition is recognized, the subchannel becomes status pending with alert status, deferred condition code 1 is set, the start-pending bit remains one, and the performance of the pending I/O operation is not initiated.

When TEST SUBCHANNEL is executed and stores an SCSW with the alert-status bit and the status-pending bit as ones in the IRB, the alert interruption condition is cleared at the subchannel. The alert interruption condition is also cleared during the execution of CLEAR SUBCHANNEL.

Whenever alert status is present at the subchannel, it is brought to the attention of the program. Examples of alert status include attention, device end (which signals a transition from the not-ready to the ready state), incorrect length, program check, and unit check.

Intermediate Status (Bit 28): When one (and when the status-pending bit is also one), bit 28 indicates an intermediate interruption condition exists. In such a case, the subchannel is said to be status pending with intermediate status. Intermediate status can be indicated when the Z bit (of the subchannel-control field), the suspended bit (of the activity-control field), or the PCI bit (of the subchannel-status field) is one.

When the initial-status-interruption-control bit is one in the command-mode ORB, the subchannel becomes status pending with intermediate status (the Z bit indicated) only after the subchannel is subchannel active. If the subchannel does not become subchannel active, the Z condition is not generated.

When suspend control is specified and the generation of an intermediate interruption condition due to suspension is not suppressed in the command-mode ORB, then the subchannel can become status pending with intermediate status due to suspension if a CCW becomes current that contains the suspend flag set to one. When the suspend flag is specified in the first CCW of a channel program, channel-program execution is suspended, and the subchannel becomes status pending with intermediate status (the suspended bit indicated) before the command in the first CCW is transferred to the device. When the suspend flag is specified in a CCW fetched during command chaining, then channel-program execution is suspended, and the subchannel becomes status pending with intermediate status (the suspended bit indicated), only after the execution of the preceding CCW is complete.

When the PCI flag is specified in a CCW, the generation of an intermediate interruption condition due to PCI depends on whether the CCW is the first CCW of the channel program. When the PCI flag is specified in the first CCW of a channel program, the subchannel becomes status pending with intermediate status (the PCI bit indicated) only after initial status is received for the first CCW of the channel program indicating the command has been accepted. When the PCI flag is specified in a CCW fetched while chaining, the subchannel becomes status pending with intermediate status (the PCI bit indicated) only after the execution of the preceding CCW is complete. If chaining occurs before an interruption condition containing PCI is cleared by TEST SUBCHANNEL, the condition is carried over to the next CCW. This carry-over occurs during both data and command chaining, and, in either case, the condition is propagated through the transfer-in-channel command.

If the subchannel is status pending with intermediate status when HALT SUBCHANNEL is executed, the intermediate interruption condition remains at the subchannel, but the interruption request, if any, is withdrawn, and the subchannel becomes no longer status-pending. The subchannel remains no longer status pending until performance of the halt function has ended. The subchannel then becomes status pending with intermediate status indicated (possibly together with any combination of primary, secondary, and alert status).

When TEST SUBCHANNEL is executed and stores an SCSW with the intermediate-status bit and the

status-pending bit as ones in the IRB, the intermediate interruption condition is cleared at the subchannel. The intermediate interruption condition is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

Primary Status (Bit 29): When one (and when the status-pending bit is also one), bit 29 indicates a primary interruption condition exists. In such a case, the subchannel is said to be status pending with primary status. A primary interruption condition is a solicited interruption condition that indicates the completion of the start function at the subchannel. The primary interruption condition is described by the SCSW stored. When an I/O operation is terminated by HALT SUBCHANNEL but the halt signal is not issued to the device because the device appeared not operational, the subchannel is made status pending with primary status (and secondary status) with both the subchannel-status field and the device-status field set to zero.

When TEST SUBCHANNEL is executed and stores an SCSW with the primary-status bit and the status-pending bit as ones in the IRB, the primary interruption condition is cleared at the subchannel. The primary interruption condition is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

Secondary Status (Bit 30): When one (and when the status-pending bit is also one), bit 30 indicates a secondary interruption condition exists. In such a case, the subchannel is said to be status pending with secondary status. A secondary interruption condition is a solicited interruption condition that normally indicates the completion of the I/O operation at the device. The secondary interruption condition is described by the SCSW stored.

When an I/O operation is terminated by HALT SUBCHANNEL but the halt signal is not issued to the device because the device appeared not operational, the subchannel is made status pending with secondary status (and primary status if the subchannel is also subchannel active) with zeros for subchannel and device status.

When TEST SUBCHANNEL is executed and stores an SCSW with the secondary-status bit as one in the IRB, the secondary interruption condition is cleared at the subchannel. The secondary interruption condition is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

Status-Pending (Bit 31): When one, bit 31 indicates that the subchannel is status pending and that information describing the cause of the interruption condition is available to the program. The subchannel becomes status pending whenever intermediate, primary, secondary, or alert status is generated. When HALT SUBCHANNEL is executed, designating a subchannel that is idle, the subchannel becomes status pending subsequent to performance of the halt function to notify the program that the halt function has been completed. When TEST SUBCHANNEL is executed, thus storing an SCSW with the status-pending bit as one in the IRB, the status-pending condition is cleared at the subchannel. The status-pending condition is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL. When CLEAR SUBCHANNEL is executed and the designated subchannel is operational, the subchannel becomes status pending subsequent to performance of the clear function to notify the program that the clear function has been completed.

Note: The status-pending bit, in conjunction with the remaining bits of the status-control field, indicates the type of status condition. For example, if bits 29 and 31 are ones, the subchannel is status pending with primary status. Alternatively, if only bit 31 is one, then the subchannel is said to be status pending or status pending alone. If only bit 31 is one in the status-control field, the settings of all bits in the subchannel-status and device-status fields are unpredictable. If bit 31 is not one, then the remaining bits of the status-control field are not meaningful.

CCW-Address Field

Bits 1-31 of word 1 form an absolute address. The address indicated is a function of the subchannel state when the SCSW is stored, as indicated in Figure 16-6 on page 16-20. When the subchannel-status field indicates channel-control check, channel-data check, or interface-control check, the CCW-address field is usable for recovery purposes if the CCW-address field-validity flag in the ESW is one.

Programming Note: When a CCW address, either detected in the channel-program address (see “Channel-Program Address” on page 15-28) or generated during chaining, would cause the channel subsystem to fetch a CCW from a location greater than $2^{24} - 1$ while format-0 CCWs are specified for the operation, the invalid address is stored in the CCW-address field of the SCSW without truncation.

If the invalid address causes the channel subsystem, while chaining, to fetch a CCW from a location greater than $2^{31} - 1$ while format-1 CCWs are speci-

fied for the operation, the rightmost 31 bits of the invalid address are stored in the CCW-address field.

Subchannel State ¹	CCW Address ²
Start pending (UUUU0/AIP SX) ³	Unpredictable
Start pending and device active (UUUU0/AIP SX) ³	Unpredictable
Subchannel-and-device active (UUUU0/AIP SX) ³	Unpredictable
Device active only (UUUU0/AIP SX)	Unpredictable
Suspended (YYYYY/AIP SX) ³	See note 1
Status pending (10001/AIP SX) because of unsolicited alert status from the device while the subchannel was start pending ³	Channel-program address + 8
Status pending (0Y111/AIP SX) because the device appeared not operational on all paths ³	Channel-program address + 8
Status pending (10011/AIP SX) because of solicited alert status from the device while the subchannel was start pending and device active ³	Channel-program address + 8
Status pending (10111/AIP SX) because of solicited alert status generated by the channel subsystem while the subchannel was start pending ³ or start pending and device active ³	See note 2
Status pending (01001/AIP SX) for the program-controlled-interruption condition while the subchannel was subchannel-and-device active ³	CCW + 8 of the CCW that contained the last recognized PCI, or 8 higher than a CCW that has subsequently become current
Status pending (01001/AIP SX) for the initial-status-interruption condition while the subchannel was subchannel-and-device active ³	CCW + 8 of the CCW causing the intermediate interruption condition, or a CCW that has subsequently become current
Status pending (1Y1Y1/AIP SX); termination occurred because of program check caused by one of the following conditions: ³	
Bit 24 of word 1 of the ORB set to one; incorrect-length-indication-suppression facility not installed	Channel-program address + 8
Unused bits in ORB not set to zeros	Channel-program address + 8
Invalid CCW-address specification in transfer in channel (TIC)	Address of TIC + 8
Invalid CCW-address specification in the channel-program address in the ORB	Channel-program address + 8 ⁴

Figure 16-6. (Part 1 of 4) CCW Address as Function of Subchannel State

Subchannel State ¹	CCW Address ²
Invalid CCW address in TIC	Address of TIC + 8
Invalid CCW address in the channel-program address in the ORB	Channel-program address + 8 ⁴
Invalid CCW address while chaining	Invalid CCW address + 8
Invalid command code	Address of invalid CCW + 8 ⁵
Invalid count	Address of invalid CCW + 8 ⁵
Invalid IDAW-address or MIDAW-address specification	Address of invalid CCW + 8 ⁵
Invalid IDAW or MIDAW address in CCW	Address of invalid CCW + 8 ⁵
Invalid IDAW address while sequentially fetching IDAWs or invalid MIDAW address while sequentially fetching MIDAWs	Address of current CCW + 8
Invalid data-address specification, format 1	Address of invalid CCW + 8 ⁵
Invalid data address in a CCW	Address of invalid CCW + 8 ⁵
Invalid data address while sequentially accessing storage	Address of current CCW + 8
Invalid data address in IDAW or MIDAW	Address of current CCW + 8
Invalid IDAW or MIDAW specification	Address of current CCW + 8
Invalid CCW, format 0 or 1, for a CCW other than a TIC	Address of invalid CCW + 8 ⁵
Invalid suspend flag – CCW fetched during data chaining has suspend flag set to one	Address of invalid CCW + 8
Invalid suspend flag – CCW has suspend flag set to one, but suspend control was not specified in the ORB	Address of invalid CCW + 8
Invalid MIDA flag – CCW has MIDA flag set to one, but modified-CCW-indirect-data addressing was not specified in the ORB	Address of invalid CCW + 8
Invalid MIDA flag – CCW has MIDA flag set to one and either the skip flag or the IDA flag or both are set to one	Address of invalid CCW + 8
Invalid CCW, format 1, for a TIC	Address of TIC + 8
Invalid sequence – two TICs	Address of second TIC + 8
Invalid sequence – 256 or more CCWs without data transfer	Address of 256th CCW + 8
Status pending (1Y1Y1/AIP SX); termination occurred because of protection check detected as follows: ³	
On a CCW access	Address of protected CCW + 8 ⁵
On data or an IDAW or a MIDAW access	Address of current CCW + 8

Figure 16-6. (Part 2 of 4) CCW Address as Function of Subchannel State

Subchannel State ¹	CCW Address ²
Status pending (1Y1Y1/AIP SX); termination occurred because of chaining check ³	Address of current CCW + 8
Status pending (YY1Y1/AIP SX); termination occurred under count control ³	Address of current CCW + 8 ⁶
Status pending (1Y1Y1/AIP SX); operation prematurely terminated by the device because of alert status ³	Address of current CCW + 8 ⁶
Status pending (YYYY1/AIP SX) after termination by HALT SUB-CHANNEL and the activity-control-field bits indicated below set to ones:	
Status pending alone	Unpredictable
Start pending ³	Unpredictable
Device active and start pending ³	Unpredictable
Device active	Unpredictable
Subchannel active and device active ³	CCW + 8 of the last-executed CCW
Suspended	CCW + 8 of CCW causing suspension
Suspended and resume pending	Unpredictable
Status pending (00001/AIP SX) after termination by CLEAR SUB-CHANNEL	Unpredictable
Status pending (YY1Y1/AIP SX); operation completed normally at the subchannel ³	CCW + 8 of the last-executed CCW ⁶
Status pending (00011/AIP SX)	Unpredictable
Status pending (10001/AIP SX)	Unpredictable
Status pending (00001/AIP SX)	Unpredictable
Status pending (1Y111/AIP SX); command chaining suppressed because of alert status other than channel-control check or interface-control check ³	Address of current CCW + 8 ⁶
Status pending (1YYY1/AIP SX) because of alert status for channel-control check or interface-control check ³	See note 3 ⁶
Status pending (1Y1Y1/AIP SX) because of channel-data check ³	See note 3 ⁶

Figure 16-6. (Part 3 of 4) CCW Address as Function of Subchannel State

Explanation:

1	<p>The meaning of the notation used in this column is as follows:</p> <ul style="list-style-type: none">A Alert statusI Intermediate statusP Primary statusS Secondary statusX Status Pending <p>The possible combination of status-control-bit settings is shown to the left of the "/" symbol by the use of these symbols:</p> <ul style="list-style-type: none">0 Corresponding condition is not indicated1 Corresponding condition is indicatedU Unpredictable. The corresponding condition is not meaningful when the subchannel is not status pending.Y The corresponding condition is not significant and is indicated as a function of the subchannel state.
2	<p>A CCW becomes current when (1) it is the first CCW of a channel program and has been fetched, (2) while command chaining, the previous CCW is no longer current and the new CCW has been fetched, or (3) in the case of data chaining, the new CCW takes over control of the I/O operation (see the section "Data Chaining" in Chapter 15, "Basic I/O Functions"). If chaining is not specified or is suppressed, a CCW is no longer current and becomes the last-executed CCW when secondary status has been accepted by the channel subsystem. During command chaining, a CCW is no longer current when device-end status has been accepted or, in the case of data chaining, when the last byte of data for that CCW has been accepted.</p>
3	<p>The subchannel may also be resume pending.</p>
4	<p>The stored address is the channel-program address (in the ORB) + 8 even though it is either invalid or protected.</p>
5	<p>The stored address is the address of the current CCW + 8 even though it is either invalid or protected.</p>
6	<p>Incorrect length is indicated as a function of the setting of the suppress-length-indication flag in the current CCW (see the section "Channel-Command Word" in Chapter 15, "Basic I/O Functions").</p>
Notes:	
1.	<p>Unless the subchannel is also resume pending, the address stored is the address of the CCW that caused suspension, plus 8. Otherwise, the address stored is unpredictable.</p>
2.	<p>The address of the CCW is given as a function of the alert status indicated. For example, if a program-check or protection-check condition is recognized, the CCW address stored is the same as for the entry for program check or protection check, respectively, in this table. Alternatively, if alert status for interface-control check or channel-control check is indicated, the CCW address stored is either the channel-program address (in the ORB) + 8 or invalid as specified by the field-validity flags in the subchannel logout. If alert status for path-verification required is indicated, the CCW address stored is the channel-program address (in the ORB) + 8.</p>
3.	<p>Bit 21 of the subchannel-logout information, when stored as one, indicates that the address is CCW + 8 of the last-fetched CCW if the command for the CCW has not been accepted by the device. If the command has been accepted by the device at the time the error condition is recognized, then the address stored is the address of the CCW + 8 of the last-executed CCW.</p>

Figure 16-6. (Part 4 of 4) CCW Address as Function of Subchannel State

Device-Status Field

Device-status conditions are generated by the I/O device and are presented to the channel subsystem over the channel path. The timing and causes of these conditions for each type of device are specified in the System Library publication for the device. The device-status field is meaningful whenever the subchannel is status pending with any combination of primary, secondary, intermediate, or alert status. Whenever the subchannel is status pending with intermediate status alone, the device-status field is zero. When the subchannel-status field indicates channel-control check, channel-data check, or interface-control check, the device-status field is usable for recovery purposes if the device-status field-validity flag in the ESW is one. When the subchannel is status pending with deferred condition code 3 indicated, the contents of the device-status field are not meaningful.

If, within a system, the I/O device is accessible from more than one channel path, status related to channel-subsystem-initiated operations in the single-path mode (solicited status) is signaled over the initiating channel path. Devices operating in the multipath mode may signal solicited status over any channel path that belongs to the same path group as the initiating channel path. The handling of conditions not associated with I/O operations (unsolicited alert status), such as attention, unit exception, and device end due to transition from the not-ready to the ready state, depends on the type of device and condition and is specified in the System Library publication for the device.

The channel subsystem does not modify the status bits received from the I/O device. These bits appear in the SCSW as received over the channel path. For more information on the status bits received from the I/O device, see the publication *ESA/390 Common I/O-Device Commands*, SA22-7204.

Subchannel-Status Field

Subchannel-status conditions are detected and indicated in the SCSW by the channel subsystem. Except for the conditions caused by equipment malfunctioning, they can occur only while the channel subsystem is involved with the performance of a halt, resume, or start function. The subchannel-status field is meaningful whenever the subchannel is status

pending with any combination of primary, secondary, intermediate, or alert status. Individual bits contained in the subchannel-status field may be unpredictable even when the subchannel-status field is meaningful. When the subchannel is status pending with deferred condition code 3 indicated, the contents of the subchannel-status field are not meaningful.

Program-Controlled Interruption

An intermediate interruption condition is generated after a CCW with the program-controlled-interruption (PCI) flag set to one becomes the current CCW. The I/O interruption due to the PCI flag may be delayed an unpredictable amount of time because of masking of the interruption request or other activity in the system. (See “Program-Controlled Interruption” on page 15-59.) When the channel subsystem recognizes an alert interruption condition due to either a channel-control-check condition or an interface-control-check condition, then any previously existing intermediate interruption condition caused by a PCI flag in a CCW may or may not be recognized by the channel subsystem.

Detection of the PCI condition does not affect the progress of the I/O operation.

Incorrect Length

Incorrect length occurs when the number of bytes contained in the storage areas assigned for the I/O operation is not equal to the number of bytes requested or offered by the I/O device. Incorrect length is indicated for one of the following reasons:

Long Block on Input: During a read, read-backward, or sense operation, the device attempted to transfer one or more bytes to main storage after the assigned main-storage areas were filled, or the device indicated that more data could have been transferred if the count had been larger. The extra bytes have not been placed in main storage. The count in the SCSW is zero.

Long Block on Output: During a write or control operation, the device requested one or more bytes from the channel subsystem after the assigned main-storage areas were exhausted, or the device indicated that more data could have been transferred if the count had been larger. The count in the SCSW is zero.

Short Block on Input: The number of bytes transferred during a read, read-backward, or sense opera-

tion is insufficient to fill the main-storage areas assigned to the operation. The count in the SCSW is not zero.

Short Block on Output: The device terminated a write or control operation before all information contained in the assigned main-storage areas was transferred to the device. The count in the SCSW is not zero.

The incorrect-length indication is suppressed when the current CCW has the SLI flag set to one and the CD flag set to zero. The indication does not occur for operations rejected during the initiation sequence. The indication also does not occur for immediate operations when the count field is nonzero and the subchannel is in the incorrect-length-suppression mode. The incorrect-length indication is not meaningful when the count field of the SCSW is not meaningful.

Presence of the incorrect-length condition suppresses command chaining unless the SLI flag in the CCW is one or unless the condition occurs in an immediate operation when the subchannel is in the incorrect-length-suppression mode.

Program Check

Program check occurs when programming errors are detected by the channel subsystem. The condition can be due to the following causes:

Invalid CCW-Address Specification: The channel-program address (CPA) or the transfer-in-channel command does not designate the CCW on a doubleword boundary, or bit 0 of the CPA or bit 32 of a format-1 CCW specifying the transfer-in-channel command is not zero.

Invalid CCW Address: The channel subsystem has attempted to fetch a CCW from a main-storage location that is not available. An invalid CCW address can occur because the program has designated an invalid address in the channel-program-address field of the command-mode ORB or in the transfer-in-channel command or because, on chaining, the channel subsystem attempts to fetch a CCW from an unavailable location. A main-storage location is unavailable when any of the following conditions is detected:

1. The absolute CCW address does not correspond to a physical location.

2. Format-0 CCWs are specified in the ORB, and the absolute CCW address is greater than $2^{24} - 1$.
3. Format-1 CCWs are specified in the ORB, and the absolute CCW address is greater than $2^{31} - 1$.

Invalid Command Code: There are zeros in the four rightmost bit positions of the command code in the CCW designated by the CPA or in a CCW fetched on command chaining. The command code is not tested for validity during data chaining.

Invalid Count, Format 0: A CCW, which is other than a CCW specifying transfer in channel, contains zeros in bit positions 48-63.

Invalid Count, Format 1: A CCW that specifies data chaining or a CCW fetched while data chaining contains zeros in bit positions 16-31.

Invalid IDAW-Address or MIDAW-Address Specification: Indirect data addressing or modified indirect data addressing is specified, and any of the following conditions is detected:

1. The ORB specifies format-1 IDAWs, the CCW specifies indirect data addressing, and the contents of the data-address field in the CCW do not designate the first IDAW on a word boundary; that is, bits 30 and 31 (format-0 CCW) or 62 and 63 (format-1 CCW) are not zeros.
2. The ORB specifies format-2 IDAWs, the CCW specifies indirect data addressing, and the contents of the data-address field in the CCW do not designate the first IDAW on a doubleword boundary; that is, bits 29-31 (format-0 CCW) or 61-63 (format-1 CCW) are not zeros.
3. The ORB specifies modified-CCW-indirect-data addressing, the CCW specifies modified-indirect-data addressing, and the contents of the data-address field in the CCW do not designate the first MIDAW on a quadword boundary.

Invalid IDAW or MIDAW Address: The channel subsystem has attempted to fetch an IDAW or MIDAW from a main-storage location that is not available. An invalid IDAW or MIDAW address can occur because the program has designated an invalid address in a CCW that specifies indirect data addressing or modified indirect data addressing, or because the channel subsystem, on sequentially fetching IDAWs or MIDAWs, attempts to fetch from

an unavailable location. A main-storage location is unavailable when any of the following conditions is detected:

1. The absolute IDAW or MIDAW address does not correspond to a physical location.
2. Format-0 CCWs are specified in the ORB, and the absolute IDAW or MIDAW address is greater than $2^{24} - 1$.
3. Format-1 CCWs are specified in the ORB, and the absolute IDAW or MIDAW address is greater than $2^{31} - 1$.

Invalid Data-Address Specification: Bit 32 of a format-1 CCW is not zero.

Invalid Data Address: When any of the following conditions is detected, an invalid data address is recognized by the channel subsystem.

1. Use of the data address has caused the channel subsystem to attempt to wrap from the maximum storage address to zero.
2. Use of the data address has caused the channel subsystem to attempt to wrap from zero to the maximum storage address during a read-backward operation.
3. The channel subsystem has attempted to transfer data to a storage location that is unavailable.

An invalid data address can occur because the program has designated an unavailable location in a CCW or in an IDAW or MIDAW, or because the channel subsystem, on sequentially accessing storage, attempted to access an unavailable location. A main-storage location is unavailable when any of the following conditions is detected:

1. The absolute address of the location does not correspond to a physical location.
2. Format-0 CCWs are specified in the ORB, indirect data addressing is not specified in the CCW, and the absolute address is greater than $2^{24} - 1$.
3. Format-1 CCWs are specified in the ORB, indirect data addressing is not specified in the CCW, and the absolute address is greater than $2^{31} - 1$.
4. Format-1 IDAWs are specified in the ORB, indirect data addressing is specified in the CCW, and the absolute address is greater than $2^{31} - 1$.

5. When the address-limit-checking facility is installed, the absolute address is outside the addressing range specified by SET ADDRESS LIMIT, and the limit mode at the subchannel is active.

Note: The maximum storage address is determined as a function of the CCW and IDAW formats used or MIDAWs used. When neither an IDAW nor a MIDAW is used, the maximum storage address is a function of the CCW format specified, as follows:

1. When 24-bit (format 0) CCWs are specified, the maximum storage address recognized by the channel subsystem is $2^{24} - 1$.
2. When 31-bit (format 1) CCWs are specified, the maximum storage address recognized by the channel subsystem is $2^{31} - 1$.

When an IDAW is used, the maximum storage address is a function of the IDAW format specified, as follows:

1. When 31-bit (format 1) IDAWs are specified, the maximum storage address recognized by the channel subsystem is $2^{31} - 1$.
2. When 64-bit (format 2) IDAWs are specified, the maximum storage address recognized by the channel subsystem is $2^{64} - 1$.

When a MIDAW is used, the maximum storage address recognized by the channel subsystem is $2^{64} - 1$.

Invalid IDAW or MIDAW Specification: When any of the following conditions is detected, an invalid IDAW specification is recognized by the channel subsystem:

1. Bit 0 of a format-1 IDAW is not zero.
2. A second or subsequent format-1 IDAW does not designate the location of the beginning byte of a 2K-byte block or, for read-backward operations, the location of the ending byte of a 2K-byte block.
3. A second or subsequent format-2 IDAW does not designate the location of the beginning byte of a 2K-byte or 4K-byte block, as required by the 2K-IDAW control in the ORB, or, for read-backward operations, the location of the ending byte of a 2K-byte or 4K-byte block.

When bit 0 of a format-1 IDAW is not zero, the sub-channel is made status pending with program-check indicated only after the I/O device attempts to transfer data, just as when an invalid data address that refers to a nonexistent data area is detected. If the device ends the operation prior to the transfer of data, no error condition is reported. See “CCW Indirect Data Addressing” in Chapter 15 for additional information about the reporting of program-check conditions.

When any of the following conditions is detected, an invalid MIDAW specification is recognized by the channel subsystem:

1. One or more bits defined as reserved in the MIDAW are not zero.
2. The MIDAW count field contains zeros.
3. The combination of the MIDAW data-address and count fields specify data addresses that cross a 4K-byte block boundary, unless skipping is in effect for the MIDAW.
4. The MIDAW in control has its last-MIDAW flag set to zero and the sum of the count values of the previous MIDAWs in the MIDAL and the MIDAW in control is greater than or equal to the count value in the CCW.
5. The MIDAW in control has its last-MIDAW flag set to one and the sum of the count values in the previous MIDAWs in the MIDAL and the MIDAW in control do not equal the count value in the CCW.
6. The MIDAW in control is a second or subsequent MIDAW whose address is on a 4K-byte boundary.

Invalid CCW, Format 0: A CCW other than a CCW specifying transfer in channel does not contain a zero in bit position 39.

Invalid CCW, Format 1: A CCW other than a CCW specifying transfer in channel does not contain a zero in bit position 15, or a CCW specifying transfer in channel does not contain zeros in bit positions 0-3 and 8-31.

Invalid Suspend Flag: A format-0 or format-1 CCW fetched during data chaining, other than a CCW specifying transfer in channel, does not contain a zero in bit position 38 or 14, respectively. A CCW

other than a CCW specifying transfer in channel does not contain a zero in bit position 38 for a format-0 CCW or bit position 14 for a format-1 CCW, and suspend control was not specified by bit 4 of word 1 of the ORB.

Invalid MIDA Flag: When either of the following conditions is detected, an invalid MIDA flag condition is recognized by the channel subsystem.

1. Bit 39 (format 0) or bit 15 (format 1) of the CCW is one, specifying modified CCW indirect data addressing; however, modified CCW indirect data addressing was not specified by bit 25 of word 1 of the ORB.
2. Bit 39 (format 0) or bit 15 (format 1) of the CCW is one, specifying modified CCW indirect data addressing, and bit 35 (format 0) or bit 11 (format 1) of the CCW is one, specifying skip.
3. Bit 39 (format 0) or bit 15 (format 1) of the CCW is one, specifying modified CCW indirect data addressing, and bit 37 (format 0) or bit 13 (format 1) of the CCW is one, specifying indirect data addressing.
4. Bit 39 (format 0) or bit 15 (format 1) of the CCW is one, specifying modified CCW indirect data addressing, and the designated subchannel is not associated with channel paths that support modified CCW indirect data addressing.

Invalid ORB Format: One or more reserved bit positions in the operation-request block (ORB) is not zero. (See “Operation-Request Block” on page 15-24 for more information.) If the incorrect-length-indication-suppression facility is not installed, then bit 24 of word 1 of the command-mode ORB must also be zero.

Invalid Sequence: The channel subsystem has fetched two successive CCWs both of which specify transfer in channel, or, depending on the model, a sequence of 256 or more CCWs with command chaining specified was executed by the channel subsystem and did not result in the transfer of any data to or from an I/O device.

Detection of the program-check condition during the initiation of an operation at the device causes the operation to be suppressed and the subchannel to be made status pending with primary, secondary, and alert status. When the condition is detected after the

I/O operation has been initiated at the device, the device is signaled to conclude the operation the next time the device requests or offers a byte of data or status. In this situation, the subchannel is made status pending as a function of the status received from the device. The program-check condition causes command chaining and command retry to be suppressed.

Protection Check

Protection check occurs when the channel subsystem attempts a storage access that is prohibited by the protection mechanism. Protection applies to the fetching of CCWs, IDAWs, MIDAWs, and output data and to the storing of input data. The subchannel key provided in the ORB is used as the access key for storage accesses associated with an I/O operation.

Detection of the protection-check condition during the fetching of the first CCW, IDAW, or MIDAW causes the operation to be suppressed and the subchannel to be made status pending with primary, secondary, and alert status. When protection check is detected after the I/O operation has been initiated at the device, the device is signaled to conclude the operation after the available data logically prior to the protection check has been transferred. However, if an access violation occurs when the channel subsystem is in the process of fetching either a new IDAW, MIDAW, or a new CCW while data chaining, and if the device signals the channel-end condition before transferring any data designated by the new CCW, IDAW or MIDAW, then the status is accepted, and the subchannel becomes status pending with primary and alert status and with protection check indicated. Other indications may accompany the protection-check indication as a function of the operation specified by the CCW, the status received from the device, and the current state of the subchannel. The protection-check condition causes command chaining and command retry to be suppressed.

Channel-Data Check

Channel-data check indicates that an uncorrected storage error has been detected in regard to data, contained in main storage, that is currently used in the performance of an I/O operation. The condition may be indicated when detected, even if the data is not used when prefetched. Channel-data check is indicated when data or the associated key has an invalid checking-block code (CBC) in main storage

when that data is referenced by the channel subsystem.

On an input operation, when the channel subsystem attempts to store less than a complete checking block, and invalid CBC is detected on the checking block in storage, the contents of the location remain unchanged and with invalid CBC. On an output operation, whenever channel-data check is indicated, no bytes from the checking block with invalid CBC are transferred to the device.

During a storage access, the maximum number of bytes that can be transferred is model dependent. If a channel-data-check condition is recognized during that storage access, the number of bytes transferred to or from storage may not be detectable by the channel subsystem. Consequently, the number of bytes transferred to or from storage may not be correctly reflected by the residual count. However, the residual count that is stored in the SCSW, when used in conjunction with the storage-access code and the CCW address, designates a byte location within the block of main storage in which the channel-data-check condition was recognized.

A condition indicated as channel-data check causes the current operation, if any, to be terminated. The subchannel becomes status pending with primary and alert status, or with primary, secondary, and alert status, as a function of the status received from the device. The count and address fields of the SCSW stored by TEST SUBCHANNEL pertain to the operation terminated. The extended-status-word-format bit is one, and subchannel-logout information is stored in the ESW, when TEST SUBCHANNEL is executed.

Whenever the channel-data-check condition pertains to prefetched data, the failing-storage-address-validity flag, bit 6 of the ERW, is one. An address of a location within the checking block for which the channel-data-check condition is generated is stored in the ESW failing-storage-address field.

Uncorrectable storage or key errors detected on prefetched data while the subchannel is start pending cause the operation to be canceled before initiation at the device. In this case, the subchannel is made status pending with primary, secondary, and alert status, with channel-data check indicated, and with the ESW failing-storage address stored.

Whenever channel-data check is indicated, no measurement data for the subchannel is stored.

Channel-Control Check

Channel-control check is caused by any machine malfunction affecting channel-subsystem controls. The condition includes invalid CBC on a CCW, an IDAW, a MIDAW, or the respective associated key. The condition may be indicated when an invalid CBC is detected on a prefetched CCW, IDAW, MIDAW, or the respective associated key, even if that CCW, IDAW, or MIDAW is not used.

Channel-control check may also indicate that an error has been detected in the information transferred to or from main storage during an I/O operation. However, when this condition is detected, the error has occurred inboard of the channel path: in the channel subsystem or in the path between the channel subsystem and main storage.

Detection of the channel-control-check condition causes the current operation, if any, to be terminated immediately. The subchannel is made status pending with primary and alert status or with primary, secondary, and alert status as a function of the type of termination, the current subchannel state, and the device status presented, if any. When the channel subsystem recognizes a channel-control-check condition, any previously existing intermediate interruption condition caused by a PCI flag in a CCW may or may not be recognized by the channel subsystem. The count and data-address fields of the SCSW stored by TEST SUBCHANNEL pertain to the operation terminated. The extended-status-word-format bit is one, and subchannel-logout information is stored in the ESW, when TEST SUBCHANNEL is executed.

Whenever the channel-control-check condition pertains to an invalid CBC detected on a prefetched CCW, a prefetched IDAW, a prefetched MIDAW, or the key associated with the prefetched CCW, the prefetched IDAW, or the prefetched MIDAW, an extended-report word with bit 6 set to one, and the failing-storage address, are stored in the ESW when TEST SUBCHANNEL is executed.

Channel-control-check conditions encountered while prefetching when the subchannel is start pending cause the operation to be canceled before initiation at the device. In this case, the subchannel is made status pending with primary, secondary, and alert status, with channel-control check indicated, and with a failing-storage address that will be stored in the ESW.

If a subchannel is halt pending and the channel subsystem encounters a channel-control-check condition while performing the halt function for that subchannel, the subchannel remains halt pending unless the channel subsystem can determine that the halt signal was issued. The subchannel remains halt pending even if the channel subsystem was attempting to issue the halt signal and is unable to determine if the halt signal was issued.

If a subchannel is start pending or resume pending and the channel subsystem encounters a channel-control-check condition while performing the start function for that subchannel, the subchannel remains start pending or resume pending unless the channel subsystem can determine that the first command was accepted. The subchannel remains start pending or resume pending even if the channel subsystem was attempting to initiate the I/O operation for the first command and is unable to determine if the command was accepted. If the channel subsystem is unable to determine whether the first command was accepted, the subchannel is made status pending with at least alert and primary status.

In some situations in which a channel-subsystem malfunction exists, the channel-control-check condition may be reported as a machine-check condition.

Whenever channel-control check is indicated, no measurement data for the subchannel is stored.

Programming Note: If the status-control field of the SCSW indicates that the subchannel is status pending with alert status but the field-validity flags of the SCSW indicate that the device-status field is not usable for error-recovery purposes, the program should (1) assume that the channel-control-check condition occurred while the channel subsystem was accepting alert status from the device and (2) take the appropriate action for alert status, even though the status itself has been lost.

Interface-Control Check

Interface-control check indicates that an invalid signal has occurred on the channel path. The condition is detected by the channel subsystem and usually indicates malfunctioning of an I/O device. Interface-control check can occur for any of the following reasons:

1. A data or status byte received from a device while the subchannel is subchannel-and-device

active or device active has an invalid checking-block code.

2. The status byte received from a device while the subchannel is idle, start pending, suspended, or halt pending has an invalid checking-block code.
3. A device responded with an address other than the address designated by the channel subsystem during initiation of an operation.
4. During command chaining, the device appeared not operational.
5. A signal from an I/O device either did not occur or occurred at an invalid time or had an invalid duration.
6. The channel subsystem recognized the I/O-error-alert condition (see “I/O-Error Alert (A)” on page 16-50).
7. ESW bit 26, indicating device-status check, is set to one.

Detection of an interface-control check during a CCW-type IPL I/O operation on a machine that retries IPL I/O operations on an alternate channel path causes the channel subsystem to terminate operations on the current channel path and to retry the IPL I/O operation on another logically-available channel path to the designated device, if one is available, without causing the subchannel to become status-pending. (See “CCW-type IPL” on page 17-17 for additional information.) In all other cases and when an interface-control check has been detected for a CCW-type IPL operation on all logically-available channel paths to the designated device, detection of the interface-control-check condition causes the current operation, if any, to be terminated immediately, and the subchannel is made status pending with alert status, primary and alert status, secondary and alert status, or primary, secondary, and alert status as a function of the type of termination, the current subchannel state, and the device status presented, if any. When the channel subsystem recognizes an interface-control-check condition, any previously existing intermediate interruption condition caused by a PCI flag in a CCW may or may not be recognized by the channel subsystem. The extended-status-word-format bit is one, and subchannel-logout information is stored in the ESW, when TEST SUBCHANNEL is executed.

If a subchannel is halt pending and the channel subsystem encounters an interface-control-check condition while performing the halt function for that subchannel, the subchannel remains halt pending unless the channel subsystem can determine that the halt signal was issued. The subchannel remains halt pending even if the channel subsystem was attempting to issue the halt signal and is unable to determine if the halt signal was issued.

If a subchannel is start pending or resume pending and the channel subsystem encounters an interface-control-check condition while performing the start function for that subchannel, the subchannel remains start pending or resume pending unless the channel subsystem can determine that the first command was accepted. The subchannel remains start pending or resume pending even if the channel subsystem was attempting to initiate the I/O operation for the first command and is unable to determine if the command was accepted. If the channel subsystem is unable to determine whether the first command was accepted, the subchannel is made status pending with at least alert and primary status.

If, while initiating a signaling sequence with the channel subsystem for the purpose of presenting status or transferring data, the device presents an address with invalid parity, the error condition is not made available to the program since the identity of the device and associated subchannel are unknown.

Whenever interface-control check is indicated, no measurement data for the subchannel is stored.

Programming Note: If the status-control field of the SCSW indicates that the subchannel is status pending with alert status but the field-validity flags of the SCSW indicate that the device-status field is not usable for error-recovery purposes, the program should (1) assume that the interface-control-check condition occurred while the channel subsystem was accepting alert status from the device and (2) take the appropriate action for alert status, even though the status itself has been lost.

Chaining Check

Chaining check is caused by channel-subsystem overrun during data chaining on input operations. The condition occurs when the I/O-data rate is too high for the particular resolution of data addresses. Chaining check cannot occur on output operations.

Detection of the chaining-check condition causes the I/O device to be signaled to conclude the operation. It causes command chaining to be suppressed.

Count Field

Bit positions 16-31 of word 2 contain the residual count. The count is to be used in conjunction with the original count specified in the last CCW and, depending upon existing conditions (see Figure 16-7 on page 16-32), indicates the number of bytes trans-

ferred to or from the area designated by the CCW. The count field is meaningful whenever the subchannel is status pending with primary status that consists of either (1) device status only or (2) device status together with subchannel status of incorrect length only, PCI only, or both.

In Figure 16-7 on page 16-32, the contents of the count field are listed for all cases where the subchannel is start pending, subchannel-and-device active, device active, suspended, or status pending.

Subchannel State ¹	Count ²
Start pending (UUUU0/AIP SX) ²	Not meaningful ³
Start pending and status pending (10YY1/AIP SX) ²	Not meaningful ³
Start pending and status pending (00111/AIP SX) because the device appeared not operational on all paths ²	Not meaningful ³
Start pending and device active (UUUU0/AIP SX) ²	Not meaningful ³
Suspended (YYYYY/AIP SX) ²	Not meaningful ³
Subchannel-and-device active (UUUU0/AIP SX) ²	Not meaningful ³
Device active (UUUU0/AIP SX)	Not meaningful ³
Status pending (01001/AIP SX) because of program-controlled-interruption condition or initial-status interruption	Not meaningful ³
Status pending (1Y1Y1/AIP SX); termination occurred because of: ²	
Program check	Not meaningful ³
Protection check	Not meaningful ³
Chaining check	Not meaningful ³
channel-control check	See note 1
Interface-control check	Not meaningful ³
Channel-data check	See note 2
Status pending (YY1Y1/AIP SX); termination occurred under count control ²	Correct
Status pending (Y0011/AIP SX) ²	Not meaningful ³
Status pending (1Y1Y1/AIP SX) ²	Correct; residual count of last used CCW
Status pending (1Y111/AIP SX); command chaining suppressed because of alert status ²	Correct; residual count of last used CCW
Status pending (YYYY1/AIP SX); after termination by HALT SUBCHANNEL ²	Unpredictable
Status pending (00001/AIP SX); after termination by CLEAR SUB-CHANNEL	Not meaningful ³
Status pending (YY1Y1/AIP SX); operation completed normally at the subchannel ²	Correct; indicates the residual count
Status pending (1Y111/AIP SX); command chaining terminated because of alert status ²	Correct; original count of CCW specifying the new I/O operation
Status pending (10001/AIP SX) because of alert status	Not meaningful ³

Figure 16-7. (Part 1 of 2) Contents of Count Field in SCSW

Explanation:	
1	<p>In situations where more than a single condition exists because of, for example, alert status that is described by program check and unit check, the entry appearing first in the table takes precedence.</p> <p>The meaning of the notation used in this column is as follows:</p> <p>A Alert status I Intermediate status P Primary status S Secondary status X Status Pending</p> <p>The allowed combinations of status-control-bit settings is shown to the left of the "/" symbol by the use of these symbols:</p> <p>0 Corresponding condition is not indicated 1 Corresponding condition is indicated U Unpredictable. The corresponding condition is not meaningful when the subchannel is not status pending. Y The corresponding condition is not significant and is indicated as a function of the subchannel state.</p>
2	The subchannel may also be resume pending.
3	The contents of the count field are not meaningful because the count field is not valid when the SCSW is stored and the subchannel is in the given state.
Notes:	
1.	The count is unpredictable unless IDAW/MIDAW check is indicated, in which case the count may not correctly reflect the number of bytes transferred to or from main storage but will (when used in conjunction with the CCW address) designate a byte location within the page in which the channel-control-check condition was recognized.
2.	During a storage access, the maximum number of bytes that can be stored by a channel subsystem is model dependent. If a channel-data-check condition is recognized during that access, the number of bytes transferred to or from storage may not be detectable by the channel subsystem. Consequently, the number of bytes transferred to or from storage may not be correctly reflected by the residual count. However, the residual count that is stored when used in conjunction with the storage-access code and the CCW address designates a byte location within the page in which the channel-data-check condition was recognized.

Figure 16-7. (Part 2 of 2) Contents of Count Field in SCSW

Transport-Mode SCSW

The format of a transport-mode SCSW is as follows:

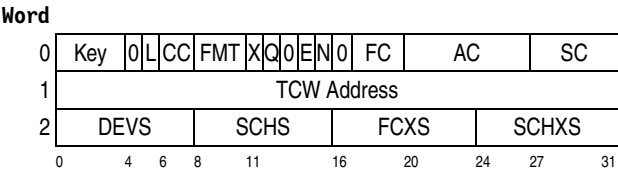


Figure 16-8. Transport-Mode SCSW Format

Figure 16-9 on page 16-34 shows the summary and contents of the transport-mode SCSW:

Bits	Name
Word 0	
0-3	Subchannel Key
4	Reserved
5	ESW format (L)
6-7	Deferred condition code (CC)
8-10	Format (FMT)
11	IRB-Format control (X)
12	Interrogate-complete (Q)
13	Reserved
14	Extended control (E)
15	Path not operational (N)
16	Reserved
17-19	Function Control (FC)
17	Start function
18	Halt function
19	Clear function
20-26	Activity Control (AC)
20	Reserved
21	Start pending
22	Halt pending
23	Clear pending
24	Reserved
25	Device active
26	Reserved
27-31	Status Control (SC)
27	Alert status
28	Intermediate status
29	Primary status
30	Secondary status
31	Status pending
Word 1	
0-31	TCW Address
Word 2	
0-7	Device Status (DEVS)
0	Attention
1	Status modifier
2	Control-unit end
3	Busy
4	Channel end
5	Device end
6	Unit check
7	Unit exception
8-15	Subchannel Status (SCHS)
8	Reserved
9	Reserved
10	Program check
11	Protection check
12	Channel-data check
13	Channel-control check
14	Interface-control check
15	Channel-subsystem retry failed
16-23	FCX Status (FCXS)
16	Reserved
17	Reserved
18	Reserved
19	Reserved
20	Reserved
21	Reserved
22	Reserved
23	TSB Valid
24-31	Subchannel-Extended Status (SCHXS)
24	Interrogate-Failed (F)
25-31	Subchannel-Extended-Status Qualifier (SESQ)

Figure 16-9. Transport-Mode SCSW

The following descriptions indicate when an SCSW field contains meaningful information for a transport-mode IRB.

Subchannel Key

This bit has the same meaning as in the command mode SCSW. (See “Subchannel Key” on page 16-9).

Reserved

SCSW bits 4, 13, 16, 20, 24, and 26 of word 0, bits 0 of word 1, and bits 1, 8, 9, 15, and 16-22 of word 2 are reserved and stored as zeros.

Extended-Status-Word Format (L)

This bit has the same meaning as in the command-mode SCSW. (See “Extended-Status-Word Format (L)” on page 16-9).

Deferred Condition Code (CC)

This field has the same meaning as in the command-mode SCSW. (See “Deferred Condition Code (CC)” on page 16-9).

Format (FMT)

Bits 8-10 of word 0 are the IRB-format field that contains a 3-bit, unsigned integer whose value specifies the layout of the IRB, when the IRB-format control bit (X), is set to one.

FMT	Description
0	Transport-Mode IRB
1-7	Reserved

IRB-Format Control (X)

Bit 11 of word 0 is the IRB-format control bit. The value of this bit is one for the transport-mode IRB. (See Figure 16-2 on page 16-7).

Interrogate Complete (Q)

Bit 12 of word 0 is the interrogate-complete bit (Q). When one, Q indicates that the interrogate operation has completed successfully, and the subchannel is made status-pending with intermediate interruption status. If the pending intermediate interruption condition has not been cleared by TEST SUBCHANNEL by the time the primary interruption condition for the original interrogated operation is presented at the subchannel, both conditions may be merged and indicated together in the SCSW.

Subchannel-Control Field

The subchannel-control information is contained in bit positions 13-31 of the first word of the SCSW.

Extended Control (E)

Bit 14 of word 0, when one, indicates that model-dependent information is stored in the extended-control word (ECW). When bit 14 is zero, the contents of words 0-7 of the ECW are unpredictable. The E bit is meaningful whenever the subchannel is status pending with alert status either alone or together with primary status, secondary status, or both. The concurrent sense information has no meaning when a transport-mode IRB is stored.

Path Not Operational (N)

This bit has the same meaning as in the command-mode SCSW. See “Path Not Operational (N)” on page 16-13.

Function Control (FC)

The function-control field indicates the basic I/O functions that are indicated at the subchannel. This field may indicate the acceptance of as many as two functions. The function-control field is contained in bit positions 17-19 of the first word of the SCSW. The function-control field is meaningful at an installed subchannel whenever the subchannel is valid (see “Function Control (FC)” on page 16-13). The function-control field contains all zeros whenever both the activity- and status-control fields contain all zeros. The meaning of the individual bits is as follows:

Start Function (Bit 17): This bit has the same meaning as in the command-mode SCSW. (See “Start Function (Bit 17)” on page 16-13).

Halt Function (Bit 18): When one, bit 18 indicates that a halt function has been requested and is either pending or in progress at the subchannel. A halt function is requested by the execution of HALT SUBCHANNEL. A halt function is indicated at the subchannel when condition code 0 is set for HALT SUBCHANNEL. The halt-function indication is cleared at the subchannel when the next status-pending condition that occurs is cleared by the execution of TEST SUBCHANNEL. The next status-pending condition depends on the state of the subchannel when HALT SUBCHANNEL is executed. If the subchannel is start pending or device active when HALT SUBCHANNEL is executed, then the

next status-pending condition is status pending with at least primary status indicated. The halt-function indication is also cleared at the subchannel during the execution of CLEAR SUBCHANNEL.

Clear Function (Bit 19): This bit has the same meaning as in the command-mode SCSW. (See “Clear Function (Bit 19)” on page 16-14).

Activity Control (AC)

The activity-control field is contained in bit positions 20-26 of word 0 of the SCSW. The activity-control field is meaningful at an installed subchannel whenever the subchannel is valid (see “Device Number Valid (V)” on page 15-4). However, if an IFCC or CCC condition is detected during the performance of a basic I/O function and that function is indicated as pending, I/O operations may or may not have been performed at the device. The activity-control bits are defined as follows:

Bit Meaning

- 20 Reserved
- 21 Start pending
- 22 Halt pending
- 23 Clear pending
- 24 Reserved
- 25 Device active
- 26 Reserved

Start Pending (Bit 21): This bit has the same meaning as in the command-mode SCSW with the exception that the subchannel remains start pending until a primary interruption condition is generated. (See “Start-Pending (Bit 21)” on page 16-15).

Halt Pending (Bit 22): This bit has the same meaning as in the command-mode SCSW. (See “Halt-Pending (Bit 22)” on page 16-16).

Clear Pending (Bit 23): This bit has the same meaning as in the command-mode SCSW. (See “Clear-Pending (Bit 23)” on page 16-16).

Device Active (Bit 25): When one, bit 25 indicates that the subchannel is device active. A subchannel is said to be device active when an I/O operation is currently in progress at the associated device. The subchannel becomes device active, only when primary status is presented without secondary.

Status Control (SC)

The status-control field is contained in bit positions 27-31 of word 0 of the SCSW. This field provides the program with a summary-level indication of the interruption condition described by either subchannel or device status or the Q bit. More than one summary indication may be signaled as a result of existing conditions at the subchannel. Whenever the subchannel is enabled (see “Enabled (E)” on page 15-3) and at least bit 31 is one, the subchannel is said to be status pending. Whenever the subchannel is disabled, the subchannel is not made status pending. Bit 31 of SCSW word 0 is meaningful at an installed subchannel whenever the subchannel is valid (see “Device Number Valid (V)” on page 15-4); bits 27-30 are meaningful when bit 31 is one. The status-control bits are defined as follows:

Alert Status (Bit 27): This bit has the same meaning as in the command-mode SCSW with the exception that a command-mode IRB is stored when alert status is presented with secondary status. (See “Alert Status (Bit 27)” on page 16-17).

Intermediate Status (Bit 28): Intermediate status can be indicated when the interrogate operation associated with the TCW is completed, and the Q-bit (interrogate-complete) is one.

Primary Status (Bit 29): This bit has the same meaning as in the command-mode SCSW. (See “Primary Status (Bit 29)” on page 16-19).

Secondary Status (Bit 30): This bit has the same meaning as in the command-mode SCSW. (See “Secondary Status (Bit 30)” on page 16-19).

Status Pending (Bit 31): This bit has the same meaning as in the command-mode SCSW. (See “Status-Pending (Bit 31)” on page 16-19).

TCW Address Field

Bits 1-31 of word 1 form an absolute address of a TCW. The TCW address indicated is the address of the current TCW when the subchannel becomes pending with primary interruption. When the subchannel-status field indicates channel-control check, channel-data check, program check or interface-control check, the TCW-address field is usable for recovery purposes if the TCW-address field-validity flag in the ESW is one.

Subchannel State	TCW Address
Start pending (UUUU0/AIP SX)	Unpredictable
Start pending and device active (UUUU0/AIP SX)	Unpredictable
Subchannel-and-device active (UUUU0/AIP SX)	N/A
Device active only (UUUU0/AIP SX)	Unpredictable
Suspended (YYYYY/AIP SX)	N/A
Status pending (10001/AIP SX) because of unsolicited alert status from the device while the subchannel was start pending	Address of current TCW ³
Status pending (0Y111/AIP SX) because the device appeared not operational on all paths	Address of current TCW ³
Status pending (10011/AIP SX) because of solicited alert status from the device while the subchannel was start pending and device active	Address of current TCW ³
Status pending (10111/AIP SX) because of solicited alert status generated by the channel subsystem while the subchannel was start pending or start pending and device active	See note 2
Status pending (01001/AIP SX) for the program-controlled-interruption condition while the subchannel was subchannel-and-device active	N/A
Status pending (01001/AIP SX) for the initial-status-interruption condition while the subchannel was subchannel-and-device active	N/A
Status pending (01001/AIP SX) for the intermediate/interrogate	Address of current TCW ²
Status pending (1Y1Y1/AIP SX); termination occurred because of program check caused by one of the following conditions:	
Bit 24 of word 1 of the ORB set to one; incorrect-length-indication-suppression facility not installed	N/A
Unused bits in ORB not set to zeros	Address of current TCW ²
Invalid TCW-address specification in the channel-program address in the ORB	Address of invalid TCW

Figure 16-10. (Part 1 of 4) TCW Address as Function of Subchannel State

Subchannel State	TCW Address
Invalid TCW address in the channel-program address in the ORB	Address of invalid TCW
Invalid TCW address while DCW data chaining	N/A
Invalid TCW count or TIDAW count	Address of invalid TCW ⁴
Invalid TIDAW-address specification	Address of invalid TCW ⁴
Invalid TIDAW address in TCW	Address of invalid TCW ⁴
Invalid TIDAW address while sequentially fetching TIDAWs	N/A
Invalid data-address specification	Address of invalid TCW ⁴
Invalid data address in a TCW	Address of invalid TCW ⁴
Invalid data address while sequentially accessing storage	Address of current TCW
Invalid data address in TIDAW	Address of current TCW
Invalid TIDAW	Address of current TCW
Invalid TSB-address specification	Address of current TCW
Invalid TSB address	Address of current TCW
Invalid TCCB address specification	Address of current TCW
Invalid TCCB address	Address of current TCW
Device-detected program checks	Address of current TCW
Status pending (1Y1Y1/AIP SX); termination occurred because of protection check detected as follows:	
On a TCW access	Address of protected TCW ⁴
On data or an TIDAW access	Address of current TCW

Figure 16-10. (Part 2 of 4) TCW Address as Function of Subchannel State (Continued)

Subchannel State	TCW Address
Status pending (1Y1Y1/AIP SX); termination occurred because of chaining check	N/A
Status pending (YY1Y1/AIP SX); termination occurred under count control	Address of current TCW
Status pending (1Y1Y1/AIP SX); operation prematurely terminated by the device because of alert status	Address of current TCW ³
Status pending (YYYY1/AIP SX) after termination by HALT SUBCHANNEL and the activity-control-field bits indicated below set to ones:	
Status pending alone	Unpredictable
Start pending	Unpredictable
Device active and start pending	Unpredictable
Device active	Unpredictable
Status pending (00001/AIP SX) after termination by CLEAR SUBCHANNEL	Unpredictable
Status pending (YY1Y1/AIP SX); operation completed normally at the subchannel	Address of current TCW
Status pending (00011/AIP SX)	Unpredictable
Status pending (10001/AIP SX)	Unpredictable
Status pending (00001/AIP SX)	Unpredictable
Status pending (1Y111/AIP SX); command chaining suppressed because of alert status other than channel-control check or interface-control check	N/A
Status pending (1YYY1/AIP SX) because of alert status for channel-control check or interface-control check	See note 3
Status pending (1Y1Y1/AIP SX) because of channel-data check	Address of current TCW
Status pending (1Y1Y1/AIP SX) because of channel-subsystem retry failed	Address of current TCW

Figure 16-10. (Part 3 of 4) TCW Address as Function of Subchannel State (Continued)

Explanation:	
1	<p>The meaning of the notation used in this column is as follows:</p> <p>A Alert status I Intermediate status P Primary status S Secondary status X Status Pending</p> <p>The possible combination of status-control-bit settings is shown to the left of the “/” symbol by the use of these symbols:</p> <p>0 Corresponding condition is not indicated 1 Corresponding condition is indicated U Unpredictable. The corresponding condition is not meaningful when the subchannel is not status pending. Y The corresponding condition is not significant and is indicated as a function of the subchannel state.</p>
2	A TCW becomes current when start subchannel is issued, specifying a transport-mode ORB, and the instruction completes with condition code 0.
3	Command-mode IRB may be stored depending on the current subchannel status
4	The stored address is the address of the current TCW even though it is either invalid or protected.
5	Not applicable
Notes:	
1.	The address stored is unpredictable.
2.	The address of the TCW is given as a function of the alert status indicated.
3.	Bit 21 of the subchannel-logout information, when stored as one, indicates that the address is the address of the current TCW.

Figure 16-10. (Part 4 of 4) TCW Address as Function of Subchannel State (Continued)

Device-Status Field

The device-status field is contained in bit positions 0-7 of word 2 of the SCSW. Each bit of the device-status field has the same meaning as in the command-mode SCSW. See “Device-Status Field” on page 16-24 for details.

Subchannel-Status Field

The subchannel-status field is contained in bit positions 8-15 of word 2 of the SCSW. Subchannel-status conditions are detected and indicated in the SCSW by the channel subsystem. Except for the conditions caused by equipment malfunctioning, they can occur only while the channel subsystem is involved with the performance of a halt or start function. The subchannel-status field is meaningful whenever the subchannel is status pending with any combination of primary, secondary pending, intermediate, or alert status. When the subchannel is status pending with deferred condition code 3 indicated, the contents of the sub-

channel-status field are not meaningful. The following subchannel-status conditions are described:

Program Check

Program check occurs when programming errors are detected by the channel subsystem. For a transport-mode operation, programming errors may also be detected by the I/O device and are reported as program checks. When a program check condition is recognized, additional information may be available in the “Subchannel-Extended-Status Qualifier (SESQ)” section of the subchannel-extended-status field when a program check is indicated.

Whenever the program-check condition pertains to the fetching of data, the failing-storage-address validity flag, bit 6 of the ERW, is one. An address of a location within the checking block for which the program-check condition is generated is stored in the ESW failing-storage-address field.

A program check condition can be due to any of the following reasons:

Invalid TCW Specification: When any of the following conditions is detected, an invalid TCW specification is recognized:

1. A reserved field that is checked for zeros in the TCW does not contain zeros.
2. A non-zero value is specified in the TCW format field.
3. The read and write bits in the TCW are both one.
4. The TCCB-length field in a TCW specifies a length that is less than 12 or greater than 244.

Invalid TCW-Address Specification: An invalid TCW address specification can occur because the TCW address is not designated on a 64-byte boundary.

Invalid TCW Address: The channel-subsystem has attempted to fetch a TCW from a main-storage location that is not available. An invalid TCW address can occur because the program has designated an invalid address in the channel-program-address field of the transport-mode ORB, when the B-bit (channel-program-type control), is one. A main-storage location is unavailable when the absolute TCW address does not correspond to a physical location.

Invalid TCW Counts: When any of the following conditions is detected by the channel subsystem, an invalid TCW byte count is recognized:

1. The write-bit is one in the TCW and the value in the TCW output-count field is zero.
2. The read-bit is one in the TCW and the value in the TCW input-count field is zero.

Invalid TSB-Address Specification: An invalid transport-status-block address specification can occur if the content of the transport-status-block-address field specified in the TCW does not designate a storage location on a doubleword boundary.

Invalid TSB Address: The channel-subsystem has attempted to store a TSB to a main-storage location that is not available. An invalid transport-status-block address can occur because the program has designated an unavailable location in a TCW, or the channel-subsystem has attempted to store the transport status to a main-storage location that is not available. A main-storage location is unavailable when the

absolute address of the location does not correspond to a physical location.

Invalid TCCB Address: The channel-subsystem has attempted to fetch a TCCB from a main-storage location that is not available. An invalid transport-command-control-block address can occur because the program has designated an unavailable location in a TCW, or has specified a transport-command-control-block address and a length which caused a list of TIDAW to cross a 4K-byte when the transport-command-control-block-TIDA flag in the TCW is one. A main-storage location is unavailable when the absolute address of the location does not correspond to a physical location. **Invalid TIDAW-Address Specification:** Transport-indirect data addressing is specified, and any of the following conditions is detected:

1. A write operation is requested, the output-TIDA flag in a TCW is one, and the contents of the output-data-address field in the TCW do not designate a storage location on a quadword boundary.
2. A read operation is requested, the input-TIDA flag in a TCW is one, and the contents of the input-data-address field in the TCW do not designate a storage location on a quadword boundary.
3. The transport-command-control-block-TIDA flag in the TCW is one, and the contents of the transport-command-control-block-address do not designate a storage location on a quadword boundary.

Invalid Data Address: An invalid data address can occur because the program has designated an unavailable location specified by a TCW or a TIDAW. When any of the following conditions is detected, an invalid data address is recognized by the channel subsystem.

1. The channel-subsystem has attempted to access a main-storage location that is not available. A main-storage location is unavailable when the absolute address of the location does not correspond to a physical location.
2. The channel subsystem, on sequentially accessing storage for a data fetch or data store, attempted to access an unavailable location.
3. Use of the data address has caused the channel subsystem to attempt to wrap from the maximum storage address to zero.

Invalid TIDAW Specification: When the use of TIDAW is specified, an invalid TIDAW specification is recognized when any of the following conditions is detected by the channel subsystem:

1. Any of the bits defined as reserved in the TIDAW are not zero.
2. The TIDAW-transfer-in-channel flag (TTIC) in a TIDAW is one, and one or more other flag bits are also one.
3. The TTIC points to a TIDAW in which the TTIC flag is one.
4. The TIDAW count field contains zeros when the TTIC flag is zero.
5. The combination of a TIDAW data-address and count fields specify data that crosses a 4K-byte block boundary, when skipping is not in effect.
6. The count value specified in the TIDAW in which the last-TIDAW flag is one and the value for either input-count or output-count specified in a TCW did not both decrement to zero for the same byte of data transferred.
7. For a read operation only, the TIDAW in control has its data-transfer-interruption control flag set to one when data transfer was attempted by the device.
8. The TIDAW is in the TIDAW list designated by the TCCB-address field in the TCW and the count field of the TIDAW contains either a value that is zero or a value that is not evenly divisible by four.

Invalid TIDAW Address: The channel-subsystem has attempted to fetch a TIDAW from a main-storage location that is not available. An invalid TIDAW address can occur because the program has designated an invalid address in a TCW that specifies TIDAW addressing, or because the channel subsystem, on sequentially fetching TIDAWs, attempts to fetch from an unavailable location. A main-storage location is unavailable when the absolute address of the location does not correspond to a physical location. **Storage-Requests Limit Exceeded:** The use of TIDAWs is specified, and the model-dependent maximum number of storage requests required to satisfy the transfer of a block of data has been reached.

Max Data-Count Exceeded: The model-dependent maximum count of data for a single transport-control area (TCA) has been reached or exceeded for the current operation.

Device-Detected Program Check: Device-detected program checks are errors detected during a transport-mode operation by the I/O device and the channel-subsystem is requested to present a program-check condition. When a device-detected-program-check condition is recognized, additional information is available in the subchannel-extended status field, and the TSB is stored with a transport-status-area format of two. (See device-detected-program check in chapter 15.)

Protection Check

Protection check occurs when the channel subsystem attempts a storage access that is prohibited by the protection mechanism. A protection check condition is indicated as a result of a storage operation that detects any of the following errors:

1. Protection applies to the fetching of TCW.
2. Protection applies to the fetching of TCCB.
3. Protection applies to the fetching of TIDAWs.
4. Protection applies to the fetching of output data.
5. Protection applies to the storing of input data.

The subchannel key provided in the transport-mode ORB is used as the access key for storage accesses associated with an I/O operation except for the TSB store.

Detection of the protection-check condition during the fetching of a TCW, TCCB, or a TIDAW would cause the operation to be suppressed and the subchannel to be made status pending with primary, secondary, and alert status. When protection check is detected after the I/O operation has been initiated at the device, the device may be signaled to conclude the operation. Other indications may accompany the protection-check indication as a function of the operation specified by the TCW, the status received from the device, on a read operation, and the current state of the subchannel. The protection-check condition may or may not cause the execution of subsequent DCWs in a TCCB to be suppressed.

Whenever the protection-check condition pertains to the fetching of data, the failing-storage-address valid-

ity flag, bit 6 of the ERW, is one. An address of a location within the checking block for which the protection-check condition is generated is stored in the ESW failing-storage-address field.

Additional information may be available in the “Sub-channel-Extended-Status Qualifier (SESQ)” section of the subchannel-extended-status field when a protection check condition check is indicated.

Channel-Data Check

Channel-data check indicates that an uncorrected storage error has been detected in regard to data, contained in main storage, that is currently used in the performance of an I/O operation. Channel-data check is indicated when data has an invalid checking-block code (CBC) in main storage when that data is referenced by the channel subsystem.

On an input operation, when the channel subsystem attempts to store less than a complete checking block, and invalid CBC is detected on the checking block in storage, the contents of the location remain unchanged and with invalid CBC. On an output operation, whenever channel-data check is indicated, no bytes from the checking block with invalid CBC are transferred to the device.

During a storage access, the maximum number of bytes that can be transferred is model dependent. If a channel-data-check condition is recognized during that storage access, the number of bytes transferred to or from storage may not be detectable by the channel subsystem. Consequently, the number of bytes transferred to or from storage may not be correctly reflected by the residual count. When a channel-data-check condition is recognized during a storage access, the channel subsystem stores the address where the storage exception is encountered.

A condition indicated as channel-data check causes the current operation, if any, to be terminated. The subchannel becomes status pending with primary, secondary and alert status, as a function of the status received from the device. The TCW address field of the SCSW stored by TEST SUBCHANNEL pertains to the operation terminated. The extended-status-word-format bit is one, and subchannel-logout information is stored in the ESW, when TEST SUBCHANNEL is executed.

Whenever the channel-data-check condition pertains to the fetching of data, the failing-storage-address-

validity flag, bit 6 of the ERW, is one. An address of a location within the checking block for which the channel-data-check condition is generated is stored in the ESW failing-storage-address field.

Uncorrectable storage or key errors detected during data access causes the operation to be terminated. In this case the subchannel is made status pending with primary, secondary, and alert status, with channel-data check indicated, and with the ESW failing-storage address stored.

Whenever channel-data check is indicated, no measurement data for the subchannel is stored. The transport-status block may or may not be stored. Additional information may be available in the “Sub-channel-Extended-Status Qualifier (SESQ)” section of the subchannel-extended-status field.

Channel-Control Check

Channel-control check is caused by any machine malfunction affecting channel-subsystem controls. The condition may be indicated when an invalid CBC is detected on a TCW, TIDAW, TCCB, TSB, or the respective associated key.

Channel-control check may also indicate that an error has been detected in the information transferred to or from main storage during an I/O operation. However, when this condition is detected, the error has occurred inboard of the channel path: in the channel subsystem or in the path between the channel subsystem and main storage.

Detection of the channel-control-check condition causes the current operation, if any, to be terminated immediately. The subchannel is made status pending with primary and alert status or with primary, secondary, and alert status as a function of the type of termination, the current subchannel state, and the device status presented, if any. When the channel subsystem recognizes a channel-control-check condition, any previously existing intermediate interruption condition may or may not be recognized by the channel subsystem.

Whenever the channel-control-check condition pertains to an invalid CBC detected on fetching a TCW or a TCCB, or a TIDAW or storing a TSB, an extended-report word with bit 6 set to one, and the failing-storage address, are stored in the ESW when TEST SUBCHANNEL is executed.

Channel-control-check conditions encountered while fetching when the subchannel is start pending cause the operation to be terminated. In this case, the subchannel is made status pending with primary, secondary, and alert status, with channel-control check indicated, and with a failing-storage address that is stored in the ESW.

If a subchannel is halt pending and the channel subsystem encounters a channel-control-check condition while performing the halt function for that subchannel, the subchannel remains halt pending unless the channel subsystem can determine that the halt signal was issued. The subchannel remains halt pending even if the channel subsystem was attempting to issue the halt signal and is unable to determine if the halt signal was issued.

If a subchannel is start pending and the channel subsystem encounters a channel-control-check condition while performing the start function for that subchannel, the subchannel is made status pending with at least alert and primary status.

In some situations in which a channel-subsystem malfunction exists, the channel-control-check condition may be reported as a machine-check condition.

Whenever channel-control check is indicated, no measurement data for the subchannel is stored. The transport-status block may or may not be stored. Additional information may be available in the "Subchannel-Extended-Status Qualifier (SESQ)" section of the subchannel-extended-status field.

Interface-Control Check

Interface-control check indicates that an invalid signal has occurred on the channel path. The condition is detected by the channel subsystem and usually indicates malfunctioning of an I/O device. Interface-control check can occur for any of the following reasons:

1. A data or status byte received from a device has an invalid checking-block code.
2. A signal from an I/O device either did not occur or occurred at an invalid time.
3. ESW bit 26, indicating device-status check, is set to one.
4. A failed interrogate function or a recovery operation.

5. The TCW residual count did not match the residual count in the response from the device.

When the channel subsystem recognizes an interface-control-check condition, any previously existing intermediate interruption condition may or may not be recognized by the channel subsystem. The extended-status-word-format bit is one, and subchannel-logout information is stored in the ESW, when TEST SUBCHANNEL is executed.

If a subchannel is halt pending and the channel subsystem encounters an interface-control-check condition while performing the halt function for that subchannel, the subchannel remains halt pending unless the channel subsystem can determine that the halt signal was issued. The subchannel remains halt pending even if the channel subsystem was attempting to issue the halt signal and is unable to determine if the halt signal was issued.

If a subchannel is start pending and the channel subsystem encounters an interface-control-check condition while performing the start function for that subchannel, the subchannel is made status pending with at least alert and primary status.

Whenever interface-control check is indicated, no measurement data for the subchannel is stored. The transport-status block may or may not be stored. Additional information may also be available in the "Subchannel-Extended-Status Qualifier (SESQ)" section of the subchannel-extended-status field.

Channel-Subsystem Retry Failed

Channel-subsystem retry failed indicates that internal channel-subsystem conditions exist that prevent the fetching of the TCCB, a TIDAW, or data. The channel subsystem retried the fetch and the retry failed.

FCX-Status Field

The FCX-status field is contained in bit positions 16-23 of word 2 of the SCSW. This field contains channel specific status information returned by the I/O device.

TSB Valid (Bit 23): Bit 23, when set to one, indicates that additional status information for the I/O operation designated by the subchannel is available in the transport-status block. When zero, the contents of the transport-status block do not contain any addi-

tional status information for the completed I/O operation.

Subchannel-Extended-Status Field

The subchannel-extended-status field is contained in bit positions 24-31 of word 2 of the SCSW. This field may contain information used to further qualify the reason for any the following conditions, when indicated in the subchannel-status: interface-control check, channel-control check, channel-data check, program check, and protection check. If none of these bits are active in the subchannel-status byte, the fields of the subchannel-extended-status contain no meaningful information. When more than one

condition is indicated, the interface-control check takes priority over the other conditions.

Interrogate Failed (F): Bit 24 is the interrogate-failed operation bit. When set to one, the F bit indicates that an interrogate operation failed because of a program check, channel-control check, or interface-control check.

Subchannel-Extended-Status Qualifier (SESQ): Bits 25-31 are the subchannel-extended-status qualifier and contain an unsigned integer value. When the subchannel-status field indicates program check, interface-control check, protection check, data check, or channel-control check, bits 25-31 may contain additional information as follows:

Value	Description	IFCC	CCC	CDC	PGC	PTC	
0	No status available for the exception condition indicated.	X	X	X	X	X	
1	Storage-request limit exceeded, a model-dependent number of storage requests has been exceeded for the requested block of data.	-	-	-	X	-	
2	Program check when all of the following conditions are met: (1) The TCW read or write data count did not go to zero. (2) CE only or CE and DE only status was received. (3) This is not an Interrogate operation.	-	-	-	X	-	
3	Transport mode not supported by the I/O device.	-	-	-	X	-	
4	Transport mode not supported on the selected channel path.	-	-	-	X	-	
5	Reserved						
6	Program check on the TCW: An invalid TCW was detected.	-	-	-	X	-	
7	A device-detected program check condition exists due to indeterminate cause.	-	-	-	X	-	
8	A device-detected program check condition exists.	-	-	-	X	-	
9 ^a	Program check on a TIDAW: An invalid TIDAW was detected.	-	-	-	X	-	-
10-31	Reserved.	-	-	-	-	-	
32 ^b	TCW access exception: An exception was detected on while fetching a TCW.	-	X	-	X	X	
33 ^c	TSB access exception: An exception was detecting while storing a TSB.	-	X	-	X	-	
34 ^d	TCCB access exception: An exception was detected while fetching a TCCB.	-	X	-	X	X	
35 ^e	TIDAW access exception: An exception was detected while fetching a TIDAW.	-	X	-	X	X	
36 ^f	Data access exception: An exception was detected while storing or fetching data.	-	-	X	X	X	
37-63	Reserved.	-	-	-	-	-	

Figure 16-11. Subchannel-Extended Status Qualifiers

Value	Description	IFCC	CCC	CDC	PGC	PTC	
64	An invalid CBC error on read data	X	-	-	-	-	
65	Reserved	-	-	-	-	-	
66	A link protocol error condition has occurred.	X	-	-	-	-	
67	A failed device-level recovery operation.	X	-	-	-	-	
68	IFCC occurred because a device-level recovery operation failed. A program, protection or data check may also be set to one in the Subchannel status.	X	-	-	-	-	
69	The residual count between the TCW and the response from the device are not the same.	X	-	-	-	-	
70	An Invalid CBC was detected on the status portion of the transport response from the device.	X	-	-	-	-	
71	An Invalid CBC was detected on the TSB transported from the device.	X	-	-	-	-	
72-127	Reserved.	-	-	-	-	-	
Explanation: X Applicable - Not applicable IFCC Interface-control check CCC Channel-control check CDC Channel-data check PGC Program check PTC Protection check Footnotes: ^a The failing-storage-address (FSA) field in the ESW is valid and contains the address of the TIDAW which was determined to be invalid. ^b The FSA field is valid and contains the address of the current TCW. ^c The FSA field is valid and contains the address of the TSB designated by the current TCW. ^d The FSA field is valid and contains the address of the TCCB designated (directly or indirectly) by the TCW. ^e The FSA field is valid and contains the address of the TIDAW for which the exception was detected. ^f The FSA field is valid and contains the address of the input or output data designated (directly or indirectly) by the TCW. Note: If the FSA field is valid for any SESQ value other than those identified by footnotes a-f, the FSA field contains the address of the current TCW.							

Figure 16-11. Subchannel-Extended Status Qualifiers (Continued)

Extended-Status Word

The extended-status word (ESW) provides additional information to the program about the subchannel and its associated device. The ESW is placed in words 3-7 of the IRB designated by the second operand of TEST SUBCHANNEL when TEST SUBCHANNEL is executed and the subchannel designated is operational. If the subchannel is status pending or status pending with any combination of primary, secondary, intermediate, or alert status (except as noted in the next paragraph) when TEST SUBCHANNEL is executed, the ESW may have any of the following types of extended-status format:

- Format 0** Subchannel logout in word 0, an ERW in word 1, a failing-storage address or zeros in words 2 and 3, and a secondary-CCW address or zeros in word 4.
- Format 1** Zeros in bytes 0, 2, and 3 of word 0, the LPUM in byte 1 of word 0, an ERW in word 1, and zeros in words 2-4.
- Format 2** Zeros in byte 0, the LPUM in byte 1, and the device-connect time in bytes 2 and 3 of word 0; an ERW in word 1; zeros in words 2-4.
- Format 3** Zeros in byte 0, the LPUM in byte 1, and unpredictable values in bytes 2 and 3 of word 0; an ERW in word 1; zeros in words 2-4.

Words 0-4 of the ESW contain unpredictable values if any of the following conditions is met:

1. The subchannel is not status pending.
2. The subchannel is status pending alone, and the extended-status-word-format bit is zero.
3. The subchannel is status pending with intermediate status alone for other than the intermediate interruption condition due to suspension.

The type of extended-status format stored depends upon conditions existing at the subchannel at the time TEST SUBCHANNEL is executed. The conditions under which each of the types of formats is stored are described in the remainder of this section.

Extended-Status Format 0

The ESW stored by TEST SUBCHANNEL is a format-0 ESW when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, is one and the subchannel is status pending with any combination of status as defined in Figure 16-12 on page 16-51. In this case, subchannel-logout information and an ERW are stored in the extended-status word. Subchannel logout provides detailed model-independent information relating to a subchannel and describing equipment errors detected by the channel subsystem. The information is provided to aid the recovery of an I/O operation, a device, or both. Whenever subchannel logout is provided, the error conditions relate only to the subchannel reporting the error. If I/O operations involving other subchannels have been affected by the error condition, those subchannels also provide similar subchannel-logout information. An extended-report word provides additional information relating to the cause of the malfunction.

A format-0 ESW has the following format:

Word	
0	Subchannel Logout
1	Extended-Report Word
2	Failing-Storage Address or ESLD
3	
4	Secondary-CCW Address or Zeros
0	31

Subchannel Logout

The subchannel logout has the following format:

0	ESF	LPUM	R	FVF	SA	TC	DE	EA	SC
0	1	8	16	22	24	26	29	31	

Extended-Status Flags (ESF): Any of bits 1-7, when one, specifies that an error-check condition has been detected by the channel subsystem. The following indications are provided in the ESF field:

Key Check. Bit 1, when one, indicates that the channel subsystem, when accessing data, when attempting to update the measurement block, or when attempting to fetch either a CCW, an IDAW, MIDAW, TCW, TCCB, TIDAW, or when accessing a TSB, has detected an invalid checking-block code (CBC) on the associated storage key. The channel-data-check bit, bit 12 of word 2 of the

SCSW, the measurement-block data-check bit, bit 3 of word 0 of the ESW, the CCW-check bit, bit 5 of word 0 of the ESW, or the IDAW/MIDAW-check bit, bit 6 of word 0 of the ESW, identifies the source of the key error.

Note: This condition may be indicated to the program when an invalid checking-block code on a key is detected but the data, CCW, IDAW, TIDAW, or MIDAW then is not used after being prefetched. In this case, the failing-storage-address-validity bit, bit 6 of the ERW, is one, indicating that the address of a location within the storage block having the key is stored in words 2 and 3 of the ESW.

Measurement-Block Program Check. Bit 2, when one, indicates that the channel subsystem, in attempting to update the measurement block, has either detected an invalid absolute address when combining the measurement-block origin with the measurement-block index for this subchannel or has detected an invalid measurement-block address at the subchannel.

Measurement-Block Data Check. Bit 3, when one, indicates that a malfunction has been detected involving the data of the measurement block in main storage. (See “Measurement Block” on page 17-3.) Measurement-block data check is indicated when the measurement block is updated and an invalid checking-block code (CBC) is detected on the storage used to contain the measurement data or on the associated key. When invalid CBC on the associated key is detected, the key-check bit, bit 1 of the ESF field, is also stored as one.

Measurement-Block Protection Check. Bit 4, when one, indicates that the channel subsystem, when attempting to update the measurement block, has been prohibited from accessing the measurement block because the storage key does not match the measurement-block key (see “Measurement Block” on page 17-3.) The key provided by SET CHANNEL MONITOR is used for the access of storage associated with measurement-block-update operations (see “SET CHANNEL MONITOR” on page 14-13.).

Note: Whenever any of the measurement-check conditions is indicated by bits 2-4, the channel subsystem sets the subchannel measurement-block-update-enable bit to zero, disabling the storing of measurement data for the subchannel

(see “Measurement-Mode Enable (MM)” on page 15-3).

CCW Check. When a command-mode IRB is stored, bit 5, when one, indicates that an invalid CBC on the contents of the CCW or its associated key has been detected. When either of these conditions is detected, the I/O operation is terminated, the subchannel becomes status pending with primary and alert status, the extended-status-word-format bit in the SCSW is stored as one, and channel-control check is indicated in the subchannel-status field. The subchannel also becomes status pending with secondary status as a function of the type of termination or status received from the device. When invalid CBC on the associated key is detected, the key-check bit, bit 1 of the ESF field, is also stored as one.

When a transport-mode IRB is stored, bit 5 is reserved and set to zero.

Note: This condition may be indicated to the program when an invalid checking-block code on a prefetched CCW is detected but the CCW is not used. In this case, the failing-storage-address-validity bit, bit 6 of the ERW, is one, indicating that the address of a location having the invalid CBC is stored in words 2 and 3 of the ESW.

IDAW/MIDAW Check. When a command-mode IRB is stored, bit 6, when one, indicates that an invalid CBC on the contents of an IDAW or MIDAW or its associated key has been detected. When either of these conditions is detected, the I/O operation is terminated with the device, the subchannel becomes status pending with primary and alert status, the extended-status-word-format bit in the SCSW is one, and channel-control check is indicated in the subchannel-status field. The subchannel also becomes status pending with secondary status as a function of the type of termination or status received from the device. When invalid CBC on the associated key is detected, the key-check bit, bit 1 of the ESF field, is also one.

When a transport-mode IRB is stored, bit 6 is reserved and set to zero.

Note: This condition may be indicated to the program when an invalid checking-block code on the contents of a prefetched IDAW or MIDAW is detected but the IDAW or MIDAW is not used. In this case, the failing-storage-address-validity bit,

bit 6 of the ERW, is one, indicating that the address of a location having the invalid CBC is stored in words 2 and 3 of the ESW. Detection of a channel-data-check condition does not cause the CCW-check and IDAW/MIDAW-check bits to be stored as ones.

Reserved. Bit 7 is stored as zero.

Last-Path-Used Mask (LPUM): Bits 8-15 indicate the channel path that was last used for communicating or transferring information between the channel subsystem and the device. When a command-mode IRB is stored, the bit corresponding to the channel path in use is set whenever any of the following occurs:

1. The first command of a start-subchannel function is accepted by the device (see “Activity Control (AC)” on page 16-14).
2. The device and channel subsystem are actively communicating when the channel subsystem performs the suspend function for the channel program in execution.
3. The channel subsystem accepts status from the device that is recognized as an interruption condition, or a condition has been recognized that suppresses command chaining (see “Interruption Conditions” on page 16-2).
4. The channel subsystem recognizes an interface-control-check condition (see “Interface-Control Check” on page 16-29), and no subchannel-logout information is currently present at the subchannel.

When a transport-mode IRB is stored, each bit of the LPUM is stored as zero, except for the bit that corresponds to the channel path last used, whenever one of the following occurs:

1. The path has been selected for transporting the TCCB for the operation.
2. The channel subsystem accepts status from the device that is recognized as an interruption condition. If the accepted status is an interrogate response, the LPUM may be different than that stored with primary or secondary status, or both.
3. The channel subsystem recognizes an interface-control-check condition (see “Interface-Control Check” on page 16-29), and no subchannel-

logout information is currently present at the subchannel.

The LPUM field contains the most recent setting and is valid whenever the ESW contains information in one of the formats 0-3 (see “Extended-Status Word” on page 16-47) and the SCSW is stored. When subchannel-logout information is present in the ESW, a zero LPUM-field-validity flag indicates that the LPUM setting is not consistent with the other subchannel-logout indications.

Ancillary Report (R): Bit 16, when one, indicates that a malfunction of a system component has occurred that has been recognized previously or that has affected the activities of multiple subchannels. When the malfunction affects the activities of multiple subchannels, an ancillary-report condition is recognized for all of the affected subchannels except one. This bit, when zero, indicates that this malfunction of a system component has not been recognized previously. This bit is meaningful only when a channel-control check, channel-data check, or an interface-control check is indicated in bit positions 12-14 of word 2 of the SCSW.

Depending on the model, recognition of an ancillary-report condition may not be provided or it may not be provided for all system malfunctions that effect subchannel activity. When ancillary-report recognition is not provided, bit 16 is set to zero

Field-Validity Flags (FVF): Bits 17-21 indicate the validity of the information stored in the corresponding fields of either the SCSW or the extended-status word. When the validity bit is one, the corresponding field has been stored and is usable for recovery purposes. When the validity bit is zero, the corresponding field is not usable.

This bit-significant field has meaning when channel-data check, channel-control check, or interface-control check is indicated in the SCSW. When these checks are not indicated, this field, as well as the termination-code and sequence-code fields, has no meaning. Furthermore, when these checks are not indicated, the last-path-used-mask, device-status, and the TCW address or the CCW-address fields are all valid. The fields are defined as follows:

- 17 Last-path-used mask
- 18 Termination code
- 19 Sequence code
- 20 Device status

21 CCW address or TCW address

Storage-Access Code (SA): Bits 22-23 indicate the type of storage access that was being performed by the channel subsystem at the time of error. The SA field pertains only to the access of storage for the purpose of fetching or storing data during the performance of an I/O operation. This encoded field has meaning only when channel-data check, channel-control check, or interface-control check is indicated in the subchannel status. The access-code assignments are as follows:

- 00 Access type unknown
- 01 Read
- 10 Write
- 11 Read backward for a command-mode operation.

Termination Code (TC): Bits 24 and 25 indicate the type of termination that has occurred. This encoded field has meaning only when channel-data check, channel-control check, or interface-control check is indicated in the SCSW. The types of termination are as follows:

- 00 Halt signal issued
- 01 Stop, stack, or normal termination
- 10 Clear signal issued
- 11 Reserved

When at least one channel check is indicated in the SCSW but the termination-code-field-validity flag is zero, it is unpredictable which, if any, termination has been signaled to the device. If more than one channel-check condition is indicated in the SCSW, the device may have been signaled one or more termination codes that are the same or different. In this situation, if the termination-code-field-validity flag is one, the termination code indicates the most severe of the terminations signaled to the device. The termination codes, in order of increasing severity, are: stop, stack, or normal termination (01); halt signal issued (00); and clear signal issued (10)

Device-Status Check (D): When the status-verification facility is installed, bit 26, when one, indicates that the subchannel logout in the ESW resulted from the channel subsystem detecting device status that had valid CBC but that contained a combination of bits that was inappropriate when the status byte was presented to the channel subsystem. When the device-status-check bit is one, the interface-control-check status bit is set to one. If, additionally, bit 20 of the subchannel-logout field has been stored as one,

then the status byte in error has been stored in the device-status field of the SCSW. If the status-verification facility is not installed, bit 26 is stored as zero.

Secondary Error (E): Bit 27, when one, indicates that a malfunction of a system component that may or may not have been directly related to any activity involving subchannels or I/O devices has occurred. Subsequent to this occurrence, the activity related to this subchannel and the associated I/O device was affected and caused the subchannel to be set status pending with either channel-control check or interface-control check.

I/O-Error Alert (A): Bit 28, when one, indicates that subchannel logout in the ESW resulted from the signaling of I/O-error alert. The I/O-error-alert signal indicates that the control unit or device has detected a malfunction that must be reported to the channel subsystem. The channel subsystem, in response, issues a clear signal and, except as described in the next paragraph, causes interface-control check to be set and extended-status-format-0 (logout) information to be stored in the ESW.

When I/O-error alert is signaled and the subchannel has previously been set disabled or no subchannel is associated with the device, the clear signal is issued to the device, and the I/O-error-alert indication is ignored by the channel subsystem.

Sequence Code (SC): When a command-mode IRB is stored, bits 29-31 identify the I/O sequence in progress at the time of error. The sequence code pertains only to I/O operations initiated by the execution of START SUBCHANNEL or RESUME SUBCHANNEL. This encoded field has meaning only when channel-data check, channel-control check, or interface-control check is indicated in the SCSW.

When a transport-mode IRB is stored, and the validity bit corresponding to the sequence-code field is one, bits 29-31 are stored as zeros.

The sequence-code assignments are:

- 000 Reserved
- 001 A nonzero command byte has been sent by the channel subsystem, but a response has not yet been analyzed by the channel subsystem. This code is set during the initiation sequence.

- 010 The command has been accepted by the device, but no data has been transferred.
- 011 At least one byte of data has been transferred between the channel subsystem and the device. This code may be used when the channel path is in an idle or polling state.
- 100 The command in the current CCW (1) has not yet been sent to the device, (2) was sent but not accepted by the device, or (3) was sent and accepted but command-retry status was presented. This code is set when any of the following conditions occurs:
1. The command address is updated during command chaining or during the initiation of a start function or resume function at the device.
 2. During the initiation sequence, the status includes attention, control unit end, unit check, unit exception, busy, status modifier (without channel end and device end), or device end (without channel end).
 3. Command retry is signaled.
 4. The channel subsystem interrogates the device in the process of clearing an interruption condition.
 5. The channel subsystem signals the conclusion of the chain of operations to the device during command chaining while performing the suspend function.
- 101 The command in the current CCW has been accepted, but data transfer is unpredictable. This code applies from the time a device is logically connected to a channel path until the time it is determined that a new sequence code applies. This code may also be used when the channel subsystem places a channel path in the polling or idle state and it is impossible to determine that code 010 or 011 applies. It may also be used at other times when a channel path cannot distinguish between code 010 or 011.
- 110 Reserved.
- 111 Reserved.

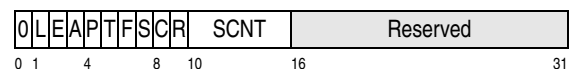
Figure 16-12 on page 16-51 defines the relationship between indications provided as subchannel-logout data and the appropriate SCSW bits.

Subchannel-Logout Condition Indicated	Logout Condition for SCSW Indication of ¹		
	CDC	CCC	IFCC
Key check	V	V	-
Measurement-block-program check ²	-	-	-
Measurement-block-data check ²	-	-	-
Measurement-block-protection check ²	-	-	-
CCW check	-	V	-
IDAW/MIDAW check	-	V	-
Last-path-used mask ³	V	V	V
Field-validity flags	V	V	V
Termination code ³	V	V	V
Device-status check	-	-	V
Secondary error	-	V	V
I/O-error alert	-	-	V
Sequence code ³	V	V	V
Explanation: - No relationship. ¹ When more than one SCSW indication is signaled, the subchannel-logout conditions that are valid are the logical OR for each of the respective SCSW indications. ² Only one measurement-block check may be indicated in a specific subchannel logout. ³ This field has a field-validity flag. CCC Channel-control check. CDC Channel-data check. IFCC Interface-control check. V Bit setting valid.			

Figure 16-12. Relationship Between Subchannel-Logout Data and SCSW Bits

Extended-Report Word

The extended-report word (ERW) provides information to the program describing specific conditions that may exist at the device, subchannel, or channel subsystem. The ERW is stored whenever the extended-status word is stored. When the extended-status-word-format bit, bit 5 of word 0 of the SCSW, and the extended-control bit, bit 14 of word 0 of the SCSW, are both zeros, the ERW contains all zeros. When the extended-status-word-format bit or the extended-control bit or both are ones, the ERW has the following format:



Request Logging Only (L) : Bit 1, when one, requests that the program only record information about the condition that caused the extended-status word to be stored.

Extended-Subchannel-Logout Pending (E): Bit 2, when one, and when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, is also one, indicates that an extended-subchannel logout is pending and that words 2-3 of the ESW contain the extended-subchannel-logout descriptor.

Bit 2 is set to zero when bit 6 is set to one.

When an ERW is stored with bits 2 and 6 both set to zero, zeros are stored in words 2 and 3 of the ESW.

Authorization Check (A): Bit 3, when one, indicates that the start or resume function was terminated because the channel subsystem has been placed in the isolated state in which pending I/O operations are not initiated and I/O operations currently being performed either are in the process of being terminated or have been terminated.

Path Verification Required (P): Bit 4, when one, indicates that the program must verify the identity of the device. The LPUM, when valid, indicates the channel path for which device verification is to be performed. When a valid LPUM is not available, the identity of the device must be verified for each available channel path.

Channel-Path Timeout (T): Bit 5, when one, indicates that, during a signaling sequence, an appropriate signal from the device did not occur within a predetermined time interval. Bit 5 is meaningful when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, and the interface-control-check bit, bit 14 of word 2 of the SCSW, are both ones.

Failing-Storage-Address Validity (F): Bit 6, when one, and when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, is also one, indicates that the channel subsystem has detected an invalid CBC on a CCW, a data location, an IDAW, a MIDAW, a TCW, TCCB, a TIDAW, a TSB or on the respective associated key and has stored, in words 2 and 3 of the ESW, the absolute address of a location associated with the invalid CBC.

Bit 6 is set to zero when bit 2 is set to one.

When an ERW is stored with bits 2 and 6 both set to zero, zeros are stored in words 2 and 3 of the ESW.

Concurrent Sense (S) : When a command-mode IRB is stored, bit 7, when one, indicates that the concurrent-sense facility has placed sense information accepted from the device in the extended-control word and has stored a value, in bit positions 10-15 of the ERW, that specifies the number of sense bytes that have been stored in the extended-control word. When bit 7 is one, bit 14 of word 0 of the SCSW is also one.

When a transport-mode IRB is stored, bit 7 is reserved and set to zero.

Concurrent-Sense Count (SCNT): When bit 7 is one, bit positions 10-15 contain a value, in the range 1-32, that specifies the number of sense bytes stored into the extended-control word by the concurrent-sense facility. When bit 7 is zero, bit positions 10-15 contain zeros.

Secondary-CCW-Address Validity (C): When a command-mode IRB is stored, bit 8, when one, and when the extended-status-word-format bit, bit 5 of word 0 of the SCSW, is also one, indicates that the channel subsystem has detected an error condition that precludes the continued performance of an I/O operation. When prefetching applies (bit 9 of word 1 of the ORB is one) and certain error conditions identified by channel-control check, channel-data check, or interface-control check are recognized by the channel subsystem, situations may exist where the termination point of execution of the channel program differs between the channel subsystem and the I/O device. To properly identify the termination points, bit 8 is set to one, and a second CCW address (secondary-CCW address) is provided in the ESW and designates the last CCW executed at the device. When the validity bit is zero for the previously mentioned errors, the channel subsystem was unable to determine the termination point of the control-unit execution, and the secondary-CCW-address field contains zeros.

Bit 8 is not set to one unless the program has permitted prefetching by setting the prefetch-control bit, bit 9 of word 1 of the ORB, to one for the channel program in execution.

When a transport-mode IRB is stored, bit 8 is reserved and set to zero.

Failing-Storage-Address Format (R): Bit 9 indicates the format of the failing-storage address when the failing-storage-address validity bit, bit 6 of the ERW, is one. When bit 6 is zero, bit 9 is not meaningful and is stored as zero. When bit 6 is one and bit 9 is zero, a format-1 failing-storage address is stored in words 2 and 3 of the ESW. When bit 6 is one and bit 9 is one, a format-2 failing-storage address is stored in words 2 and 3. See “Failing-Storage Address” below for a description of format-1 and format-2 addresses.

Failing-Storage Address

Words 2 and 3 of the extended-status contain a 24-, 31-, or 64-bit absolute address. When the failing-storage-address-validity flag, bit 6 of the ERW, is one, words 2 and 3 contain either a format-1 failing-storage address or a format-2 failing-storage address. When bit 6 is one, the failing-storage-address field designates a byte location within the invalid checking block associated with an invalid CBC for a CCW, data location, IDAW, MIDAW, TCW, TCCB, TIDAW, TSB, or their respective associated key.

When a transport-mode IRB is stored, and the failing-storage-address validity flag, bit 6 of the ERW, is one, the failing-storage address contains the absolute address associated with the reported failing condition.

When an error condition is detected and a command-mode IRB is stored, the form of the address stored in words 2 and 3 depends on the format-2-IDAW control, bit 14 of word 1 of the ORB, and on the MIDA control, bit 25 of word 1 of the ORB, as follows:

1. When the format-2-IDAW control is zero, specifying that fullword IDAWs containing 31-bit addresses are used, and the modified-CCW-indirect-data-addressing-control is zero, a format-1 address is stored, and the failing-storage-address-format bit, bit 9 of the ERW, is stored as zero.
2. When either the format-2-IDAW control is one, specifying that doubleword IDAWs containing 64-bit addresses are used, or the modified-CCW-indirect-data-addressing-control is one, or both are one, a format-2 address is stored, and the failing-storage-address-format bit is stored as one.

When an error condition is detected and a transport-mode IRB is stored, a format-2 failing-storage address is stored. A format-2 failing-storage address is stored for a transport-mode IRB when a program-check, protection check, channel-data check or channel-control check condition is indicated by the subchannel status.

When a format-1 address is stored, bits 1-31 of word 2 form the address associated with the reported error condition, and bit 0 of word 2 and all of word 3 are stored as zeros. When a format-2 address is stored, bits 0-31 of word 2 followed by bits 0-31 of word 3 form the 64-bit address associated with the reported error condition.

Extended-Subchannel-Logout Descriptor (ESLD)

When the extended-status-word-format (L) bit, bit 5 of SCSW word 0, is one and the extended-subchannel-logout pending (E) flag, bit 2 of the ERW, is one, an extended-subchannel-logout descriptor is provided in words 2-3 of the ESW. The ESLD identifies the logout and the channel path for which the extended-subchannel logout is pending.

Secondary-CCW Address

When the subchannel-status field indicates channel-control check, channel-data check, or interface-control check and the secondary-CCW-address-validity flag, bit 8 of word 1, is one, bits 1-31 of word 4 form an absolute address of the last CCW executed by the I/O device at the point the reported check condition caused channel-program termination. When provided, the secondary-CCW address may be used for recovery purposes. When the secondary-CCW-address-validity flag is zero, this field contains zeros.

When a transport-mode IRB is stored and the subchannel-status field indicates channel-control check, channel-data check, or interface-control check, word 4 is reserved and contains zeros.

Extended-Status Format 1

The ESW stored by TEST SUBCHANNEL is a format-1 ESW when all of the following conditions are met:

1. The extended-status-word-format bit, bit 5 of word 0 of the SCSW, is zero.

2. The subchannel status-control field has the status-pending bit, bit 31 of word 0 of the SCSW, set to one, together with:
 - a. The primary-status bit, bit 29 of word 0 of the SCSW, alone,
 - b. The primary-status bit and other status-control bits, or
 - c. The intermediate-status bit, bit 28 of word 0 of the SCSW, and the suspended bit, bit 26 of word 0 of the SCSW.
3. At least one of the following conditions is indicated:
 - a. The device-connect-time-measurement mode is inactive.
 - b. The channel-subsystem-timing facility is not available for the subchannel.
 - c. The subchannel is not enabled for the device-connect-time-measurement mode.

Zeros are stored in bytes 0, 2, and 3 of word 0, and the LPUM is stored in byte 1 of word 0; an ERW is stored in word 1; zeros are stored in words 2-4.

The device-connect-time-measurement mode is made inactive when SET CHANNEL MONITOR is executed and bit 31 of general register 1 is zero.

A format-1 ESW has the following format:

Word				
0	Zeros	LPUM	Zeros	
1	Extended-Report Word			
2	Zeros			
3				
4				
	0	8	16	31

Last-Path-Used Mask (LPUM): For a definition of the LPUM, see “Last-Path-Used Mask (LPUM)” on page 16-49.

Extended-Report Word (ERW): For a definition of the ERW, see “Extended-Report Word” on page 16-51.

Extended-Status Format 2

The ESW stored by TEST SUBCHANNEL is a format-2 ESW when all of the following conditions are met:

1. The extended-status-word-format bit, bit 5 of word 0 of the SCSW, is zero.
2. The channel-subsystem-timing facility is available for the subchannel.
3. The subchannel is enabled for the device-connect-time-measurement mode.
4. The device-connect-time-measurement mode is active.
5. The subchannel status-control field has the status-pending bit, bit 31 of word 0 of the SCSW, set to one, together with:
 - a. The primary-status bit, bit 29 of word 0 of the SCSW, alone,
 - b. The primary-status bit and other status-control bits, or
 - c. The intermediate-status bit, bit 28 of word 0 of the SCSW, and the suspended bit, bit 26 of word 0 of the SCSW.

Zeros are stored in byte 0 of word 0, the LPUM is stored in byte 1 of word 0, and the device-connect time is stored in bytes 2 and 3 of word 0; an ERW is stored in word 1; zeros are stored in words 2-4.

A format-2 ESW has the following format:

Word				
0	Zeros	LPUM	DCTI	
1	Extended-Report Word			
2	Zeros			
3				
4				
	0	8	16	31

Last-Path-Used Mask (LPUM): For a definition of the LPUM, see “Last-Path-Used Mask (LPUM)” on page 16-49.

Device-Connect-Time Interval (DCTI): Bit positions 16-31 contain the binary count of time increments accumulated by the channel subsystem during

the time that the channel subsystem and the device were actively communicating and the subchannel was subchannel active for a command-mode operation, or start pending for a transport-mode operation. The time increment of the DCTI is 128 microseconds.

If the above conditions for the storing of the DCTI value in the ESW are met but the device-connect-time-measurement mode was made active by SET CHANNEL MONITOR subsequent to the execution of START SUBCHANNEL for this subchannel, the DCTI value stored is greater than or equal to zero and less than or equal to the correct DCTI value.

Note: The DCTI value stored in the ESW is the same as that used to update the corresponding measurement-block data for the subchannel if the measurement-block-update mode is in use for the subchannel. If the measurement-block-update mode for the channel subsystem is active and the subchannel is enabled for the device-connect-time-measurement mode but no DCTI value is stored in the ESW (because of the presence of subchannel-logout information), or if the DCTI is zero, then nothing is added to the corresponding measurement-block data.

Extended-Report Word (ERW): For a definition of the ERW, see “Extended-Report Word” on page 16-51.

Extended-Status Format 3

The ESW stored by TEST SUBCHANNEL is a format-3 ESW when the extended-status-word-format

bit, bit 5 of word 0 of the SCSW, is zero and the subchannel is status pending with (1) secondary status, alert status, or both when primary status is not also present, or (2) intermediate status when the subchannel is not suspended. Zeros are stored in byte 0 of word 0, and the LPUM is stored in byte 1 of word 0. Bytes 2 and 3 of word 0 contain unpredictable values; an ERW is stored in word 1; zeros are stored in words 2-4.

A format-3 ESW has the following format:

Word				
0	Zeros	LPUM	XXXXXXXXXXXXXXXXXXXX	
1	Extended-Report Word			
2	Zeros			
3				
4				
	0	8	16	31

Last-Path-Used Mask (LPUM): For a definition of the LPUM, see “Last-Path-Used Mask (LPUM)” on page 16-49.

An “X” in the format indicates the bit may be zero or one.

Extended-Report Word (ERW): For a definition of the ERW, see “Extended-Report Word” on page 16-51.

Figure 16-13 on page 16-56 summarizes the conditions at the subchannel under which each type of information is stored in the ESW.

Subchannel Conditions When IRB is stored						Extended-Status Word (ESW), Word 0	
Subchannel-Status Word			Path-Management-Control Word				
Status-Control Field	L Bit	Suspended Bit	Device Connect-Time-Msrmt Mode	Timing-Facility Bit	Device-Connect-Time-Msrmt-Mode-Enable Bit	Contents	Bytes 0,1,2,3
AIP SX							
----	0						
00001	0					U	----
01001	0	0					
		1	Inactive				
			Active	0		1	ZMZZ
				1		2	ZMDD
**1*1	0	////	Inactive				
		////	Active	0		1	ZMZZ
		////		1			
		////			1	2	ZMDD
**011	0						
1*001	0					3	ZM**
****1	1					0	RRRR

Explanation:

- Defined to be not meaningful when X is zero.

* Bits may be zeros or ones.

/ Information not relevant in this situation.

A Alert status.

D Accumulated device-connect-time-interval (DCTI) value stored in bytes 2 and 3.

I Intermediate status.

L Extended-status-word format.

M Last-path-used mask (LPUM) stored in byte 1.

P Primary status.

R Subchannel-logout information stored in word 0.

S Secondary status.

U No format defined.

X Status pending.

Z Bits are stored as zeros.

Figure 16-13. Information Stored in ESW

Extended-Control Word

The extended-control word, which is words 8-15 of an interruption-response block (see “Interruption-Response Block” on page 16-6), provides additional information to the program describing conditions that may exist at the channel subsystem, subchannel, or device. The extended-control (E) bit, bit 14 of word 0

of the SCSW, when one, indicates that model-dependent information or concurrent-sense information has been stored in the extended-control word.

The information provided in the extended-control word is as follows:

SCSW Bits 5 14	ERW Bit 7	ERW Bits 10-15	ECW Words 0-7
0 0	0	Zeros	Unpredictable ¹
0 1	0	(⁵)	(⁵)
0 1	1	(³)	Concurrent-sense information ⁴
1 0	0	Zeros	Unpredictable ¹
1 1	0	Zeros	Model-dependent information ²
1 1	1	(³)	Concurrent-sense information ⁴

Explanation:

¹ If stored, the value of these words is unpredictable.

² Unused bits in the model-dependent information are stored as zeros.

³ Bit positions 10-15 contain a value equal to the number of sense bytes returned.

⁴ Unused bytes in the concurrent-sense information are stored as zeros.

⁵ The combination of SCSW bit 5 as 0, SCSW bit 14 as one, and ERW bit 7 as zero does not occur.

Extended-Measurement Word

The extended-measurement word (EMW) provides I/O measurement information to the program for the most recent start or resume operation performed at the subchannel. When the extended-I/O-measurement-word facility is installed and enabled, the EMW is conditionally stored in words 16-23 of the IRB designated by the second operand of TEST SUBCHANNEL when TEST SUBCHANNEL is executed. The EMW is stored by TEST SUBCHANNEL when all of the following conditions are met:

1. The extended-status-word-format (L) bit (bit 5, word 0 of the SCSW) is zero.
2. The channel-subsystem-timing facility is available for the subchannel, as indicated by the timing facility bit (T) at the subchannel.
3. The extended-I/O-measurement-word-mode enable bit (X) at the subchannel is one.

4. The subchannel status-control field has the status-pending bit (bit 31, word 0 of the SCSW) set to one, together with:
 - a. The primary-status bit (bit 29, word 0 of the SCSW) alone, or
 - b. The primary-status bit and other status-control bits, or
 - c. The secondary-status bit (bit 29, word 0 of the SCSW) alone, or
 - d. The intermediate-status bit (bit 28, word 0 of the SCSW) and the suspended bit (bit 26, word 0 of the SCSW).

Words 16-23 of the IRB contain unpredictable values when the extended-I/O-measurement-word facility is installed and enabled but the conditions listed above are not met.

When the extended-I/O-measurement-word facility is not installed, or the facility is installed but not enabled, words 16-23 of the IRB are not accessed by the channel subsystem.

The format of the extended-measurement word is shown below:

Word	
0	Device-Connect Time
1	Function-Pending Time
2	Device-Disconnect Time
3	Control-Unit-Queuing Time
4	Device-Active-Only Time
5	Device-Busy Time
6	Initial Command Response Time
7	Reserved
0	31

Each field in the EMW, when valid, contains a 32 bit binary count in which each increment of the count represents a value of 0.5 microseconds. A value of 00000000 hex represents a time period of zero seconds; the maximum value of FFFFFFFF hex represents approximately 35.79 minutes.

The accuracy of each of the measurement fields stored by the measurement facility is undefined and may vary depending on the resolution of timers implemented at the channel subsystem, the types of channels used to perform the operation, the capabilities

of control units accessed during the operation, and the overall length of the I/O operation that was performed. The maximum value of FFFFFFFF hex is stored if a counter overflows; the program is not alerted when an overflow occurs.

Device-Connect Time: Bit positions 0-31 of word 0 contain the measured device-connect time for the operation. The device-connect time is the sum of the time intervals measured whenever the device is logically connected to a channel path while the subchannel is subchannel active for a command-mode operation, or start-pending for a transport-mode operation, and the device is actively communicating with the channel path, as defined in the section "Device-Connect Time" on page 17-4.

Function-Pending Time: Bit positions 0-31 of word 1 contain the SSCH- or RSCH-function-pending time for the operation. Function-pending time is the time interval between acceptance of the start function (or resume function if the subchannel is in the suspended state) at the subchannel and acceptance of the first command associated with the initiation or resumption of channel-program execution at the device, as defined in the section "Function-Pending Time" on page 17-5.

Device-Disconnect Time: Bit positions 0-31 of word 2 contain the device-disconnect time for the operation. Device-disconnect time is the sum of the time intervals measured whenever the device is logically disconnected from the channel subsystem while the subchannel is subchannel-active for a command-mode operation, or start-pending for a transport-mode operation, as defined in the section "Device-Disconnect Time" on page 17-5.

Control-Unit-Queuing Time : Bit positions 0-31 of word 3 contain the control-unit-queuing time for the operation. Control-unit-queuing time is the sum of the time intervals measured by the control unit whenever the device is logically disconnected from the channel subsystem during an I/O operation while the device is busy with an operation initiated from a different system, as defined in the section "Control-Unit-Queuing Time" on page 17-5.

Device-active-only time: Bit positions 0-31 of word 4 contain the device-active-only time for the operation. For a command-mode operation, device-active-only time is the sum of the time intervals when the subchannel is device-active but not subchannel-active at the end of an I/O operation or chain of I/O

operations initiated by a start function or resume function, as defined in section “Device-Active-Only Time” on page 17-6. When a transport-mode IRB is stored, device-active-only time is the sum of the times that the I/O device was active executing device command and the time interval between when channel-end and device-end status were presented.

Device-busy time: Bit positions 0-31 of word 5 contain the device-busy time for the operation. When a command-mode IRB is stored, the device-busy time is the sum of the time intervals when the device is found to be device busy during an attempt to initiate a start function or resume function at the sub-channel, as defined in the section “Device-Busy Time” on page 17-6. When a transport-mode IRB is stored, device-busy time is the sum of the times that the I/O device was busy attempting to initiate a command and the device busy time accumulated by the channel subsystem.

Initial Command Response Time: Bit positions 0-31 of word 6 contain the initial-command-response time for the operation. When a command-mode IRB is stored, initial-command-response time for an operation is the time interval beginning from when the first command of the channel program is sent to the device until the device indicates it has accepted the command. When a transport-mode IRB is stored, the initial-command-response time, is the time interval difference between when a TCCB is sent to the I/O device until the response is received from the I/O device, minus the time from when the control unit receives the TCCB until the channel subsystem receives the response from the control unit.

Chapter 17. I/O Support Functions

Channel-Subsystem Monitoring	17-1	Signals	17-11
Channel-Subsystem Timing	17-2	Halt Signal	17-11
Channel-Subsystem Timer	17-2	Clear Signal	17-11
Measurement-Block Update	17-2	Reset Signal	17-12
Measurement Block	17-3	Resets	17-12
Measurement-Block Format	17-7	Channel-Path Reset	17-12
Measurement-Block Origin	17-7	I/O-System Reset	17-13
Measurement-Block Address	17-7	Externally Initiated Functions	17-16
Measurement-Block Key	17-7	Initial Program Loading	17-16
Measurement-Block Index	17-7	CCW-type IPL	17-17
Measurement-Block-Update Mode	17-7	List-Directed IPL	17-19
Measurement-Block-Format Control	17-8	Reconfiguration of the I/O System	17-22
Measurement-Block-Update Enable	17-8	Status Verification	17-22
Control-Unit-Queuing Measurement	17-8	Address-Limit Checking	17-22
Control-Unit-Defer Time	17-8	Configuration Alert	17-23
Device-Active-Only Measurement	17-9	Incorrect-Length-Indication Suppression	17-23
Initial-Command-Response Measurement	17-9	Concurrent Sense	17-24
Time-Interval-Measurement Accuracy	17-9	Channel-Subsystem Recovery	17-24
Device-Connect-Time Measurement	17-9	Channel Report	17-25
Device-Connect-Time-Measurement		Channel-Report Word	17-26
Mode	17-10	Restore-Subchannel Facility	17-28
Device-Connect-Time-Measurement		Extended-Subchannel-Logout Facility	17-28
Enable	17-10	Channel-Subsystem-I/O-Priority Facility	17-29
Extended Measurement Word	17-10	Number of Channel-Subsystem-Priority	
Extended-Measurement-Word Enable	17-10	Levels	17-29
Signals and Resets	17-11	Multiple-Subchannel-Set Facility	17-29

The I/O support functions are those functions of the channel subsystem that are not directly related to the initiation or control of I/O operations. The following I/O support functions are described in this chapter:

- Channel-subsystem monitoring
- Signals and resets
- Externally initiated functions
- Status verification
- Address-limit checking
- Configuration alert
- Incorrect-length-indication suppression
- Concurrent sense
- Channel-subsystem recovery
- I/O-priority facility
- Restore-subchannel facility
- Multiple-subchannel-set facility

Channel-Subsystem Monitoring

Monitoring facilities are provided in the channel subsystem so that the program can retrieve measured values on performance for a designated subchannel. The use of these facilities is under program control by means of the execution of the SET CHANNEL MONITOR instruction and the MODIFY SUBCHANNEL instruction.

The principal components of the channel-subsystem-monitoring facilities are the channel-subsystem-timing facility, measurement-block-update facility, and device-connect-time-measurement facility. The measurement-block-update facility and device-connect-time-measurement facility both use the channel-subsystem-timing facility but otherwise are logically distinct and operate independent of one another.

Other components of the channel-subsystem-monitoring facilities are the control-unit-queuing-measurement facility, the control-unit-defer-time facility, the device-active-only-measurement facility, the initial-command-response-measurement facility, the extended-I/O-measurement-block facility, and the extended-I/O-measurement-word facility. These enhance the measurements of the measurement-block-update facility if they are available as described in later sections, where each of the facilities that constitute the channel-subsystem-monitoring facilities is described in this chapter.

Channel-Subsystem Timing

The channel-subsystem-timing facility provides the channel subsystem with the capability of measuring the elapsed time required for performing several different phases of the processing of a start function initiated by START SUBCHANNEL. These elapsed-time measurements are used by both the measurement-block-update facility and the device-connect-time-measurement facility to provide subchannel performance information to the program.

While every channel subsystem has a channel-subsystem-timing facility, it may or may not be provided for use with all subchannels. Subchannels for which the facility is provided have the timing-facility bit, bit 14 of word 1, stored as one in the associated subchannel-information block. (See “Timing Facility (T)” on page 15-4.) If the channel-subsystem-timing facility is not provided for the subchannel, the subchannel's timing-facility bit is stored as zero.

Subchannels that do not have the channel-subsystem-timing facility provided are those for which the characteristics of the associated device, the manner in which it is attached to the channel subsystem, or the channel-subsystem resources required to support the device are such that use of the channel-subsystem-timing facility is precluded.

The channel-subsystem-timing facility consists of at least one channel-subsystem timer and the associated logic and storage required for computing and recording the elapsed-time intervals for use by the two measurement facilities. The aspects of the channel-subsystem-timing facility that are of importance to the program are described below.

Channel-Subsystem Timer

Each channel-subsystem timer is a binary counter that is not accessible to the program. The channel-subsystem timer provides a minimum timer resolution of 128 microseconds. A timer resolution of 1.0 microseconds is provided when FICON-I/O-interface channel paths are supported. When incrementing the channel-subsystem timer causes a carry out of the leftmost bit position, the carry is ignored, and counting continues from zero. No indications are generated as a result of the overflow.

Just as every CPU has access to a TOD clock, every channel subsystem has access to at least one channel-subsystem timer. When multiple channel-subsystem timers are provided, synchronization among these timers is also provided, creating the effect that all the timing facilities of the channel subsystem share a single timer. Synchronization among these timers may be supplied either through some TOD clock or independently by the channel subsystem.

If the TOD clocks are not synchronized, the elapsed times measured by the channel-subsystem-timing facility may have unpredictable values for some or all of the subchannels, depending on the model and on the particular channel-subsystem timer and the way the associated devices are physically attached to the system. The values are unpredictable for those devices attached to the system by separately configurable channel paths whose associated CPU TOD clocks are not synchronized.

Synchronization: If either the measurement-block-update mode or the device-connect-time-measurement mode is active and any of the channel-subsystem timers is found to be out of synchronization, a channel-subsystem-timer-sync check is recognized, and a channel report is generated to alert the program (see “Channel-Subsystem Recovery” on page 17-24). If neither of these modes is active, the lack of synchronization is not recognized.

Measurement-Block Update

The measurement-block-update facility provides the program with the capability of accumulating performance information for subchannels that are enabled for the measurement-block-update mode when the measurement-block-update mode is active. A subchannel is enabled for the measurement-block-update mode by setting bit 11 of word 1 of the SCHIB operand to one and then issuing MODIFY SUB-

CHANNEL. The measurement-block-update mode is made active by the execution of SET CHANNEL MONITOR when bit 62 of general register 1 is one.

When the measurement-block-update mode is active and the subchannel is enabled for the measurement-block-update mode, information is accumulated in a measurement block associated with the subchannel. A measurement block is either a 32-byte area (format-0 measurement block) or a 64-byte area (format-1 measurement block) in main storage that is associated with a subchannel for the purpose of accumulating measurement data.

For format-0 measurement blocks, the program specifies a contiguous area of absolute storage, referred to as the measurement-block area, and subdivides this area into 32-byte blocks, one block for each subchannel for which measurement data is to be accumulated. The measurement-block-update facility uses the measurement-block index contained at the subchannel in conjunction with the measurement-block origin established by the execution of SET CHANNEL MONITOR to compute the absolute address of the measurement block associated with a subchannel.

For format-1 measurement blocks, the program provides a 64-byte contiguous area of absolute storage for the subchannel. The measurement-block-update facility uses the measurement-block address provided by the MSCH instruction to access the measurement block.

Measurement data is stored in the measurement block associated with the subchannel each time an I/O operation or chain of I/O operations initiated by a START SUBCHANNEL instruction or RESUME SUBCHANNEL instruction is suspended or is completed at the device. The completion of an I/O operation or chain of I/O operations at the device is normally determined when secondary status is accepted from the device.

The measurement data accumulated in the format-0 and format-1 measurement blocks by the measurement-block-update facility is described in the following section, "Measurement Block".

Measurement Block

A measurement block is either a 32-byte area (format-0 measurement block) at a location designated by the program by its use of a measurement-block-

origin address in conjunction with the measurement-block index, or a 64-byte area (format-1 measurement block) at a location designated by the measurement-block address provided in the SCHIB during the execution of the MODIFY SUBCHANNEL instruction. The measurement block contains the accumulated values of the measurement data described below. When the measurement-block-update mode is active and the subchannel is enabled for measurement-block update, the measurement-block-update facility accumulates the values for the measurement data that accrue during the performance of an I/O operation or chain of I/O operations initiated by START SUBCHANNEL.

When the I/O operation or chain of I/O operations is suspended or completed and no error condition is encountered, the accrued values are added to the accumulated values in the measurement block for that subchannel. If an error condition is detected and subchannel-logout information is stored in the extended-status word (ESW), the accrued values are not added to the accumulated values in the measurement block for the subchannel, and the two count fields in the measurement block are not incremented.

If (1) any of the accrued time values is detected to exceed the internal storage provided for containing these values, (2) the control unit cannot provide an accurate queuing time or defer time for the current operation, or (3) the channel subsystem successfully recovers from certain error conditions, none of the accrued values is added to the measurement block for the subchannel, and the sample count in the measurement block is not incremented, but the SSCH+RSCH count in the block is incremented.

References to the measurement block by the measurement-block-update facility, in order to accumulate measurement data at the suspension or completion of an I/O function, are single-access references and appear to be word concurrent as observed by CPUs.

The measurement-block-update facility updates all fields in the measurement block that are required to be updated for a suspended I/O operation prior to putting the subchannel into the suspended state. The measurement-block-update facility updates all fields in the measurement block that are required to be updated for a completed I/O operation prior to making the subchannel status pending with secondary status or, if the subchannel is start pending for a subsequent operation, prior to initiating the start function.

A format-0 measurement block is stored when the measurement-block-format-control bit at the sub-channel is zero; a format-1 measurement block is stored when the measurement-block-format-control bit at the subchannel is one.

The format-0 measurement block has the following format:

Word

0	SSCH+RSCH Count	Sample Count
1	Device-Connect Time	
2	Function-Pending Time	
3	Device-Disconnect Time	
4	Control-Unit-Queuing Time	
5	Device-Active-Only Time	
6	Device-Busy Time	
7	Initial Command Response Time	

01631

The format-1 measurement block has the following format:

Word

0	SSCH+RSCH Count
1	Sample Count
2	Device-Connect Time
3	Function-Pending Time
4	Device-Disconnect Time
5	Control-Unit-Queuing Time
6	Device-Active-Only Time
7	Device-Busy Time
8	Initial Command Response Time
9	Reserved
15	

031

SSCH+RSCH Count: Bits 0-15 of word 0 in the format-0 measurement block and bits 0-31 of word 0 in the format-1 measurement block are used as a binary counter. During the performance of a start function for which measurement-block update is active, when (1) the secondary status condition is recognized or (2) the suspend function is performed,

the counter is incremented by one, and the measurement data is stored. The counter wraps around from the maximum value to 0. The program is not alerted when counter overflow occurs.

If the measurement-block-update mode is active and the subchannel is enabled for measuring, the SSCH+RSCH count is incremented even when the lack of measured values for an individual start function precludes the updating of the remaining fields of the measurement block or when the timing-facility bit for the subchannel is zero. The SSCH+RSCH count is not incremented if the measurement-block-update mode is inactive, if the subchannel is not enabled for the measurement-block update, or if subchannel-logout information has been generated for the start function.

Sample Count: Bits 16-31 of word 0 in the format-0 measurement block and bits 0-31 of word 1 in the format-1 measurement block are used as a binary counter. When the time-accumulation fields following word 0 of the measurement block are updated, the counter is incremented by one. On some models, certain conditions may prevent the measurement-block-update facility from obtaining the accrued values of the measurement data for an individual start function, even when the measurement-block-update mode is active and the subchannel is enabled for that mode. The control unit may also signal that it was not able to accumulate an accurate queuing time. In these situations, the sample-count field is not incremented.

The counter wraps around from the maximum value to 0. The program is not alerted when counter overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the sub-channel.

The System Library publication for the system model specifies the conditions, if any, that preclude the updating of the sample count and time-accumulation fields of the measurement block.

Device-Connect Time: Bit positions 0-31 of word 1 in the format-0 measurement block and bit positions 0-31 of word 2 in the format-1 measurement block contain the accumulation of measured device-connect-time intervals. The device-connect-time interval (DCTI) is the sum of the time intervals measured whenever the device is logically connected to a channel path while the subchannel is subchannel active and the device is actively communicating with the

channel path. The device-connect time does not include the intervals when a device is logically connected to a channel path but is not actively communicating with the channel. The device reports the accumulation of time intervals when the device is logically connected but not actively communicating with the channel path as control-unit-defer time. The control-unit-defer time is not included in the device-connect-time measurement but, instead, is added to the accrued device-disconnect-time measurement for the operation.

The time intervals are stored using a resolution of 128 microseconds. The accumulated value is modulo approximately 152.71 hours, and the program is not alerted when an overflow occurs. This field is not updated if (1) the channel-subsystem-timing facility is not provided for the subchannel, (2) the measurement-block-update mode is inactive, or (3) any of the time values accrued for the current start function has been detected to exceed the internal storage in which it was accrued.

Accumulation of device-connect-time intervals for a subchannel and storing this data in the ESW are not affected by whether the measurement-block-update mode is active. (See “Device-Connect-Time Measurement” on page 17-9.)

Function-Pending Time: Bit positions 0-31 of word 2 in the format-0 measurement block and bit positions 0-31 of word 3 in the format-1 measurement block contain the accumulated SSCH- and RSCH-function-pending time. Function-pending time is the time interval between acceptance of the start function (or resume function if the subchannel is in the suspended state) at the subchannel and acceptance of the first command associated with the initiation or resumption of channel-program execution at the device.

When channel-program execution is suspended because of a suspend flag in the first CCW of a channel program, the suspension occurs prior to transferring the first command to the device. In this case, the function-pending time accumulated up to that point is added to the value in the function-pending-time field of the measurement block. Function-pending time is not accrued while the subchannel is suspended. Function-pending time begins to be accrued again, in this case, when RESUME SUBCHANNEL is subsequently executed while the designated subchannel is in the suspended state.

The function-pending-time interval is stored using a resolution of 128 microseconds. The accumulated value is modulo approximately 152.71 hours, and the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

Device-Disconnect Time: Bit positions 0-31 of word 3 in the format-0 measurement block and bit positions 0-31 of word 4 in the format-1 measurement block contain the accumulated device-disconnect time. Device-disconnect time is the sum of the time intervals measured whenever the device is logically disconnected from the channel subsystem while the subchannel is subchannel active. The device-disconnect time also includes the sum of control-unit-defer-time intervals reported by the device during the I/O operation.

Device-disconnect time is not accrued while the subchannel is in the suspended state. Device-disconnect time begins to be accrued again, in this case, on the first device disconnection after channel-program execution has been resumed at the device (the subchannel is again subchannel active).

The device-disconnect-time interval is stored using a resolution of 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

The device-disconnect time does not include the interval between the primary status condition and the secondary status condition at the end of an I/O operation when the subchannel is no longer subchannel active but the I/O device is active. If the channel subsystem provides the device-active-only measurement facility, this time is accumulated in the device-active-only-time field of the measurement block.

Control-Unit-Queuing Time: Bit positions 0-31 of word 4 in the format-0 measurement block and bit positions 0-31 of word 5 in the format-1 measurement block contain the accumulated control-unit-queuing time. Control-unit-queuing time is the sum of the time intervals measured by the control unit whenever the device is logically disconnected from the channel subsystem during an I/O operation while the device is busy with an operation initiated from a different system.

Control-unit-queuing time is not accrued while the subchannel is in the suspended state. Control-unit-queuing time may be accrued for the channel program after the subchannel becomes subchannel active following a successful resumption.

The control-unit-queuing-time field is updated such that bit 31 represents 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel or if the control unit does not provide a queuing time.

Device-Active-Only Time: Bit positions 0-31 of word 5 in the format-0 measurement block and bit positions 0-31 of word 6 in the format-1 measurement block contain the accumulated device-active-only time. Device-active-only time is the sum of the time intervals when the subchannel is device active but not subchannel active at the end of an I/O operation or chain of I/O operations initiated by a start function or resume function.

Device-active-only time is not accumulated when the subchannel is device active during periods when the subchannel is active; such time is accumulated as device-connect time or device-disconnect time, as appropriate.

The device-active-only-time field is updated such that bit 31 represents 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

Device-Busy Time: When the extended-I/O-measurement-block facility is installed, bit positions 0-31 of word 6 in the format-0 measurement block, and bit positions 0-31 of word 7 in the format-1 measurement block contain the accumulated device-busy time. Device-busy time is the sum of the time intervals when the subchannel is device busy during an attempt to initiate a start function or resume function at the subchannel.

The device-busy-time field is updated such that bit 31 represents 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs. This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel.

Initial-Command-Response Time: Bit positions 0-31 of word 7 in the format-0 measurement block, and bit positions 0-31 of word 8 in the format-1 measurement block contain the accumulated initial-command-response time for the subchannel. The initial-command-response time for a start or resume function is the time interval beginning from when the first command of the channel program is sent to the device until the device indicates it has accepted the command.

The initial-command-response time is stored at a resolution of 128 microseconds. The accumulated value is modulo approximately 152.71 hours; the program is not alerted when an overflow occurs.

This field is not updated if the channel-subsystem-timing facility is not provided for the subchannel or if the initial-command-response-measurement facility is not installed.

Control-Unit-Defer Time: Control-unit-defer time is the sum of the time intervals measured by the control unit whenever the device is logically connected to the channel subsystem during an I/O operation but is not actively communicating with the channel because of device-dependent delays in channel-program execution. The control-unit-defer time is not stored in the measurement block as a separate measurement field but is used in the calculation of device-connect-time measurement and device-disconnect-time measurement for an operation.

Control-unit-defer time, if provided by a control unit, is accrued while the device is logically connected to the channel. The time is reported to the channel when channel-end status is presented that causes a device disconnection or terminates the I/O operation. Control-unit-defer time is subtracted from the device-connect-time measurement and is added to the device-disconnect-time measurement reported for the operation.

Reserved: The remaining words of the measurement block, along with any words associated with facilities that are not provided by the channel subsystem or the subchannel, are reserved for future use. They are not updated by the measurement-block-update facility.

Programming Note: It is possible for the program to fetch a portion of a measurement block immediately prior to, or during, a measurement block update by the measurement-block-update facility. To ensure

that a consistent measurement block is fetched, the program should not fetch measurement data for a subchannel during the time from when a start or resume function is initiated at the subchannel until the operation is suspended or completes with secondary status.

Measurement-Block Format

The measurement block is stored as either a format-0 measurement block or a format-1 measurement block. The format-0 measurement block is a 32-byte area (format-0 measurement block) at a location designated by the program using the measurement-block origin in conjunction with the measurement-block index. The format-1 measurement block is a 64-byte area (format-1 measurement block) at a location designated by the measurement-block address provided in the SCHIB during the execution of the MODIFY SUBCHANNEL instruction. The measurement-block-format-control bit at the subchannel indicates whether a format-0 or format-1 measurement block is stored when measurement-block-update mode is active and enabled at the subchannel.

Measurement-Block Origin

The measurement-block origin is the beginning of the measurement-block area in main storage, used to store format-0 measurement blocks. The absolute address of the measurement-block origin, specified on a 32-byte boundary, is passed in general register 2 to the measurement-block-update facility when SET CHANNEL MONITOR is executed with bit 62 of general register 1 set to one.

Measurement-Block Address

The measurement-block address is set at the subchannel through the execution of MODIFY SUBCHANNEL. The measurement block address specifies the absolute address of the beginning of the 64-byte area to be used for accumulating format-1 measurement-block parameters for that subchannel.

Programming Note: The initial value of the measurement-block address is zero. The program is responsible for setting the measurement-block address to the proper value prior to enabling the subchannel for the measurement-block-update mode for format-1 measurement blocks and making the mode active.

Measurement-Block Key

Bits 32-35 of general register 1 form the four-bit access key to be used for subsequent measurement-block updates when SET CHANNEL MONITOR causes the measurement-block-update mode to be made active. The measurement-block key is passed to the measurement-block-update facility whenever the measurement-block-update mode is made active. The key is used for both format-0 and format-1 measurement block updates.

Measurement-Block Index

The measurement-block index is set in the subchannel through the execution of MODIFY SUBCHANNEL. The measurement-block index specifies which 32-byte measurement block, relative to the measurement-block origin, is to be used for accumulating the format-0 measurement-block parameters for that subchannel. The location of the format-0 measurement block of a subchannel is computed by the measurement-block-update facility by appending five rightmost zeros to the measurement-block index of the subchannel and adding the result to the measurement-block origin. The result is the absolute address of the 32-byte format-0 measurement block for that subchannel. When the computed measurement-block address exceeds 2^{31} , a measurement-block program-check condition is recognized, and measurement-block updating does not occur for the preceding subchannel-active period.

Programming Note: The initial value of the measurement-block index is zero. The program is responsible for setting the measurement-block index to the proper value prior to enabling the subchannel for the measurement-block-update mode and making the mode active. To ensure predictable results for the measured parameters in the measurement block, each subchannel for which measured parameters are to be accumulated must have a different value for its measurement-block index.

Measurement-Block-Update Mode

The measurement-block-update mode is made active by the execution of SET CHANNEL MONITOR with bit 62 of general register 1 set to one. If bit 62 of general register 1 is zero when SET CHANNEL MONITOR is executed, the mode is made inactive. When the measurement-block-update mode is inactive, no measurement values are accumulated in main storage. When the measurement-block-update mode is made active, the contents of general register 2 are passed to the measurement-block-update facil-

ity as the absolute address of the measurement-block origin and is used to calculate the address of format-0 measurement blocks. The measurement-block origin is not used for format-1 measurement blocks. The MBK is also passed to the measurement-block-update facility as the access key to be used when updating either format-0 or format-1 measurement blocks for each subchannel. When the measurement-block-update mode is active, the measurement-block-update facility accumulates measurements in individual measurement blocks for subchannels whose measurement-block-update-enable bit is one. (See the section “Measurement Block” on page 17-3 for a description of the measured parameters.)

If the measurement-block-update mode is already active when SET CHANNEL MONITOR is executed, the values for the measurement-block origin and measurement-block key that are used for a subchannel enabled for measuring by the measurement-block-update facility are dependent upon whether SET CHANNEL MONITOR is executed prior to, during, or subsequent to the execution of START SUBCHANNEL for that subchannel. If SET CHANNEL MONITOR is executed prior to START SUBCHANNEL, the current measurement-block origin and measurement-block key are in control. If SET CHANNEL MONITOR is executed during or subsequent to execution of START SUBCHANNEL, it is unpredictable whether the measurement-block origin and measurement-block key that are in control are old or current.

Measurement-Block-Format Control

Bit 29, word 6, of the SCHIB is the measurement-block-format-control bit. This bit provides the capability of specifying whether a format-0 or format-1 measurement block is stored on a subchannel basis. The initial value of the bit is zero. When MODIFY SUBCHANNEL is executed with the measurement-block-format-control bit in the SCHIB operand set to one, the format-1 measurement block is specified for the subchannel. If the measurement-block-update mode is active and enabled at the subchannel, the measurement-block-update facility stores a format-1 measurement-block for the subchannel, starting with the next START SUBCHANNEL issued to that subchannel. Similarly, if MODIFY SUBCHANNEL is executed with measurement-block-format-control bit of the SCHIB operand set to zero by the program, the measurement-block-update facility stores a format-0 measurement-block for the subchannel, starting with

the next START SUBCHANNEL issued to that subchannel.

Measurement-Block-Update Enable

Bit 11 of word 1 of the SCHIB is the measurement-block-update-enable bit. This bit provides the capability of controlling the accumulation of measurement-block parameters on a subchannel basis. The initial value of the enable bit is zero. When MODIFY SUBCHANNEL is executed with the enable bit set to one in the SCHIB, the subchannel is enabled for the measurement-block-update mode. If the measurement-block-update mode is active, the measurement-block-update facility accumulates measurement-block parameters for the subchannel, starting with the next START SUBCHANNEL issued to that subchannel. Conversely, if MODIFY SUBCHANNEL is executed with bit 11 of word 1 of the SCHIB operand set to zero by the program, the subchannel is disabled for the measurement-block-update mode, and no additional measurement-block parameters are accumulated for that subchannel.

Control-Unit-Queuing Measurement

The control-unit-queuing-measurement facility allows the channel subsystem to accept queuing times from control units and, in conjunction with the measurement-block-update facility, to accumulate those times in the measurement block.

The System Library publication for the control-unit model specifies its ability to supply queuing time. If a control-unit model is capable of supplying queuing time, the publication specifies the conditions that prevent the control unit from accumulating an accurate control-unit-queuing time.

Control-Unit-Defer Time

The control-unit-defer-time facility allows the channel subsystem to accept defer times from control units and, in conjunction with the measurement-block-update facility, to modify the device-connect and device-disconnect times reported in the measurement block to reflect the defer time. The control-unit-defer time is subtracted from the device-connect-time measurement and is added to the device-disconnect-time measurement reported for an I/O operation.

The System Library publication for the control-unit model specifies its ability to supply defer time. If a control-unit model is capable of supplying defer time, the publication specifies the conditions that prevent

the control unit from accumulating an accurate control-unit-defer time.

Device-Active-Only Measurement

The device-active-only-measurement facility permits the channel subsystem to report the times that the device is disconnected between primary status and secondary status at the end of an I/O operation or chain of I/O operations.

The measurement-block updates are performed at the time that secondary status is accepted from the I/O device, in order that the device-active time between primary status and secondary status can be reported.

If the subchannel is start pending when secondary status is accepted from the I/O device and the measurement-block update is to be performed, the measurement-block update is performed prior to performing the start function. If measurement-block errors occur, they are reported to the program along with the secondary status instead of performing the start function.

Initial-Command-Response Measurement

The initial-command-response-measurement facility allows the channel subsystem to calculate initial-command-response time for I/O operations and, in conjunction with the measurement-block-update facility, to accumulate those times in the measurement block. The initial-command-response time is accumulated in word 7 of the measurement block.

Time-Interval-Measurement Accuracy

On some models, when time intervals are to be measured and condition code 0 is set for START SUBCHANNEL (or RESUME SUBCHANNEL in the case of a suspended subchannel), a period of latency may occur prior to the initiation of the function-pending time measurement. The System Library publication for the system model specifies the mean latency value and variance for each of the measured time intervals.

Programming Notes:

1. Excessive delays may be encountered by the channel subsystem when attempting to update measurement data if the program is concurrently accessing the same measurement-block area. A programming convention should ensure that the

storage block designated by SET CHANNEL MONITOR is made read-only while the measurement-block-update mode is active.

2. To ensure that programs written to support measurement functions are executed properly, the program should initialize all the measurement blocks to zeros prior to making the measurement-block-update mode active. Only zeros should appear in the reserved and unused words of the measurement blocks.
3. When the incrementing of an accumulated value causes a carry to be propagated out of bit position 0, the carry is ignored, and accumulating continues from zero on.

Device-Connect-Time Measurement

The device-connect-time-measurement facility provides the program with the capability of retrieving the length of time that a device is actively communicating with the channel subsystem while executing a channel program. The measured length of time that the device spends actively communicating on a channel path during the execution of a channel program is called the device-connect-time interval (DCTI). Control-unit-defer time is not included in the DCTI.

If timing facilities are provided for the subchannel, the DCTI value is passed to the program in the extended-status word (ESW) at the completion of the operation when the primary-status condition is cleared by TEST SUBCHANNEL and when TEST SUBCHANNEL clears an intermediate-status condition alone while the subchannel is suspended. The DCTI value passed in the ESW pertains to the previous subchannel-active period. The passing of the DCTI in the ESW is under program control by the SET CHANNEL MONITOR device-connect-time-measurement mode-control bit and the corresponding enable bit in the subchannel. However, the DCTI value is not stored in the ESW if the I/O function initiated by START SUBCHANNEL is terminated because of an error condition that is described by subchannel logout. See the section "Extended-Status Format 0" on page 16-47. In this case, the extended-status bit (L) of the SCSW is stored as one, indicating that the ESW contains logout information describing the error condition. See the section "Extended-Status Word" on page 16-47 for the description of the logout information. If the accrued DCTI value exceeded 8.388608 seconds

during the previous subchannel-active period, then the maximum value (FFFF hex) is passed in the ESW.

Device-Connect-Time-Measurement Mode

The device-connect-time-measurement mode is made active by the execution of SET CHANNEL MONITOR when bit 63 of general register 1 is one. If bit 63 of general register 1 is zero when SET CHANNEL MONITOR is executed, the mode is made inactive, and DCTIs are not passed to the program. When timing facilities are provided for the subchannel, the device-connect-time-measurement mode is active, and the subchannel is enabled for the mode, the DCTI value is passed to the program in the ESW stored when TEST SUBCHANNEL (1) clears the primary-interruption condition with no logout information indicated in the SCSW (extended-status-word-format bit is zero) or (2) clears the intermediate-status condition alone while the subchannel is suspended.

If a start function is currently being executed with a subchannel enabled for the device-connect-time-measurement mode when SET CHANNEL MONITOR makes this mode active for the channel subsystem, the value of the DCTI stored under the appropriate conditions may be zero, a partial result, or the full and correct value, depending on the model and the progress of the start function at the time the mode was activated.

Provision of the DCTI value in the measurement-block area is not affected by whether the device-connect-time-measurement mode is active.

Device-Connect-Time-Measurement Enable

Bit 12 of word 1 of the SCHIB is the device-connect-time measurement-mode-enable bit. This bit provides the program with the capability of selectively controlling the storing of DCTI values for a subchannel when the device-connect-time-measurement mode is active. The initial value of the enable bit is zero. When this enable bit is one in the SCHIB and MODIFY SUBCHANNEL is executed, the subchannel is enabled for the device-connect-time-measurement mode. If the device-connect-time-measurement mode is active, the device-connect-time-measurement facility begins providing DCTI values for the subchannel, starting with the next START SUBCHANNEL issued to the subchannel. In this situation, the DCTI values are provided in the ESW (see

the section “Extended-Status Format 2” on page 16-54). Similarly, if MODIFY SUBCHANNEL is executed with bit 12 of word 1 of the SCHIB operand set to zero by the program, the subchannel is disabled for the device-connect-time-measurement mode, and no further DCTI values are passed to the program for that subchannel.

Extended Measurement Word

The extended-I/O-measurement-word facility provides the program with the capability of retrieving measurement information for a channel program. The measurement information is stored into the extended-measurement word (EMW) in the Interruption Response Block when the extended-measurement-word enable bit is one at the subchannel. See the section “Extended-Measurement Word” on page 16-56 in Chapter 16, “I/O Interruptions,” for the description of the extended-measurement word.

When the extended-measurement-word is enabled for the subchannel, measurement values are passed to program in the EMW when TEST SUBCHANNEL clears a primary-status condition, secondary-status condition alone, or an intermediate-status condition alone while the subchannel is suspended. The measurement values stored in the EMW pertain to the previous subchannel-active and device-active period. Measurement values are not stored in the EMW if the I/O function initiated by START SUBCHANNEL is terminated because of an error condition that is described by subchannel logout (see the section “Extended-Status Format 0” on page 16-47). In this case, the extended-status bit (L) of the SCSW is stored as one, indicating that the ESW contains logout information describing the error condition. See the section “Extended-Status Word” on page 16-47 for the description of the logout information. If any of the accrued measurement values exceeded the maximum value capable of being measured during the previous subchannel-active and device-active period, then the maximum value is stored for that value in the EMW.

Extended-Measurement-Word Enable

Bit 30 of word 6 of the SCHIB is the extended-measurement-word enable bit. This bit provides the program with the capability of selectively controlling the storing of measurement values for a subchannel. The initial value of the enable bit is zero. When this enable bit is one in the SCHIB and MODIFY SUBCHANNEL is executed, the subchannel is enabled

for the extended-measurement-word and the extended-measurement-word facility begins providing measurement values for the subchannel starting with the next START SUBCHANNEL issued to the subchannel. Similarly, if MODIFY SUBCHANNEL is executed with bit 30, word 6, of the SCHIB operand set to zero by the program, the subchannel is disabled for the extended-measurement-word and no further measurement values are passed to the program for that subchannel.

Signals and Resets

During system operation, it may become necessary to terminate an I/O operation or to reset either the I/O system or a portion of the I/O system. (The I/O system consists of the channel subsystem plus all of the attached control units and devices.) Various signals and resets are provided for this purpose. Three signals are provided for the channel subsystem to notify an I/O device to terminate an operation or perform a reset function or both. Two resets are provided to cause the channel subsystem to reinitialize certain information contained either at the I/O device or at the channel subsystem.

Signals

The request that the channel subsystem initiate a signaling sequence is made by one of the following:

1. The program's issuance of the CLEAR SUBCHANNEL, HALT SUBCHANNEL, or RESET CHANNEL PATH instruction
2. The I/O device's signaling of I/O-error alert
3. The channel subsystem itself, upon detecting certain error conditions or equipment malfunctions

The three signals are the halt signal, the clear signal, and the reset signal.

Halt Signal

The halt signal is provided so the channel subsystem can terminate an I/O operation. The halt signal is issued by the channel subsystem as part of the halt function performed subsequent to the execution of HALT SUBCHANNEL. The halt signal is also issued

by the channel subsystem when certain error conditions are encountered.

For the parallel-I/O-interface type of channel path, the halt signal results in the channel subsystem using the interface-disconnect sequence control defined in the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974.

For the ESCON-I/O-interface type of channel path, the halt signal results in the channel subsystem using the cancel function defined in the System Library publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

For the FICON-I/O-interface type of channel path and the subchannel is operating in command-mode, the halt signal results in the channel subsystem using the cancel function defined in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

For the FICON-I/O-interface type of channel path with the subchannel is operating in transport mode, the halt signal results in the channel subsystem using the transport-mode abort-sequence function.

Clear Signal

The clear signal is provided so the channel subsystem can terminate an I/O operation and reset status and control information contained at the device. The clear signal is issued as part of the clear function performed subsequent to the execution of CLEAR SUBCHANNEL. The clear signal is also issued by the channel subsystem when certain error conditions or equipment malfunctions are detected by the I/O device or the channel subsystem.

For the parallel-I/O-interface type of channel path, the clear signal results in the channel subsystem using the selective-reset sequence control defined in the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974.

For the ESCON-I/O-interface type of channel path, the clear signal results in the channel subsystem using the selective-reset function defined in the System Library publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

For the FICON-I/O-interface type of channel path and the subchannel is operating in command-mode, the clear signal results in the channel subsystem using the selective-reset function defined in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

For the FICON-I/O-interface type of channel path with the subchannel is operating in transport mode, the clear signal results in the channel subsystem using the selective-reset function.

When the subchannel is operating in command mode, if an I/O operation is in progress at the device and the device is actively communicating over a channel path in the performance of that I/O operation when a clear signal is received on that channel path, the device disconnects from that channel path upon receiving the clear signal. Data transfer and any operation using the facilities of the control unit are immediately concluded, and the I/O device is not necessarily positioned at the beginning of a block. Mechanical motion not involving the use of the control unit, such as rewinding magnetic tape or positioning a disk-access mechanism, proceeds to the normal stopping point, if possible. The device may appear busy until termination of the mechanical motion or the inherent cycle of operation, if any, whereupon it becomes available. Status information in the device and control unit is reset, but an interruption condition may be generated upon the completion of any mechanical operation.

Reset Signal

The reset signal is provided so the channel subsystem can reset all I/O devices on a channel path. The reset signal is issued by the channel subsystem as part of the channel-path-reset function performed subsequent to the execution of RESET CHANNEL PATH. The reset signal is also issued by the channel subsystem as part of the I/O-system-reset function.

For the parallel-I/O-interface type of channel path, the reset signal results in the channel subsystem using the system-reset sequence control defined in the System Library publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974.

For the ESCON-I/O-interface type of channel path, the reset signal results in the channel subsystem using the system-reset function defined in the Sys-

tem Library publication *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202.

For the FICON-I/O-interface type of channel path, the reset signal results in the channel subsystem using the system-reset function defined in the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

Resets

Two resets are provided so the channel subsystem can reinitialize certain information contained at either the I/O device or the channel subsystem. The request that the channel subsystem initiate one of the reset functions is made by one of the following:

1. The program's issuance of the RESET CHANNEL PATH instruction
2. The operator's activation of a system-reset-clear or system-reset-normal key or a load-clear or load-normal key
3. The channel subsystem itself upon detecting certain error conditions or equipment malfunctions

The resets are channel-path reset and I/O-system reset.

Channel-Path Reset

The channel-path-reset facility provides a mechanism to reset certain indications that pertain to a designated channel path at all associated subchannels. Channel-path reset occurs when the channel subsystem performs the channel-path-reset function initiated by RESET CHANNEL PATH. (See "RESET CHANNEL PATH" on page 14-9.) All internal indications of dedicated allegiance, control unit busy, and device busy that pertain to the designated channel path are cleared in all subchannels, and reset is signaled on that channel path.

The receipt of the reset signal by control units attached to that channel path causes all operations in progress and all status, mode settings, and allegiance, pertaining to that channel path, of the control unit and its attached devices to be reset. (See also the description of the system-reset-signal actions in "I/O-System Reset" on page 17-13.)

The results of the channel-path-reset function on the designated channel path are communicated to the

program by means of a subsequent machine-check-interruption condition generated by the channel subsystem (see “Channel-Subsystem Recovery” on page 17-24).

I/O-System Reset

The I/O-system-reset function is performed when the channel subsystem is powered on, when initial program loading is initiated manually (see “Initial Program Loading” on page 17-16), and when the system-reset-clear or system-reset-normal key is activated. The I/O-system-reset function cannot be initiated under program control; it must be initiated manually. I/O-system reset may fail to complete due to malfunctions detected at the channel subsystem or on a channel path. I/O-system reset is performed as part of subsystem reset, which also resets all floating interruption requests, including pending I/O interruptions. (See “Subsystem Reset” on page 4-54.) Detailed descriptions of the effects of I/O-system reset on the various components of the I/O system appear later in this chapter.

I/O-system reset provides a means for placing the channel subsystem and its attached I/O devices in the initialized state. I/O-system reset affects only the channel-subsystem configuration in which it is performed, including all channel-subsystem components configured to that channel subsystem. I/O-system reset has no effect on any system components that are not part of the channel-subsystem configuration that is being reset. The effects of I/O-system reset on the configured components of the channel subsystem are described in the following sections.

Channel-Subsystem State: I/O-system reset causes the channel subsystem to be placed in the initialized state, with all the channel-subsystem components in the states described in the following sections. All operations in progress are terminated and reset, and all indications of prior conditions are reset. These indications include status information, interruption conditions (but not pending interruptions), dedicated-allegiance conditions, pending channel reports, and all internal information regarding prior conditions and operations. In the initialized state, the channel subsystem has no activity in progress and is ready to perform the initial-program-loading (IPL) function or respond to I/O instructions, as described in “I/O Instructions” on page 14-1.

Control Units and Devices: I/O-system reset causes a reset signal to be sent on all configured

channel paths, including those which are not physically available (as indicated by the PAM bit being zero) because of a permanent error condition detected earlier. When the reset signal is received by a control unit, control-unit functions in progress, control-unit status, control-unit allegiance, and control-unit modes for the resetting channel path are reset. Device operations in progress, device status, device allegiance, and the device mode for the resetting channel path are also reset. Control-unit and device mode, allegiance, status, and I/O functions in progress for other channel paths are not affected.

For devices that are operating in the single-path mode, an operation can be in progress for, at most, one channel path. Therefore, if the reset signal is received on that channel path, the operation in progress is reset. Devices that have the dynamic-reconnection feature and are operating in the multipath mode, however, have the capability to establish an allegiance to a group of channel paths during an I/O operation, where all the channel paths of the path group are configured to the same channel subsystem. If an operation is in progress for a device that is operating in the multipath mode and the reset signal is received on one of the channel paths of that path group, then the operation in progress is reset for the resetting channel path only. Although the operation in progress cannot continue on the resetting channel path, it can continue on the other channel paths of the path group, subject to the following restrictions:

1. If the device is actively communicating with the channel subsystem on a channel path when it receives the reset signal on that channel path, then the operation is reset unconditionally, regardless of path groups.
2. If the operation is in progress in the multipath mode but the path group consists only of the resetting path, then the operation is reset.
3. Except as noted in item 2, if the operation in progress is currently in a disconnected state (device not actively communicating with the channel subsystem) or is active on another channel path of a path group, system reset has no effect upon the continued performance of the operation.

A control unit is completely reset after the reset signal has been received on all its channel paths, provided no new activity is initiated at the control unit

between the receipt of the first and last reset signal. "Completely reset" means that the current operation, if any, at the control unit is terminated and that control-unit allegiance, control-unit status, and the control-unit mode, if any, are reset.

An I/O device is completely reset after the reset signal has been received on all channel paths of all control units by which the device is accessible, provided no new activity is initiated at the device between the receipt of the first and last reset signal. "Completely reset" means that the current operation, if any, at the device is terminated and that device allegiance, device status, and the device mode are reset.

In summary, system reset always causes an operation in progress to be reset for the channel path on which the reset signal is received. If the resetting channel path is the only channel path for which the operation is in progress, then the operation is completely reset. If a device is actively communicating on a channel path over which the reset signal is received, then the operation in progress is unconditionally and completely reset.

The reset signal is not received by control units and devices on channel paths from which the control unit has been partitioned. A control unit is partitioned from a channel path by means of an enable/disable switch on the control unit for each channel path by which it is accessible. Multi-tagged, unsolicited status, if any, remains pending at the control unit for such a channel path in this case. However, from the point of view of the program, the control unit and device appear to be completely reset if the reset signal is received by the control unit on all the channel paths by which it is currently accessible.

The resultant reset state of individual control units and devices is described in the System Library publication for the control unit.

Channel Paths: I/O-system reset causes a reset signal to be sent on all configured channel paths and causes the channel subsystem to be placed in the reset and initialized state, as described in the previous sections. As a result of these actions, all communication between the channel subsystem and its attached control units and devices is terminated and the components reset, and all configured channel paths are made quiescent or are deconfigured.

Subchannels: I/O-system reset causes all operations on all subchannels to be concluded. Status

information, all interruption conditions (but not pending interruptions), dedicated-allegiance conditions, and internal indications regarding prior conditions and operations in all subchannels are reset, and all valid subchannels are placed in the initialized state.

In the initialized state, the subchannel parameters of all valid subchannels are set to their initial values. The initial values of the following subchannel parameters are zeros:

- Interruption parameter
- I/O-interruption-subclass code (ISC)
- Enabled
- Limit mode when the address-limit-checking facility is installed
- Measurement mode
- Multipath mode
- Path-not-operational mask
- Last-path-used mask
- Measurement-block index
- Concurrent sense

The initial values of the following subchannel parameters are assigned as part of the installation procedure for the device associated with each valid subchannel:

- Timing facility
- Device number
- Logical-path mask (same value as path-installed mask)
- Path-installed mask
- Path-available mask
- Channel-path ID 0-7

The values assigned may depend upon the particular system model and the configuration; dependencies, if any, are described in the System Library publication for the system model. Programming considerations may further constrain the values assigned.

The initial value of the path-operational mask is all ones.

The device-number-valid bit is one for all subchannels having an assigned I/O device.

The initial value of the model-dependent area of the subchannel-information block is described in the System Library publication for the system model.

The initial value of the subchannel-status word and extended-status word is all zeros.

The initialized state of the subchannel is the state specified by the initial values for the subchannel parameters described above. The description of the subchannel parameters can be found in “Subchannel-Information Block” on page 15-2, “Subchannel-Status Word” on page 16-7, and “Extended-Status Word” on page 16-47.

Channel-Path-Reset Facility: I/O-system reset causes the channel-path-reset facility to be reset. A channel-path-reset function initiated by RESET CHANNEL PATH, either pending or in progress, is overridden by I/O-system reset. The machine-check-interruption condition, which normally signals the completion of a channel-path-reset function, is not generated for a channel-path-reset function that is pending or in progress at the time I/O-system reset occurs.

Address-Limit-Checking Facility: When the address-limit-checking facility is installed, I/O-system reset causes the address-limit-checking facility to be reset. The address-limit value is initialized to all zeros and validated.

Channel-Subsystem-Monitoring Facilities: I/O-system reset causes the channel-subsystem-monitoring facilities to be reset. The measurement-block-update mode and the device-connect-time-measurement mode, if active, are made inactive. The measurement-block origin and the measurement-block key are both initialized to zeros and validated.

Pending Channel Reports: I/O-system reset causes pending channel reports to be reset.

Channel-Subsystem Timer: I/O-system reset does not necessarily affect the contents of the channel-subsystem timer. In models that provide channel-subsystem-timer checking, I/O-system reset may cause the channel-subsystem timer to be validated.

Pending I/O Interruptions: I/O-system reset does not affect pending I/O interruptions. However, during subsystem reset, I/O interruptions are cleared concurrently with the performance of I/O-system reset. (See “Subsystem Reset” on page 4-54.)

Area Affected	Effect on I/O-System Reset ¹
Channel-subsystem state	Reset and initialized
Control units and devices	Reset
Channel paths	Quiescent
Subchannels	Reset and initialized
Interruption parameter	Zeros ²
I/O-interruption-subclass code (ISC)	Zeros ²
Enabled bit	Zero ²
Address-limit-mode bits ⁴	Zeros ²
Timing-facility bit	Installed value ²
Multipath-mode bit	Zero ²
Measurement-mode bits	Zeros ²
Device-number-valid bit	Installed value ²
Device number	Installed value ²
Logical-path mask	Equal to path-installed mask value ²
Path-not-operational mask	Zeros ²
Last-path-used mask	Zeros ²
Path-installed mask	Installed value ²
Measurement-block index	Zeros ²
Path-operational mask	Ones ²
Path-available mask	Installed value ^{2 3}
Channel-path ID 0-7	Installed value ²
Concurrent-sense bit	Zero ²
Subchannel-status word	Zeros ²
Extended-status word	Zeros ²
Model-dependent area	Model dependent ²
Channel-path-reset facility	Reset
Address-limit-checking facility ⁴	Reset and initialized
Address-limit value	Zeros ²
Channel-subsystem-monitoring facility	Reset and initialized
Measurement-block-update mode	Inactive ²
Device-connect-time-measurement mode	Inactive ²
Measurement-block origin	Zeros ²
Measurement-block key	Zeros ²
Pending channel-report words	Cleared
Channel-subsystem timer	Unchanged/validated
Explanation: ¹ For a detailed description of the effect of I/O-system reset on each area, see the text. ² Initialized value. ³ Also subject to model-dependent configuration controls, if any. ⁴ When the FCX facility is not installed.	

Figure 17-1. Summary of I/O-System-Reset Actions

Externally Initiated Functions

I/O-system reset, which is an externally initiated function, is described in “I/O-System Reset” on page 17-13.

Initial Program Loading

Initial program loading (IPL) provides a manual means for causing a program to be read from a designated device and for initiating the execution of that program.

Some models may provide additional controls and indications relating to IPL; this additional information is specified in the System Library publication for the model.

There are two types of IPL: CCW-type IPL and list-directed IPL. CCW-type IPL is provided by all machine configurations. List-directed IPL may be provided, depending on the model.

CCW-type IPL

CCW-type IPL is initiated manually by setting the load-unit-address controls to a four-digit number to designate an input device and by subsequently activating the load-clear or load-normal key.

Activating the load-clear or load-normal key sets the architectural mode to the ESA/390 mode. For ease of reference, the additional elements of the description of ESA/390 IPL are given below.

Activating the load-clear key causes a clear reset to be performed on the configuration.

Activating the load-normal key causes an initial CPU reset to be performed on this CPU, CPU reset to be propagated to all other CPUs in the configuration, and a subsystem reset to be performed on the remainder of the configuration.

In the loading part of the operation, after the resets have been performed, this CPU enters the load state. This CPU does not necessarily enter the stopped state during performance of the reset. The load indicator is on while the CPU is in the load state.

Subsequently, if conditions allow, a read operation is initiated from the designated input device and associated subchannel. The read operation is performed as if a START SUBCHANNEL instruction were executed that designated (1) the subchannel corresponding to the device number specified by the load-unit-address controls and (2) an ORB containing all zeros, except for a byte of all ones in the logical-path-mask field. The ORB parameters are interpreted by the channel subsystem as follows:

- **Interruption parameter:** All zeros
- **Subchannel key:** All zeros
- **Suspend control:** Zero (suspension not allowed)
- **CCW format:** Zero
- **CCW prefetch:** Zero (prefetching not allowed)

- **Initial-status-interruption control:** Zero (no request)
- **Address-limit-checking control:** Zero (no checking), when the address-limit-checking facility is installed
- **Suppress suspended interruption:** Zero (suppression not allowed)
- **Channel-program-type control:** Zero (CCW channel program), when the FCX-facility is installed
- **Logical-path mask:** Ones (all channel paths logically available)
- **Incorrect-length-suppression mode:** Zero (ignored because format-0 CCWs are specified)
- **Channel-program address:** Absolute address 0

The first CCW to be executed is not fetched from storage. Instead, the effect is as if an implied format-0 CCW, beginning in absolute location 0 and having the following detailed format, were executed:

Loc.

0	00000010	000000000000000000000000000000
4	01100000	////////000000000000110000
	0	8 16 31

In the illustration above, the CCW specifies a read command with the modifier bits zeros, a data address of 0, a byte count of 24, the chain-command flag one, the suppress-incorrect-length-indication flag one, the chain-data flag zero, the skip flag zero, the program-controlled-interruption (PCI) flag zero, the indirect-data-address (IDA) flag zero, and the suspend flag zero. The CCW fetched, as a result of command chaining, from location 8 or 16, as well as any subsequent CCW in the IPL sequence, is interpreted the same as a CCW in any I/O operation, except that any PCI flags that are specified in the IPL channel program are ignored.

At the time the subchannel is made start pending for the IPL read, it is also enabled, which ensures proper handling of subsequent status from the device by the channel subsystem and facilitates subsequent I/O operations using the IPL device. (Except for the subchannel used by the IPL I/O operation, each subchannel must first be made enabled by MODIFY SUBCHANNEL before it can accept a start function or any status from the device.)

When the IPL subchannel becomes status pending for the last operation of the IPL channel program, no I/O-interruption condition is generated. Instead, the

subsystem ID is stored in absolute locations 184-187, zeros are stored in absolute locations 188-191, and the subchannel is cleared of the pending status as if TEST SUBCHANNEL had been executed but without storing information usually stored in an IRB. If the subchannel-status field that would normally have been stored is all zeros and the device-status field that would normally have been stored contains only the channel-end indication, with or without the device-end indication, the IPL I/O operation is considered to be completed successfully. If the device-end status for the IPL I/O operation is provided separately after channel-end status, it causes an I/O-interruption condition to be generated.

If an interface-control check is detected during execution of the IPL channel program, some models may re-initiate the same channel program on another channel path. The other path chosen is any logically-available channel path to the designated device on which the IPL channel program has not previously been initiated during the current IPL operation. (See programming note 6 on page 17-19 for additional information.)

When the IPL I/O operation is completed successfully, a new PSW is loaded from absolute locations 0-7. If the PSW loading is successful and no malfunctions are recognized that preclude the completion of IPL, then the CPU leaves the load state, and the load indicator is turned off. If the rate control is set to the process position, the CPU enters the operating state, and CPU operation proceeds under control of the new PSW. If the rate control is set to the instruction-step position, the CPU enters the stopped state, with the manual indicator on, after the new PSW has been loaded.

If the IPL I/O operation or the PSW loading is not completed successfully, the CPU remains in the load state, and the load indicator remains on.

IPL does not complete when any of the following occurs

- No subchannel contains a valid device number equal to the IPL device number specified by the load-unit-address controls.
- A malfunction is detected in the CPU, main storage, or channel subsystem that precludes the completion of IPL.

- Unsolicited alert status is presented by the device subsequent to the subchannel becoming start pending for the IPL read and before the IPL subchannel becomes subchannel active. The IPL read operation is not initiated in this case.
- There were no available channel paths to the IPL device.
- On every available channel path to the IPL device, either the IPL device appeared not operational or an interface-control check was detected.
- The IPL device presented a status byte containing indications other than channel end, device end, status modifier, control unit end, control unit busy, device busy, or retry status during the IPL I/O operation. Whenever control unit end, control unit busy, or device busy is presented in the status byte, normal path-management actions are taken.
- A subchannel-status indication other than PCI was generated during the IPL I/O operation, except that on some models, detection of an interface-control check may cause the I/O operation to be retried on another available channel path to the IPL device.
- The PSW loaded from absolute locations 0-7 has a PSW-format error of the type that is recognized early.

Except in the cases of no corresponding subchannel for the device number entered or a machine malfunction, the subsystem ID of the IPL device is stored in absolute locations 184-187; otherwise, the contents of these locations are unpredictable. In all cases of unsuccessful IPL, the contents of absolute locations 0-7 are unpredictable.

Subsequent to a successful IPL, the subchannel parameters contain the normal values as if an actual START SUBCHANNEL had been executed, designating the ORB as described above.

Programming Notes:

1. The information read and placed at absolute locations 8-15 and 16-23 may be used as CCWs for reading additional information during the IPL I/O operation: the CCW at location 8 may specify reading additional CCWs elsewhere in storage, and the CCW at location 16 may specify the

transfer-in-channel command, causing transfer to these CCWs.

2. The status-modifier bit has its normal effect during the IPL I/O operation, causing the channel subsystem to fetch and chain to the CCW whose address is 16 higher than that of the current CCW. This applies also to the initial chaining that occurs after completion of the read operation specified by the implicit CCW.
3. The PSW that is loaded at the completion of the IPL operation may be provided by the first eight bytes of the IPL I/O operation or may be placed at absolute locations 0-7 by a subsequent CCW.
4. Activating the load-normal key implicitly specifies the use of the first 24 bytes of main storage and the eight bytes at absolute locations 184-191. Since the remainder of the IPL program may be placed in any part of storage, it is possible to preserve such areas of storage as may be helpful in debugging or recovery. The IPL program should not be placed in the low 512 bytes of storage since that area is reserved as described in a programming note under "Compatibility between z/Architecture and ESA/390" on page 1-27. When the load-clear key is activated, the IPL program starts with a cleared machine in a known state, except that information on external storage remains unchanged.
5. When the PSW at absolute location 0 has bit 14 set to one, the CPU is placed in the wait state after the IPL operation is completed. At that point, the load and manual indicators are off, and the wait indicator is on.
6. When the program receives control after the machine leaves the load state during IPL, the last-path-used mask (LPUM) in the subchannel-information block (SCHIB) for the IPL device should be saved. The program should use this saved LPUM as the logical-path mask (LPM) in the ORB until the program reaches the point in its initialization where it has the ability to process and recover from interface-control checks, and verify that all of the channel paths associated with the subchannel are connected to the same device.

When multiple paths are associated with the subchannel of the IPL device, it is possible that mis-cabling of one or more of the paths can cause the program initialization to proceed on different

devices. This affects the integrity of the IPL process. The program should ensure that all paths are connected to the same device, but this typically is not possible before program initialization completes. Thus the use of the saved LPUM ensures that all I/O operations to the IPL device during program initialization are executed on the same device, allowing postponement of the checking of the paths.

Using the saved LPUM also helps to ensure that the channel subsystem will not select a path on which errors would prevent completion of the program initialization, since the path indicated in the LPUM has demonstrated its functionality. On some models, using the saved LPUM also ensures that the path selection will not re-select a path on which an interface-control check was detected while the machine was in the load state.

List-Directed IPL

List-directed IPL supports the use of I/O devices which are not accessed by CCWs. List-directed IPL may be provided for a logical partition or virtual machine, depending on the model and the amount of memory available to the logical partition or virtual machine. The terms logical partition and virtual machine are not defined in this publication. The term logical partition is defined in the System Library publications. The term virtual machine is defined in the virtual-machine product manuals.

List-directed IPL is initiated by activating the load-clear-list-directed or load-with-dump key. Prior to activating either of these keys, the list-directed-IPL parameters are entered using a model-dependent console. List-directed-IPL parameters include the following information:

- Information which identifies the I/O device from which a program is to be loaded. This I/O device is referred to as the load device.
- Information which identifies a location on a load device from which a program will be loaded.
- Information to be used by the program which will be loaded from the load device.

The list-directed-IPL parameters required by a specific machine and the means by which they are entered are specified in the System Library publications.

After the list-directed-IPL parameters have been entered, the load-clear-list-directed or load-with-dump key for a particular CPU is activated. In the description which follows, the term “this CPU” refers to the CPU in the configuration for which the load-clear-list-directed key was activated.

Activating the load-clear-list-directed key causes a clear reset to be performed on all of the CPUs in the configuration.

Activating the load-with-dump key causes a store-status operation to be performed on this CPU. After the store-status operation, an initial-CPU-reset operation is performed on this CPU, CPU reset is propagated to all other CPUs in the configuration, and a subsystem reset is performed on the remainder of the configuration.

In the loading part of the operation, after the resets have been performed, the operations listed below are performed.

- If the rate control is not in the process position, it is forced into the process position.
- If the list-directed IPL operation was initiated by activating the load-with-dump key, the contents of a model-dependent number of contiguous logical-partition or virtual-machine storage locations starting with absolute location 0 are stored in a model-dependent holding area. The data stored includes the contents of all storage locations into which the program will be loaded from the load device. See Programming Note 4 for additional information.
- The program to be loaded from the load device is loaded into logical-partition or virtual-machine storage at the locations specified on the load device.
- A system IPL-parameter block used by the machine during list-directed IPL is placed in storage in the format shown in Figure 17-2 on page 17-20. The absolute address of the first byte of the system IPL-parameter block is stored in absolute address locations 20-23. An XML document containing the system IPL parameters immediately follows the last byte of the system IPL-parameter block. (See “Extensible Markup Language (XML) 1.0,”available from “World Wide Web Consortium”(www.w3c.org) for information regarding XML.)

The system IPL-parameter block has the following format:

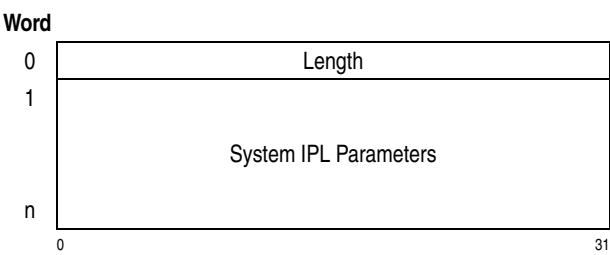


Figure 17-2. System IPL-Parameter Block

Bits 0-31 of word 0 of the system IPL-parameter block contains a length field. The length field contains a 32-bit binary number representing the number of bytes occupied by the system IPL-parameter block. Words 1 through n contain the system IPL parameters.

If no errors occur during the operations listed above, then if a subchannel is associated with the IPL-device, the subsystem-identification word is stored in absolute locations 184-187; otherwise zeroes are stored in absolute locations 184-187. Additionally, zeroes are stored in absolute locations 188-191, and a new PSW is loaded from absolute locations 0-7. If the PSW loading is successful and if no malfunctions are recognized which preclude execution of the program which was loaded from the load device, then operation proceeds under control of the new PSW.

If errors occur during the operations listed above, or if the PSW loading is not successful, then this CPU remains in the operating state. See the appropriate System Library publications for additional information.

When list-directed IPL was initiated by activating the load-clear-list-directed key, the state of the machine resources at the time when the loaded program is initiated is the same as it was immediately after the clear reset was performed except as indicated below. For a description of the state of the machine resources immediately after the clear reset was performed, see “Resets” on page 4-50.

- This CPU is in the operating state. The contents of the ALB and TLB for this CPU are unpredictable.
- The PSW for this CPU is set to the contents of absolute storage locations 0-7.

- Main storage locations into which the program was loaded from the load device contain the program which was loaded, except for absolute storage locations 20-23 and 184-191. Absolute storage locations 20-23 contain the absolute address of the system IPL-parameter block; storage locations starting with this absolute address contain the system IPL-parameter block, immediately followed by an XML document containing the same system IPL parameters. If a subchannel is associated with the IPL device, then absolute storage locations 184-187 contain the subsystem-identification word for the IPL-device and absolute storage locations 188-191 contain zeroes; otherwise, absolute storage locations 184-191 contain zeroes. The contents of all other storage locations are unpredictable.
 - The ACC and F bits of the storage keys associated with all main storage locations are set to zero. The R and C bits of the storage keys associated with the first 4 K-byte block of storage, the blocks of storage containing the XML document and IPL-parameter list, and the blocks of storage containing the loaded program are set to one. The R and C bits of the storage keys associated with all other blocks of storage are unpredictable.
 - The TOD clock is unaffected by initial-program loading.
- The contents of all main storage locations which were stored in the model-dependent holding area are unpredictable except as follows:
 - Main storage locations into which the program was loaded from the load device contain the program which was loaded, except for absolute storage locations 20-23.
 - Absolute storage locations 20-23 contain the absolute address of the system IPL-parameter block; storage locations starting with the absolute address indicated by the contents of absolute addresses 20-23 contain the system IPL-parameter block, which is immediately followed by an XML document containing the system IPL parameters.
 - If a subchannel is associated with the IPL device, then absolute storage locations 184-187 contain the subsystem-identification word for the IPL-device and absolute storage locations 188-191 contain zeroes; otherwise, absolute storage locations 184-191 contain zeroes.
 - The storage keys associated with the blocks of main storage which were stored in the model-dependent holding area are unpredictable. The storage keys associated with all other blocks of main storage are in the same state as they were immediately after the CPUs were reset.
 - The TOD clock is unaffected by initial-program loading.

When list-directed IPL was initiated by activating the load-with-dump key, the state of the machine resources when control is passed to the loaded program is as follows:

- The I/O system has been reset.
 - The state of all CPUs in the configuration except for this CPU is the same as it was immediately after the CPUs were reset.
 - This CPU is in the operating state. The contents of all registers, the ALB, and the TLB for this CPU are unpredictable.
 - The PSW for this CPU is set to the contents of absolute storage locations 0-7.
 - The contents of all main storage locations which were not stored in the model-dependent holding area are unchanged from the time when the load-with-dump key was activated.
- List-directed IPL does not complete if any of the following occurs:
- A malfunction is detected in the CPU, channel subsystem, or main storage which precludes the completion of IPL.
 - The list-directed IPL parameters did not contain sufficient information to identify the load device or the program on the load device.
 - An I/O device designated by the list-directed-IPL parameters is not in the machine configuration.
 - The I/O device designated by the list-directed-IPL parameters is a CCW-type device and is not supported by list-directed IPL.
 - The I/O device appeared not operational.

- The amount of storage in the logical partition or virtual machine is insufficient to allow list-directed IPL to complete.
- The PSW to be loaded from locations 0-7 has a PSW-format error of the type that is recognized early.

Programming Notes:

1. The executable portion of the program loaded during list-directed IPL should be located in absolute storage starting at absolute storage locations 8192 and higher. This is necessary in order to avoid use of the architecturally assigned storage locations. See “Assigned Storage Locations” on page 3-61.
2. When the address indicated by storage locations 20-23 is above 16 MB, it is necessary for the machine to be in 31-bit addressing mode in order to access the IPL-parameter list and XML document.
3. When list-directed IPL is initiated by activation of the load-clear-list-directed key, the program loaded from the load device is typically an operating-system loader. In this case, the load device may also contain the operating system to be loaded by the operating-system loader. See the applicable operating-system manuals for additional information.
4. When list-directed IPL is initiated by activation of the load-with-dump key, the contents of storage which are affected by the IPL operation are stored in a model-dependent holding area so that they can be retrieved by the program which was loaded from the load device. The program loaded from the load device is typically an O/S-specific system dump program which stores the retrieved data onto a dump device. This process ensures that the contents of storage affected by the IPL operation are preserved for analysis during subsequent debug operations. See the applicable operating system manuals for additional information.

Reconfiguration of the I/O System

Reconfiguration of the I/O system is handled in a model-dependent manner. For example, changes may be made under program control, by using the model-dependent DIAGNOSE instruction; or manu-

ally, by using system-operator configuration controls; or by using a combination of DIAGNOSE and manual controls. The method used depends on the system model. The System Library publication for the system model specifies how the changes are made. The partitioning of channel paths because of reconfiguration is indicated by the setting of the PAM bits in the SCHIB stored when STORE SUBCHANNEL is executed (see “Path-Available Mask (PAM)” on page 15-7).

Status Verification

The status-verification facility provides the channel subsystem with a means of indicating that a device has presented a device-status byte that has valid CBC but that contained a combination of bits that was inappropriate when the status byte was presented to the channel subsystem. The indication provided to the program in the ESW by the channel subsystem is called device-status check. When the channel subsystem recognizes a device-status-check condition, an interface-control-check condition is also recognized. For a summary of the status combinations considered to be appropriate or inappropriate, see the System Library publications *IBM Enterprise Systems Architecture/390 ESCON I/O Interface*, SA22-7202, and *IBM System/360 and System/370 I/O Interface Channel to Control Unit OEMI*, GA22-6974, and the ANSI standards document *Fibre Channel - Single-Byte Command Code Sets-2 (FC-SB-2)*.

Address-Limit Checking

The address-limit-checking facility may be installed on a model and when installed provides a storage-protection mechanism for I/O data accesses to storage that augments key-controlled protection. The address-limit-checking facility is not installed when the FCX facility is installed. When address-limit checking is used, absolute storage is divided into two parts by a program-controlled address-limit value. I/O data accesses can then be optionally restricted to only one of the two parts of absolute storage by the limit mode at each subchannel. The address-limit constraint applies at a higher priority than key-controlled protection, that is, I/O data accesses to the part of main storage that is protected by address-limit checking are prevented even when the subchannel

key is zero or matches the key in storage. Address-limit checking does not apply to the fetching of CCWs, IDAWs and MIDAWs.

The address-limit-checking facility consists of the following elements:

- The I/O instruction SET ADDRESS LIMIT.
- The limit mode at each subchannel.
- The address-limit-checking-control bit in the ORB.

The execution of SET ADDRESS LIMIT passes the contents of general register 1 to the address-limit-checking facility to be used as the address-limit value. Bits 32 and 48-63 of general register 1 must be zeros to designate a valid absolute address on a 64K-byte boundary; otherwise, an operand exception is recognized, and the execution of the instruction is suppressed.

The limit mode at each subchannel indicates the manner in which address-limit checking is to be performed. The limit mode is set by placing the desired value in bit positions 9 and 10 of word 1 in the SCHIB and executing MODIFY SUBCHANNEL. The settings of these bits in the SCHIB have the following meanings:

- | | |
|----|--|
| 00 | No limit checking (initialized value). |
| 01 | Data address must be equal to or greater than the current address limit. |
| 10 | Data address must be less than the current address limit. |
| 11 | Reserved. This combination of limit-mode bits causes an operand exception to be recognized when MODIFY SUBCHANNEL is executed. |

The address-limit-checking-control bit, bit 11 of word 1 of the ORB, specifies whether address-limit checking is to be used for the start function that is accepted when the execution of START SUBCHANNEL causes the contents of the ORB to be passed to the subchannel. If the address-limit-checking-control bit is zero when the contents of the ORB are passed, address-limit checking is not specified for that start function. If the bit is one, address-limit checking is specified and is under the control of the current

address limit and the current setting of the limit mode at the subchannel.

During the performance of the start function, an attempt to access an absolute storage location for data that is protected by an address limit (either high or low) is recognized as an address-limit violation, and the access is not allowed. A program-check condition is recognized, and channel-program execution is terminated, just as when an attempt is made to access an invalid address.

Configuration Alert

The configuration-alert facility provides a detection mechanism for devices that are not associated with a subchannel in the configuration. The configuration-alert facility notifies the program, by means of a channel report, that a device which is not associated with a subchannel has attempted to communicate with the program.

Each device must be assigned to a subchannel during an installation procedure; otherwise, the channel subsystem is unable to generate an I/O-interruption condition for the device. This is because the I/O-interruption code contains the subchannel number that identifies the particular device causing the I/O-interruption condition. When a device that is not associated with a subchannel attempts to communicate with the channel subsystem, the configuration-alert facility generates a channel report in which the unassociated device is identified. For a description of the means by which the program is notified of a pending channel report and how the information in the channel report is retrieved, see "Channel Report" on page 17-25.

Incorrect-Length-Indication Suppression

The incorrect-length-indication-suppression facility allows the indication of incorrect length for immediate operations to be suppressed in the same manner when using format-1 CCWs as when using format-0 CCWs. When the incorrect-length-indication-suppression facility is installed, bit 24 of word 1 of the ORB specifies whether the channel subsystem is to suppress the indication of incorrect length for an

immediate operation when format-1 CCWs are used or whether this indication will remain under the control of the SLI flag of the current CCW (as is the case for CCWs not executed as immediate operations). This bit provides the capability for a channel program to operate in the same manner regarding the indication of incorrect length regardless of whether format-0 or format-1 CCWs are used.

Concurrent Sense

The concurrent-sense facility provides a mechanism whereby sense information that is provided by the device can be presented by the channel subsystem to the program in the same command-mode IRB that contains the unit-check indication when the subchannel is in the concurrent-sense mode. The concurrent-sense mode is made active at a subchannel for which the concurrent-sense facility is applicable when MODIFY SUBCHANNEL is executed and bit 31 of word 6 of the SCHIB operand is set to one. The concurrent-sense facility is applicable to subchannels that are operating in command-mode and are associated with channel paths by which the channel subsystem can attempt to retrieve sense information from the device without requiring program intervention.

The concurrent-sense facility is not applicable when a subchannel is operating in transport mode. When a transport-mode IRB is presented that contains the unit-check indication, the associated sense information is found in the TSB for the associated TCW channel program. Therefore, the concurrent sense facility is not necessary with FCX.

Channel-Subsystem Recovery

The channel subsystem provides various methods for extensive detection of malfunctions and other conditions to ensure the integrity of channel-subsystem operation and to achieve automatic recovery of some malfunctions.

The method used to report a particular malfunction or other condition is dependent upon the severity of the malfunction or other condition and the degree to which the malfunction or other condition can be iso-

lated. A malfunction or other condition in the channel subsystem may be indicated to the program by information being stored by one of the following methods:

1. Information is provided in the IRB describing a condition that has been recognized by either the channel subsystem or device that must be brought to the attention of the program. Generally, this information is made available to the program by the execution of TEST SUBCHANNEL, which is usually executed in response to the occurrence of an I/O interruption. (See "Interruption Action" on page 16-6, for a definition of the information stored, as well as "Interruptions" on page 6-1.)
2. Information is provided in a channel report describing a machine malfunction affecting the identified facility associated with the channel-subsystem. This information is made available to the program by the execution of STORE CHANNEL REPORT WORD, which is usually executed in response to the occurrence of a machine-check interruption. (See "Machine-Check Handling" on page 11-1 for a description of the machine-check-interruption mechanism and the contents of the machine-check-interruption code.)
3. Information is provided in a channel report describing a malfunction or other condition affecting a collection of channel-subsystem facilities. This information is made available to the program as indicated in item 2.
4. Information is provided in the machine-check-interruption code (MCIC) describing a malfunction affecting the continued operational integrity of the channel subsystem. (See "Channel-Subsystem Damage" on page 11-16.)
5. Information is provided in the MCIC describing a malfunction affecting the continued operational integrity of a process or of the system. (See "Instruction-Processing Damage" on page 11-14 and "System Damage" on page 11-14.)

Channel reports are used to report malfunctions or other conditions only when the use of the I/O-interruption facility is not appropriate and in preference to reporting channel-subsystem damage, instruction-processing damage, or system damage.

Channel Report

When a malfunction or other condition affecting elements of the channel subsystem has been recognized, a channel report is generated. The performance of recovery actions by the program or by external means may be required to gain recovery from the error condition. The channel report indicates the source of the channel report and the recovery state to the extent necessary for determining the proper recovery action. A channel report consists of one or more channel-report words (CRWs) that have been generated from an analysis of the malfunction or other condition. The inclusion of two or more CRWs within a channel report is indicated by the chaining flag being stored as one in all of the CRWs of the channel report except the last one in the chain.

When a channel report is made pending by the channel subsystem for retrieval and analysis by the program (by means of the execution of STORE CHANNEL REPORT WORD), a malfunction or other condition that affects the normal operation of one or more of the channel-subsystem facilities has been recognized. If the channel report that is made pending is an initial channel report, a machine-check-interruption condition is generated that indicates one or more CRWs are pending at the channel subsystem. A channel report is initial either if it is the first channel report to be generated after the most recent I/O-system reset or if no previously generated reports are pending and the last STORE CHANNEL REPORT WORD instruction that was executed resulted in the setting of condition code 1, indicating that no channel report was pending. When the machine-check interruption occurs and bit 9 of the machine-check-interruption code (channel report pending) is one, a channel report is pending. If the program clears the first CRW of a channel report before the associated machine-check interruption has occurred, some models may reset the machine-check-interruption condition, and the associated machine-check interruption does not occur. A machine-check interruption indicating that a channel report is pending occurs only if the machine-check mask (PSW bit 13) and the channel-report-pending subclass mask, bit 3 of control register 14, are both ones.

If the channel report that is made pending is not an initial channel report, a machine-check-interruption condition is not generated. The CRW that is pre-

sented to the program in response to the first STORE CHANNEL REPORT WORD instruction that is executed after a machine-check interruption may or may not be part of the initial channel report that caused the machine-check condition to be generated. A pending channel-report word is cleared by any CPU executing STORE CHANNEL REPORT WORD, regardless of whether a machine-check interruption has occurred in any CPU. If a CRW is not pending and STORE CHANNEL REPORT WORD is executed, condition code 1 is set, and zeros are stored at the location designated by the second-operand address. During the execution of STORE CHANNEL REPORT WORD as a result of receiving a machine-check interruption, condition code 1 may be set, and zeros may be stored because (1) the related channel report has been cleared by another CPU or (2) a malfunction occurred during the generation of a channel report. In the latter case, if, during a subsequent attempt, a valid channel report can be made pending, an additional machine-check-interruption condition is generated.

When a channel report consists of multiple chained CRWs, they are presented to the program in the same order that they are placed in the chain by the channel subsystem as a result of consecutive executions of STORE CHANNEL REPORT WORD. If, for example, the first CRW of a chain is presented to the program as a result of executing STORE CHANNEL REPORT WORD, the CRW that is presented as a result of the next execution of STORE CHANNEL REPORT WORD is the second CRW of the same chain and not a CRW that is part of another channel report.

Channel reports are not presented to the program in any special order, except for channel reports whose first or only CRW indicates the same reporting-source code and the same reporting-source ID. These channel reports are presented to the program in the same order that they are generated by the channel subsystem, but they are not necessarily presented consecutively. For example, suppose the channel subsystem generates channel reports A, B, and C, in that order. The first CRW of channel reports B and C indicates the same reporting-source code and the same reporting-source ID. Channel report B is presented to the program before channel report C is presented, but channel report A may be presented after channel report B and before channel report C.

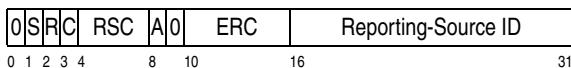
Programming Notes:

1. The information that is provided in a single CRW may be made obsolete by another CRW that is subsequently generated for the same channel-subsystem facility. Therefore, the information that is provided in one channel report should be interpreted in light of the information provided by all of the channel reports that are pending at a given instant.
2. A machine-check-interruption condition is not always generated when a channel report is made pending. The conditions that result in a machine-check-interruption condition being generated are described earlier in this section.
3. After a machine-check interruption has occurred with bit 9 of the machine-check-interruption code set to one, STORE CHANNEL REPORT WORD should be issued repeatedly until all of the pending channel reports have been cleared and condition code 1 has been set.
4. A CRW-overflow condition can occur if the program does not issue successive STORE CHANNEL REPORT WORD instructions in a timely manner after the machine-check interruption occurs.
5. The number of CRWs that can be pending at the same time is model dependent. During the existence of an overflow condition, CRWs that would have otherwise been made pending are lost and are never presented to the program.

Channel-Report Word

The channel-report word (CRW) provides information to the program that can be used to facilitate the recovery of an I/O operation, a device, or some element of the channel subsystem, such as a channel path or subchannel.

The format of the CRW is as follows:



Solicited CRW (S): Bit 1, when one, indicates a solicited CRW. A CRW is considered by the channel subsystem to be solicited if it is made pending as the direct result of some action that is taken by the program. When bit 1 is zero, the CRW is unsolicited and

has been made pending as the result of an action taken by the channel subsystem that is independent of the program.

Overflow (R): Bit 2, when one, indicates that a CRW-overflow condition has been recognized since this CRW became pending and that one or more CRWs have been lost. This bit is one in the CRW that has most recently been set pending when the overflow condition is recognized. When bit 2 is zero, a CRW-overflow condition has not been recognized.

A CRW that is part of a channel report is not made pending, even though the overflow condition does not exist, if an overflow condition prevented a previous CRW of that report from being made pending.

Chaining (C): Bit 3, when one, and when the overflow flag is zero, indicates chaining of associated CRWs. Chaining of CRWs is indicated whenever a malfunction or other condition is described by more than a single CRW. The chaining flag is zero if the channel report is described by a single CRW or if the CRW is the last CRW of a channel report.

The chaining flag is not meaningful if the overflow bit, bit 2, is one.

Reporting-Source Code (RSC): Bits 4-7 identify the channel-subsystem facility that is associated with the channel report. Some facilities are further identified in the reporting-source-identification field (see below). The following combinations of bits identify the facilities:

Bits				Facility
4	5	6	7	
0	0	1	0	Monitoring facility
0	0	1	1	Subchannel
0	1	0	0	Channel Path
1	0	0	1	Configuration-alert facility
1	0	1	1	Channel subsystem

All other bit combinations in the reporting-source-code field are reserved.

Ancillary Report (A): Bit 8, when one, indicates that a malfunction of a system component has occurred that was recognized previously or which has affected the activity of multiple channel-subsystem facilities. When the malfunction affects the activity of multiple channel-subsystem facilities, an

ancillary-report condition is recognized for all of the affected facilities except one. This bit, when zero, indicates that this malfunction of a system component was not recognized previously. This bit is meaningful for all channel reports.

Depending on the model, recognition of an ancillary-report condition may not be provided, or it may not be provided for all system malfunctions that affect channel-subsystem facilities. When ancillary-report recognition is not provided, bit 8 is set to zero.

Error-Recovery Code (ERC): Bits 10-15, when zero, indicate that the channel subsystem has error information regarding the channel-subsystem facility identified in the reporting-source code, and that the program can now request that information. Otherwise, bit positions 10-15 contain the error-recovery code that defines the recovery state of the channel-subsystem facility identified in the reporting-source code. This field, when used in conjunction with the reporting-source code, can be used by the program to determine whether the identified facility has already been recovered and is available for use or whether recovery actions are still required. The following error-recovery codes are defined:

Bits						State
10	11	12	13	14	15	
0	0	0	0	0	0	Event-information pending
0	0	0	0	0	1	Available
0	0	0	0	1	0	Initialized
0	0	0	0	1	1	Temporary error
0	0	0	1	0	0	Installed parameters initialized
0	0	0	1	0	1	Terminal
0	0	0	1	1	0	Permanent error with facility not initialized
0	0	0	1	1	1	Permanent error with facility initialized
0	0	1	0	0	0	Installed parameters modified
0	0	1	0	1	0	Installed parameters restored ¹

Explanation:

- ¹ Installed-parameters restored is presented only if reporting of the installed-parameters-restored CRW is enabled in the restore-subchannel facility.

All other bit combinations in the error-recovery-code field are reserved.

The specific meaning of each error-recovery code depends on the particular reporting-source code that accompanies it in a CRW. The error-recovery codes are defined as follows:

Event-Information Pending: Event information for the identified facility is available for retrieval by the program. This CRW does not indicate the state of the identified facility.

Available: The identified facility is in the same state that the program would expect if the CRW had not been generated.

Initialized: The identified facility is in the same state that existed immediately following the I/O-system reset that was part of the most recent system IPL.

Temporary Error: The identified facility is not operating in a normal manner or has recognized the occurrence of an abnormal event. It is expected that subsequent actions either will restore the facility to normal operation or will record the appropriate information describing the abnormal event.

Installed Parameters Initialized: This state is the same as the initialized state, except that one or more parameters that are associated with the facility and that are not modifiable by the program may have been changed.

On some models that do not implement the restore-subchannel facility, or in the absence of enablement by the program of reporting of the installed-parameters-restored CRW by the restore-subchannel facility, the installed-parameters-initialized CRW may be reported as a result of the channel subsystem's recovery of the facility designated by the reporting-source ID.

Terminal: The identified facility is in a state such that an operation that was in progress can neither be completed nor terminated in the normal manner.

Permanent Error with Facility Not Initialized: The identified facility is in a state of malfunction, and the channel subsystem has not caused a reset function to be performed for that facility.

Permanent Error with Facility Initialized: The identified facility is in a state of malfunction, and the channel subsystem has caused or may have caused a reset function to be performed for that facility.

Installed Parameters Modified: One or more parameters of the specified facility have been changed.

Installed Parameters Restored: This state is the same as the initialized state, except that one or more

parameters that are associated with the facility and that are not modifiable by the program may have been changed from their most recent values back to their initialized values a result of the channel subsystem's recovery of the facility.

Reporting-Source ID (RSID): Bit positions 16-31 contain the reporting-source ID, which may, depending upon the condition that caused the channel report and the reporting-source code, either further identify the affected channel-subsystem facility or provide additional information describing the condition that caused the channel report. The RSID field has the following format as a function of the bit settings of the reporting-source code.

Reporting-Source Code				Reporting-Source ID Bits 16-31			
4	5	6	7				
0	0	1	0	0000	0000	0000	0000
0	0	1	1	xxxx	xxxx	xxxx	xxxx
0	0	1	1	0000	0000	00ss	0000
0	1	0	0	0000	0000	yyyy	yyyy
1	0	0	1	0000	0000	yyyy	yyyy
1	0	1	1	0000	0000	0000	0000

Note:

xxxx	xxxx	xxxx	xxxx	Subchannel Number
ss				Subchannel-set ID when the MSS facility is installed and the CRW is chained immediately following a CRW for a subchannel.
yyyy	yyyy			Channel-path ID (CHPID)

When the MSS facility is installed an additional CRW is chained to every channel report for subchannels. The content of this chained CRW is identical to the original CRW except for the RSID field. In this chained CRW, bit positions 10-11 of the RSID field contain the SSID identifying the subchannel set.

Restore-Subchannel Facility

The restore-subchannel facility may be available on a model implementing z/Architecture. This facility provides the means for the channel subsystem to recover a damaged subchannel and report the recovery to the program by means of a CRW. When the facility is installed and reporting of the installed-parameters-restored CRW is enabled in the restore-subchannel facility, the channel subsystem may

recover the damaged subchannel and make a CRW pending that indicates installed-parameters restored. When the facility is installed and reporting of the installed-parameters-restored CRW is not enabled, the channel subsystem may recover the damaged subchannel and make a CRW pending that indicates installed-parameters initialized.

In some models that do not implement the restore-subchannel facility, the channel subsystem may nevertheless recover a damaged subchannel, and make a CRW pending that indicates installed-parameters initialized.

Extended-Subchannel-Logout Facility

When the extended-subchannel-logout facility is installed and enabled, all fibre-channel (FC) channel paths can create extended-subchannel logouts, under applicable error conditions, and generate I/O interruptions that indicate such a logout is pending retrieval. The extended-subchannel logouts are stored in and retrieved from a model-dependent number of extended-subchannel-logout buffers associated with each channel path supported by the facility.

When TEST SUBCHANNEL for a subchannel that specifies an FC channel path stores an interruption-response block (IRB) containing a format-0 ESW in which the extended-subchannel-logout-pending (E) bit is 1, an extended-subchannel logout is pending for the specified subchannel. Words 2-3 of the format-0 ESW contain the extended-subchannel-logout descriptor (ESLD) that identifies the logout and the channel path for which the logout is pending.

If all of the buffers for a given channel contain pending extended-subchannel logouts, the logouts will remain pending for a model-dependent time interval. If, during that time interval, a subsequent condition occurs that warrants a new extended-subchannel logout, that logout is not made and is lost. If the time interval has been exceeded and the program has not used the ESLD to retrieve a pending logout, the logout may be removed from the buffer or the buffer may be reused to contain a subsequent logout. When a logout is removed from a buffer and the buffer is not reused, the ESLD is no longer recognized as a valid logout identifier. When a buffer is reused, the ESLD may be recognized as a valid

logout identifier albeit for a logout different from the logout for which the original I/O interruption was generated.

Channel-Subsystem-I/O-Priority Facility

The channel-subsystem-I/O-priority facility provides a means by which the program can establish a priority relationship, at the channel subsystem, among the subchannels that are placed into the start-pending state when START SUBCHANNEL is executed and condition code 0 is indicated. For I/O-subchannels that are configured to fibre-channel channel paths (FICON and FICON-converted channel paths), it also provides a means by which the program can establish a priority relationship for I/O operations at the fibre-channel-attached control units.

The program assigns the desired channel-subsystem priority and control-unit priority by specifying the desired priority numbers in the ORB extension when START SUBCHANNEL is executed.

The channel-subsystem-priority number specified in the ORB is used by the channel subsystem to determine the order in which start-pending and resume-pending subchannels are selected when the channel subsystem attempts to initiate a start function or a resume function. See the section “Start Function and Resume Function” on page 15-20 for details about these functions. In general, I/O subchannels that are in the start-pending or resume-pending state and have a higher priority number are selected for start-function or resume-function initiation by the channel subsystem before start-pending or resume-pending subchannels that have a lower priority number. The specific priority selection algorithm used by the channel subsystem for this purpose depends on the model. Additionally, the channel subsystem also applies a fairness selection algorithm in conjunction with the priority selection algorithm when selecting I/O subchannels. The specific fairness selection algorithm also depends on the model. For all models, the channel-subsystem priority and fairness selection algorithms are always applied to I/O subchannels that are either start pending or resume pending. Some models may also apply both algorithms to subchannels that are either clear pending or halt pending. See a model's System-Library publication for a

description of the priority and fairness selection algorithms that the model provides and whether these algorithms are also applied to clear-pending or halt-pending subchannels.

The control-unit-priority number specified in the ORB is used by control units attached to fibre-channel channel paths in order to determine the priority of the execution of CCWs at the control unit. See “Control-Unit (CU) Priority:” on page 15-29 for additional information.

Number of Channel-Subsystem-Priority Levels

Depending on the model, fewer than 256 channel-subsystem-priority levels may be provided by the channel subsystem. Each priority level that the model provides is designated by an eight-bit unsigned binary integer. The lowest provided channel-subsystem-priority level is designated by the integer 0, and each succeeding higher priority level is designated by the next-higher sequential integer. For example, if the model provides 16 priority levels, they are numbered 0-15, respectively, from the lowest priority level to the highest priority level.

Multiple-Subchannel-Set Facility

The multiple-subchannel-set (MSS) facility increases the maximum number of subchannels that can be configured to a program. When the MSS facility is not provided, a single set of subchannels, in the range 0-65,535 may be provided for a program. When the MSS facility is provided, a maximum of four sets of subchannels may be provided for a program. Each subchannel set provides from one to 64K subchannels in the range 0 to-65,535. A two-bit unsigned binary integer, in the range of 0-3, called the subchannel-set identifier (SSID) is used to specify each provided subchannel set. The default subchannel set for a program is SSID 0. The initialized state of the MSS facility is disabled, in this state the MSS facility functions are not provided to programs executing in logical partitions.

When the MSS facility is provided, a subchannel is specified by a unique address formed by the concatenation of:

1. The subchannel-set identifier (SSID) of the subchannel-image set to which the designated subchannel-image is configured,
2. The subchannel number of the subchannel-image being accessed.

Chapter 18. Hexadecimal-Floating-Point Instructions

HFP Arithmetic	18-1	LOAD AND TEST	18-13
HFP Number Representation	18-1	LOAD COMPLEMENT	18-14
Normalization	18-2	LOAD FP INTEGER	18-15
HFP Data Formats	18-2	LOAD LENGTHENED	18-15
HFP Short Format	18-4	LOAD NEGATIVE	18-16
HFP Long Format	18-4	LOAD POSITIVE	18-16
HFP Extended Format	18-4	LOAD ROUNDED	18-17
Instructions	18-5	MULTIPLY	18-17
ADD NORMALIZED	18-8	MULTIPLY AND ADD	18-19
ADD UNNORMALIZED	18-9	MULTIPLY AND SUBTRACT	18-19
COMPARE	18-10	MULTIPLY AND ADD UNNORMALIZED	18-20
CONVERT FROM FIXED	18-11	MULTIPLY UNNORMALIZED	18-22
CONVERT TO FIXED	18-11	SQUARE ROOT	18-23
DIVIDE	18-12	SUBTRACT NORMALIZED	18-24
HALVE	18-13	SUBTRACT UNNORMALIZED	18-25

HFP Arithmetic

HFP Number Representation

A hexadecimal-floating-point (HFP) number consists of a sign bit, a hexadecimal fraction, and an unsigned seven-bit binary integer called the characteristic. The characteristic represents a signed exponent and is obtained by adding 64 to the exponent value (excess-64 notation). The range of the characteristic is 0 to 127, which corresponds to an exponent range of -64 to +63. The magnitude of an HFP number is the product of its fraction and the number 16 raised to the power of the exponent that is represented by its characteristic. The number is positive or negative depending on whether the sign bit is zero or one, respectively.

The fraction of an HFP number is treated as a hexadecimal number because it is considered to be multiplied by a number which is a power of 16. The name, fraction, indicates that the radix point is assumed to be immediately to the left of the leftmost fraction digit.

When an HFP operation would cause the result exponent to exceed 63, the characteristic wraps around from 127 to 0, and an HFP-exponent-overflow condition exists. The result characteristic is then too small by 128. When an operation would cause the exponent to be less than -64, the characteristic

wraps around from 0 to 127, and an HFP-exponent-underflow condition exists. The result characteristic is then too large by 128, except that a zero characteristic is produced when a true zero is forced.

A true zero is an HFP number with a zero characteristic and zero fraction. A true zero may arise as the normal result of an arithmetic operation because of the particular magnitude of the operands. For HFP operations, the result is forced to be a positive true zero when:

1. An HFP exponent underflow occurs and the HFP-exponent-underflow mask bit in the PSW is zero.
2. The result fraction of a normalized or unnormalized addition or subtraction operation is zero and the HFP-significance mask bit in the PSW is zero.
3. The operand of the CONVERT FROM FIXED instruction is zero.
4. The dividend in the DIVIDE instruction has a zero fraction.
5. The operand of the HALVE, LOAD FP INTEGER, or SQUARE ROOT instruction has a zero fraction.
6. One or both operands of a multiplication operation has a zero fraction.

When a program interruption for HFP exponent underflow occurs, a true zero is not forced; instead, the fraction and sign remain correct, and the characteristic is too large by 128. When a program interruption for HFP significance occurs, the fraction remains zero, the sign is positive, and the characteristic remains correct.

The sign of a sum, difference, product, quotient, square root, the result of CONVERT FROM FIXED, or the result of LOAD FP INTEGER with a zero fraction is positive. The sign for a zero fraction resulting from other HFP operations is established from the operand sign, the same as for nonzero fractions.

Normalization

A quantity can be represented with the greatest precision by an HFP number of a given fraction length when that number is normalized. A normalized HFP number has a nonzero leftmost hexadecimal fraction digit. If one or more leftmost fraction digits are zeros, the number is said to be unnormalized.

Unnormalized numbers are normalized by shifting the fraction left, one digit at a time, until the leftmost hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A number with a zero fraction cannot be normalized; either its characteristic remains unchanged or its characteristic is made zero when the result is forced to be a true zero.

Addition and subtraction with extended operands, as well as the MULTIPLY, DIVIDE, CONVERT FROM FIXED, HALVE, LOAD FP INTEGER, and SQUARE ROOT operations, are performed only with normalization. Addition and subtraction with short or long operands may be specified as either normalized or unnormalized. For all other operations, the result is produced without normalization.

With unnormalized operations, leftmost zeros in the result fraction are not eliminated. The result may or

may not be in normalized form, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. The operands for multiply, divide, and square-root operations are normalized before the arithmetic process. For other normalized operations, normalization takes place when the intermediate arithmetic result is changed to the final result.

When the intermediate result of addition, subtraction, or rounding causes the fraction to overflow, the fraction is shifted right by one hexadecimal-digit position, and the value one is supplied to the vacated leftmost digit position. The fraction is then truncated to the final result length, while the characteristic is increased by one. This adjustment is made for both normalized and unnormalized operations.

Figure 18-1 on page 18-3 summarizes, for all instructions producing HFP results, the handling of zero results and whether normalization occurs for nonzero results.

Programming Note: Up to three leftmost bits of the fraction of a normalized number may be zeros, since the nonzero test applies to the entire leftmost hexadecimal digit.

HFP Data Formats

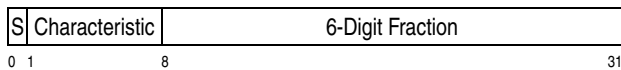
HFP numbers have a 32-bit (short) format, a 64-bit (long) format, or a 128-bit (extended) format. Numbers in the short and long formats may be designated as operands both in storage and in the floating-point registers, whereas operands having the extended format can be designated only in the floating-point registers.

In all formats, the first bit (bit 0) is the sign bit (S). The next seven bits are the characteristic. In the short and long formats, the remaining bits constitute the fraction, which consists of six or 14 hexadecimal digits, respectively.

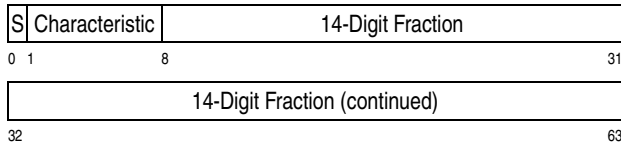
Instruction	Nonzero Result Normalized	Zero Result Forced to True Zero		Zero Result Made Positive	
		Short and Long	Extended	Short and Long	Extended
ADD NORMALIZED	Yes	Y/N	Y/N	Yes	Yes
ADD UNNORMALIZED	No	Y/N	–	Yes	–
CONVERT BFP TO HFP ¹	Yes	Yes	–	No	–
CONVERT FROM FIXED	Yes	Yes	Yes	Yes	Yes
DIVIDE	Yes	Yes	Yes	Yes	Yes
HALVE	Yes	Yes	–	Yes	–
LOAD ¹	No	No	No	No	No
LOAD AND TEST	No	No	Yes	No	No
LOAD COMPLEMENT	No	No	Yes	No	No
LOAD FP INTEGER	Yes	Yes	Yes	Yes	Yes
LOAD LENGTHENED	No	No	Yes	No	No
LOAD NEGATIVE	No	No	Yes	No	No
LOAD POSITIVE	No	No	Yes	Yes	Yes
LOAD ROUNDED	No	No	–	No	–
LOAD ZERO ¹	–	Yes	Yes	Yes	Yes
MULTIPLY	Yes	Yes	Yes	Yes	Yes
MULTIPLY AND ADD	Yes	Yes	–	Yes	–
MULTIPLY AND ADD UNNORMALIZED	No	–	No	–	No
MULTIPLY AND SUBTRACT	Yes	Yes	–	Yes	–
MULTIPLY UNNORMALIZED	No	–	No	–	No
PERFORM FLOATING-POINT OPERATION ¹	Yes	Yes	Yes	No	No
SQUARE ROOT	Yes	Yes	Yes	Yes	Yes
STORE ¹	No	No	–	No	–
SUBTRACT NORMALIZED	Yes	Y/N	Y/N	Yes	Yes
SUBTRACT UNNORMALIZED	No	Y/N	–	Yes	–
Explanation: - Not applicable. ¹ Floating-point-support instruction. Y/N When the HFP-significance mask bit (PSW bit 23) is zero, a true zero is forced. When the HFP-significance mask bit is one, the characteristic remains unchanged, and a program interruption for HFP significance occurs.					

Figure 18-1. Normalization and Zero Handling for Instructions with HFP Results

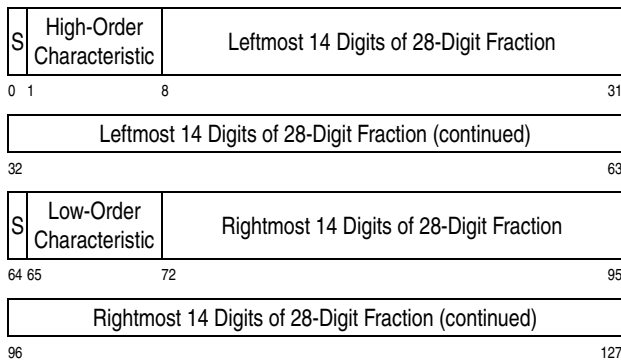
HFP Short Format



HFP Long Format



HFP Extended Format



An extended HFP number has a 28-digit fraction and consists of two long HFP numbers that are called the high-order and low-order parts. The high-order part may be any long HFP number. The fraction of the high-order part contains the leftmost 14 hexadecimal digits of the 28-digit fraction. The characteristic and sign of the high-order part are the characteristic and sign of the extended HFP number. If the high-order part is normalized, the extended number is considered normalized. The fraction of the low-order part contains the rightmost 14 digits of the 28-digit fraction. The sign and characteristic of the low-order part of an extended operand are ignored.

When a result is generated in the extended format and placed in a register pair, the sign of the low-order part is made the same as that of the high-order part, and, unless the result is a true zero, the low-order characteristic is made 14 less than the high-order characteristic. When the subtraction of 14 would cause the low-order characteristic to become less than zero, the characteristic is made 128 greater than its correct value. (Thus, the subtraction is performed modulo 128.) HFP exponent underflow is indicated only when the high-order characteristic underflows.

When an extended result is made a true zero, both the high-order and low-order parts are made a true zero.

The range covered by the magnitude (M) of a normalized HFP number depends on the format.

In the short format:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \leq M \leq (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \leq M \leq (1 - 16^{-28}) \times 16^{63}$$

In all formats, approximately:

$$5.4 \times 10^{-79} \leq M \leq 7.2 \times 10^{75}$$

Although the final result of an HFP operation has six hexadecimal fraction digits in the short format, 14 fraction digits in the long format, and 28 fraction digits in the extended format, intermediate results have one additional hexadecimal digit on the right. This digit is called the guard digit. The guard digit may increase the precision of the final result because it participates in addition, subtraction, and comparison operations and in the left shift that occurs during normalization.

The entire set of HFP operations with normalized results is available for short, long, and extended operands in register-register versions; and for short and long operands in register-storage versions. Most instructions generate a result that has the same format as the source operands, except that there are multiplication operations which can generate a long product from short operands or an extended product from long operands. Other exceptions are instructions which convert operands from one floating-point format to another or between floating-point and fixed-point (binary-integer) formats.

Programming Notes:

1. In the absence of an HFP exponent overflow or HFP exponent underflow, the long HFP number constituting the low-order part of an extended result correctly expresses the value of the low-order part of the extended result when the characteristic of the high-order part is 14 or higher. This applies also when the result is a true zero. When the high-order characteristic is less than 14 but the number is not a true zero, the low-

order part, when considered as a long HFP number, does not express the correct characteristic value.

2. The entire fraction of an extended result participates in normalization. The low-order part alone may or may not appear to be a normalized long HFP number, depending on whether the 15th digit of the normalized 28-digit fraction is nonzero or zero.

Instructions

The HFP instructions and their mnemonics and operation codes are listed in Figure 18-2 on page 18-5. The figure indicates, in the column labeled “Characteristics”, the instruction format, when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

All HFP instructions are subject to the AFP-register-control bit, bit 45 of control register 0. The AFP-register-control bit must be one when an AFP register is specified as an operand location; otherwise, an AFP-register data exception, DXC 1, is recognized.

Mnemonics for the HFP instructions have an R as the last letter when the instruction is in the RR, RRE, or RRF format. Certain letters are used for HFP instructions to represent operand-format length and normalization, as follows:

- F Thirty-two-bit fixed point
- G Sixty-four-bit fixed point
- D Long normalized
- E Short normalized
- U Short unnormalized
- W Long unnormalized
- X Extended normalized

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand and designation for the assembler language are shown with each instruction. For a register-to-register operation using COMPARE (short), for example, CER is the mnemonic and R₁,R₂ the operand designation.

Programming Notes:

1. The following additional HFP instructions are available when the HFP-multiply-and-add/subtract facility is installed:
 - MULTIPLY AND ADD (MAD, MADR, MAE, MAER)
 - MULTIPLY AND SUBTRACT (MSD, MSDR, MSE, MSER)
2. The following additional HFP instructions are available when the HFP-unnormalized-extensions facility is installed:
 - MULTIPLY AND ADD UNNORMALIZED (MAY, MAYH, MAYHR, MAYL, MAYLR, MAYR)
 - MULTIPLY UNNORMALIZED (MY, MYH, MYHR, MYL, MYLR, MYR)

Name	Mne- monic	Characteristics							Op Code	Page		
ADD NORMALIZED (extended HFP)	AXR	RR	C	SP	Da	EU	EO	LS		B ₂	36	18-8
ADD NORMALIZED (long HFP)	AD	RX-a	C	A	Da	EU	EO	LS		B ₂	6A	18-8
ADD NORMALIZED (long HFP)	ADR	RR	C		Da	EU	EO	LS			2A	18-8
ADD NORMALIZED (short HFP)	AE	RX-a	C	A	Da	EU	EO	LS		B ₂	7A	18-8
ADD NORMALIZED (short HFP)	AER	RR	C		Da	EU	EO	LS			3A	18-8
ADD UNNORMALIZED (long HFP)	AW	RX-a	C	A	Da		EO	LS		B ₂	6E	18-9
ADD UNNORMALIZED (long HFP)	AWR	RR	C		Da		EO	LS			2E	18-9
ADD UNNORMALIZED (short HFP)	AU	RX-a	C	A	Da		EO	LS		B ₂	7E	18-9
ADD UNNORMALIZED (short HFP)	AUR	RR	C		Da		EO	LS			3E	18-9
COMPARE (extended HFP)	CXR	RRE	C	SP	Da						B369	18-10
COMPARE (long HFP)	CD	RX-a	C	A	Da					B ₂	69	18-10
COMPARE (long HFP)	CDR	RR	C		Da						29	18-10
COMPARE (short HFP)	CE	RX-a	C	A	Da					B ₂	79	18-10
COMPARE (short HFP)	CER	RR	C		Da						39	18-10
CONVERT FROM FIXED (32 to extended HFP)	CXFR	RRE		SP	Da						B3B6	18-10

Figure 18-2. Summary of HFP Instructions (Part 1 of 4)

Name	Mne- monic	Characteristics						Op Code	Page
CONVERT FROM FIXED (32 to long HFP)	CDFR	RRE			Da			B3B5	18-10
CONVERT FROM FIXED (32 to short HFP)	CEFR	RRE			Da			B3B4	18-10
CONVERT FROM FIXED (64 to extended HFP)	CXGR	RRE	N	SP	Da			B3C6	18-10
CONVERT FROM FIXED (64 to long HFP)	CDGR	RRE	N		Da			B3C5	18-10
CONVERT FROM FIXED (64 to short HFP)	CEGR	RRE	N		Da			B3C4	18-10
CONVERT TO FIXED (extended HFP to 32)	CFXR	RRF-e	C	SP	Da			B3BA	18-11
CONVERT TO FIXED (extended HFP to 64)	CGXR	RRF-e	C N	SP	Da			B3CA	18-11
CONVERT TO FIXED (long HFP to 32)	CFDR	RRF-e	C	SP	Da			B3B9	18-11
CONVERT TO FIXED (long HFP to 64)	CGDR	RRF-e	C N	SP	Da			B3C9	18-11
CONVERT TO FIXED (short HFP to 32)	CFER	RRF-e	C	SP	Da			B3B8	18-11
CONVERT TO FIXED (short HFP to 64)	CGER	RRF-e	C N	SP	Da			B3C8	18-11
DIVIDE (extended HFP)	DXR	RRE		SP	Da EU EO FK			B22D	18-12
DIVIDE (long HFP)	DD	RX-a		A	Da EU EO FK		B ₂	6D	18-12
DIVIDE (long HFP)	DDR	RR			Da EU EO FK			2D	18-11
DIVIDE (short HFP)	DE	RX-a		A	Da EU EO FK		B ₂	7D	18-12
DIVIDE (short HFP)	DER	RR			Da EU EO FK			3D	18-11
Explanation:									
A	Access exceptions for logical addresses.								
B ₂	B ₂ field designates an access register in the access-register mode.								
C	Condition code is set.								
Da	AFP-register data exception.								
EO	HFP-exponent-overflow exception.								
EU	HFP-exponent-underflow exception.								
FK	HFP-divide exception.								
HM	HFP-multiply-add/subtract facility.								
LS	HFP-significance exception.								
N	Instruction is new in z/Architecture as compared to ESA/390.								
RR	RR instruction format.								
RRD	RRD instruction format.								
RRE	RRE instruction format.								
RRF	RRF instruction format.								
RX	RX instruction format.								
RXE	RXE instruction format.								
SP	Specification exception.								
SQ	HFP-square-root exception.								
UE	HFP unnormalized-extensions facility.								
HALVE (long HFP)	HDR	RR			Da EU			24	18-13
HALVE (short HFP)	HER	RR			Da EU			34	18-13
LOAD AND TEST (extended HFP)	LTXR	RRE	C	SP	Da			B362	18-13
LOAD AND TEST (long HFP)	LTDR	RR	C		Da			22	18-13
LOAD AND TEST (short HFP)	LTER	RR	C		Da			32	18-13
LOAD COMPLEMENT (extended HFP)	LCXR	RRE	C	SP	Da			B363	18-14
LOAD COMPLEMENT (long HFP)	LCDR	RR	C		Da			23	18-14
LOAD COMPLEMENT (short HFP)	LCER	RR	C		Da			33	18-14
LOAD FP INTEGER (extended HFP)	FIXR	RRE		SP	Da			B367	18-14
LOAD FP INTEGER (long HFP)	FIDR	RRE			Da			B37F	18-14
LOAD FP INTEGER (short HFP)	FIER	RRE			Da			B377	18-14
LOAD LENGTHENED (long to extended HFP)	LXD	RXE		A SP	Da		B ₂	ED25	18-15
LOAD LENGTHENED (long to extended HFP)	LXDR	RRE		SP	Da			B325	18-15
LOAD LENGTHENED (short to extended HFP)	LXE	RXE		A SP	Da		B ₂	ED26	18-15
LOAD LENGTHENED (short to extended HFP)	LXER	RRE		SP	Da			B326	18-15
LOAD LENGTHENED (short to long HFP)	LDE	RXE		A	Da		B ₂	ED24	18-15
LOAD LENGTHENED (short to long HFP)	LDER	RRE			Da			B324	18-15

Figure 18-2. Summary of HFP Instructions (Part 2 of 4)

Name	Mnemonic	Characteristics						Op Code	Page
LOAD NEGATIVE (extended HFP)	LNXR	RRE C	SP	Da				B361	18-15
LOAD NEGATIVE (long HFP)	LNDR	RR C		Da				21	18-15
LOAD NEGATIVE (short HFP)	LNER	RR C		Da				31	18-15
LOAD POSITIVE (extended HFP)	LPXR	RRE C	SP	Da				B360	18-16
LOAD POSITIVE (long HFP)	LPDR	RR C		Da				20	18-16
LOAD POSITIVE (short HFP)	LPER	RR C		Da				30	18-16
LOAD ROUNDED (extended to long HFP)	LDXR	RR	SP	Da EO				25	18-16
LOAD ROUNDED (extended to long HFP)	LRDR	RR	SP	Da EO				25	18-16
LOAD ROUNDED (extended to short HFP)	LEXR	RRE	SP	Da EO				B366	18-16
LOAD ROUNDED (long to short HFP)	LEDR	RR		Da EO				35	18-16
LOAD ROUNDED (long to short HFP)	LRER	RR		Da EO				35	18-16
MULTIPLY & ADD UNNORMALIZED (long to ext. HFP)	MAY	RXF UE	A	Da			B ₂	ED3A	18-20
MULTIPLY & ADD UNNORMALIZED (long to ext. HFP)	MAYR	RRD UE		Da				B33A	18-20
MULTIPLY (extended HFP)	MXR	RR	SP	Da EU EO				26	18-17
MULTIPLY (long HFP)	MD	RX-a	A	Da EU EO			B ₂	6C	18-17
MULTIPLY (long HFP)	MDR	RR		Da EU EO				2C	18-17
MULTIPLY (long to extended HFP)	MXD	RX-a	A SP	Da EU EO			B ₂	67	18-17
MULTIPLY (long to extended HFP)	MXDR	RR	SP	Da EU EO				27	18-17
MULTIPLY (short HFP)	MEE	RXE	A	Da EU EO			B ₂	ED37	18-17
MULTIPLY (short HFP)	MEER	RRE		Da EU EO				B337	18-17
MULTIPLY (short to long HFP)	MDE	RX-a	A	Da EU EO			B ₂	7C	18-17
MULTIPLY (short to long HFP)	MDER	RR		Da EU EO				3C	18-17
MULTIPLY (short to long HFP)	ME	RX-a	A	Da EU EO			B ₂	7C	18-17
MULTIPLY (short to long HFP)	MER	RR		Da EU EO				3C	18-17
MULTIPLY AND ADD (long HFP)	MAD	RXF HM	A	Da EU EO			B ₂	ED3E	18-19
MULTIPLY AND ADD (long HFP)	MADR	RRD HM		Da EU EO				B33E	18-18
MULTIPLY AND ADD (short HFP)	MAE	RXF HM	A	Da EU EO			B ₂	ED2E	18-19
MULTIPLY AND ADD (short HFP)	MAER	RRD HM		Da EU EO				B32E	18-18
MULTIPLY AND ADD UNNRM. (long to ext. high HFP)	MAYH	RXF UE	A	Da			B ₂	ED3C	18-20
MULTIPLY AND ADD UNNRM. (long to ext. high HFP)	MAYHR	RRD UE		Da				B33C	18-20
MULTIPLY AND ADD UNNRM. (long to ext. low HFP)	MAYL	RXF UE	A	Da			B ₂	ED38	18-20
MULTIPLY AND ADD UNNRM. (long to ext. low HFP)	MAYLR	RRD UE		Da				B338	18-20
MULTIPLY AND SUBTRACT (long HFP)	MSD	RXF HM	A	Da EU EO			B ₂	ED3F	18-19
MULTIPLY AND SUBTRACT (long HFP)	MSDR	RRD HM		Da EU EO				B33F	18-19
MULTIPLY AND SUBTRACT (short HFP)	MSE	RXF HM	A	Da EU EO			B ₂	ED2F	18-19
MULTIPLY AND SUBTRACT (short HFP)	MSER	RRD HM		Da EU EO				B32F	18-19
MULTIPLY UNNORM. (long to ext. high HFP)	MYH	RXF UE	A	Da			B ₂	ED3D	18-21
MULTIPLY UNNORM. (long to ext. high HFP)	MYHR	RRD UE		Da				B33D	18-21
MULTIPLY UNNORM. (long to ext. low HFP)	MYL	RXF UE	A	Da			B ₂	ED39	18-21
MULTIPLY UNNORM. (long to ext. low HFP)	MYLR	RRD UE		Da				B339	18-21
MULTIPLY UNNORMALIZED (long to ext. HFP)	MY	RXF UE	A SP	Da			B ₂	ED3B	18-21
MULTIPLY UNNORMALIZED (long to ext. HFP)	MYR	RRD UE	SP	Da				B33B	18-21
SQUARE ROOT (extended HFP)	SQXR	RRE	SP	Da SQ				B336	18-23
SQUARE ROOT (long HFP)	SQD	RXE	A	Da SQ			B ₂	ED35	18-23
SQUARE ROOT (long HFP)	SQDR	RRE		Da SQ				B244	18-23
SQUARE ROOT (short HFP)	SQE	RXE	A	Da SQ			B ₂	ED34	18-23
SQUARE ROOT (short HFP)	SQER	RRE		Da SQ				B245	18-23
SUBTRACT NORMALIZED (extended HFP)	SXR	RR C	SP	Da EU EO LS				37	18-24
SUBTRACT NORMALIZED (long HFP)	SD	RX-a C	A	Da EU EO LS			B ₂	6B	18-24
SUBTRACT NORMALIZED (long HFP)	SDR	RR C		Da EU EO LS				2B	18-24

Figure 18-2. Summary of HFP Instructions (Part 3 of 4)

Name	Mne- monic	Characteristics							Op Code	Page
SUBTRACT NORMALIZED (short HFP)	SE	RX-a	C	A	Da	EU	EO	LS	B ₂	7B 18-24
SUBTRACT NORMALIZED (short HFP)	SER	RR	C		Da	EU	EO	LS	3B	18-24
SUBTRACT UNNORMALIZED (long HFP)	SW	RX-a	C	A	Da		EO	LS	B ₂	6F 18-24
SUBTRACT UNNORMALIZED (long HFP)	SWR	RR	C		Da		EO	LS	2F	18-24
SUBTRACT UNNORMALIZED (short HFP)	SU	RX-a	C	A	Da		EO	LS	B ₂	7F 18-24
SUBTRACT UNNORMALIZED (short HFP)	SUR	RR	C		Da		EO	LS	3F	18-24

Figure 18-2. Summary of HFP Instructions (Part 4 of 4)

ADD NORMALIZED

Mnemonic1 R₁,R₂ [RR]

Op Code	R ₁	R ₂
0	8	12 15

Mnemonic1	Op Code	Operands
AER	'3A'	Short HFP
ADR	'2A'	Long HFP
AXR	'36'	Extended HFP

Mnemonic2 R₁,D₂(X₂,B₂) [RX-a]

Op Code	R ₁	X ₂	B ₂	D ₂
0	8	12	16	20 31

Mnemonic2	Op Code	Operands
AE	'7A'	Short HFP
AD	'6A'	Long HFP

The second operand is added to the first operand, and the normalized sum is placed at the first-operand location.

Addition of two HFP numbers consists in characteristic comparison, fraction alignment, and signed fraction addition. The characteristics of the two operands are compared, and the fraction accompanying the smaller characteristic is aligned with the other fraction by a right shift, with its characteristic increased by one for each hexadecimal digit of shift until the two characteristics agree.

When a fraction is shifted right during alignment, the leftmost hexadecimal digit shifted out is retained as a guard digit. The fraction that is not shifted is considered to be extended with a zero in the guard-digit position. When no alignment shift occurs, both operands are considered to be extended with zeros in the guard-digit position. The fractions with signs are then added algebraically to form a signed intermediate sum.

The intermediate-sum fraction consists of seven (short format), 15 (long format), or 29 (extended format) hexadecimal digits, including the guard digit, and a possible carry. If a carry is present, the sum is shifted right one digit position so that the carry becomes the leftmost digit of the fraction, and the characteristic is increased by one.

If the addition produces no carry, the intermediate-sum fraction is shifted left as necessary to eliminate any leading hexadecimal zero digits resulting from the addition, provided the fraction is not zero. Zeros are supplied to the vacated rightmost digits, and the characteristic is reduced by the number of hexadecimal digits of shift. The fraction thus normalized is then truncated on the right to six (short format), 14 (long format), or 28 (extended format) hexadecimal digits. In the extended format, a characteristic is generated for the low-order part, which is 14 less than the high-order characteristic.

The sign of the sum is determined by the rules of algebra, unless all digits of the intermediate-sum fraction are zero, in which case the sign is made plus.

An HFP-exponent-overflow exception exists when a carry from the leftmost position of the intermediate-sum fraction would cause the characteristic of the normalized sum to exceed 127. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct. For extended results, the characteristic of the low-order part remains correct.

An HFP-exponent-underflow exception exists when the characteristic of the normalized sum would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow

occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero. For extended results, HFP exponent underflow is not recognized when the low-order characteristic is less than zero but the high-order characteristic is equal to or greater than zero.

The result fraction is zero when the intermediate-sum fraction, including the guard digit, is zero. With a zero result fraction, the action depends on the setting of the HFP-significance mask bit in the PSW. If the HFP-significance mask bit in the PSW is one, no normalization occurs, the intermediate and final result characteristics are the same, and a program interruption for HFP significance occurs. If the HFP-significance mask bit in the PSW is zero, the program interruption does not occur; instead, the result is made a positive true zero.

For AXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 --

Program Exceptions:

- Access (fetch, operand 2 of AE and AD only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP exponent underflow
- HFP significance
- Specification (AXR only)

Programming Notes:

1. An example of the use of the ADD NORMALIZED instruction (AE) is given in Appendix A.
2. Interchanging the two operands in an HFP addition does not affect the value of the sum.
3. The ADD NORMALIZED instruction normalizes the sum but not the operands. Thus, if one or both operands are unnormalized, precision may be lost during fraction alignment.

ADD UNNORMALIZED

Mnemonic1 R₁,R₂ [RR]

Op Code	R ₁	R ₂
0	8	12 15

Mnemonic1	Op Code	Operands
AUR	'3E'	Short HFP
AWR	'2E'	Long HFP

Mnemonic2 R₁,D₂(X₂,B₂) [RX-a]

Op Code	R ₁	X ₂	B ₂	D ₂
0	8	12	16	20 31

Mnemonic2	Op Code	Operands
AU	'7E'	Short HFP
AW	'6E'	Long HFP

The second operand is added to the first operand, and the unnormalized sum is placed at the first-operand and location.

The execution of ADD UNNORMALIZED is identical to that of ADD NORMALIZED, except that:

1. When no carry is present after the addition, the intermediate-sum fraction is truncated to the proper result-fraction length without a left shift to eliminate leading hexadecimal zeros and without the corresponding reduction of the characteristic.
2. HFP exponent underflow cannot occur.
3. The guard digit does not participate in the recognition of a zero result fraction. A zero result fraction is recognized when the fraction (that is, the intermediate-sum fraction, excluding the guard digit) is zero.

Resulting Condition Code:

- 0 Result fraction zero
- 1 Result less than zero
- 2 Result greater than zero
- 3 --

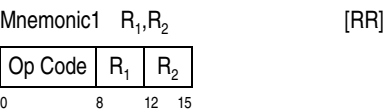
Program Exceptions:

- Access (fetch, operand 2 of AU and AW only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP significance

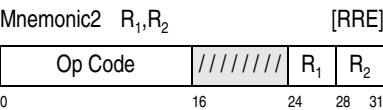
Programming Notes:

- 1. An example of the use of the ADD UNNORMALIZED instruction (AU) is given in Appendix A.
- 2. Except when the result is made a true zero, the characteristic of the result of ADD UNNORMALIZED is equal to the greater of the two operand characteristics, increased by one if the fraction addition produced a carry, or set to zero if HFP exponent overflow occurred.

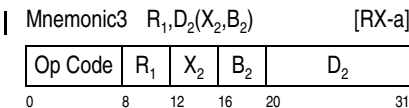
COMPARE



Mnemonic1	Op Code	Operands
CER	'39'	Short HFP
CDR	'29'	Long HFP



Mnemonic2	Op Code	Operands
CXR	'B369'	Extended HFP



Mnemonic3	Op Code	Operands
CE	'79'	Short HFP
CD	'69'	Long HFP

The first operand is compared with the second operand, and the condition code is set to indicate the result.

The comparison is algebraic and follows the procedure for normalized subtraction, except that the difference is discarded after setting the condition code and both operands remain unchanged. When the difference, including the guard digit, is zero, the operands are equal. When a nonzero difference is positive or negative, the first operand is high or low, respectively.

An HFP-exponent-overflow, HFP-exponent-underflow, or HFP-significance exception cannot occur.

For CXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 --

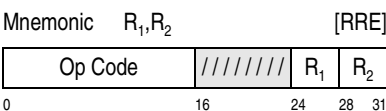
Program Exceptions:

- Access (fetch, operand 2 of CE and CD only)
- Data with DXC 1, AFP register
- Specification (CXR only)

Programming Notes:

- 1. Examples of the use of the COMPARE instruction (CDR) are given in Appendix A.
- 2. An exponent inequality alone is not sufficient to determine the inequality of two operands with the same sign, because the fractions may have different numbers of leading hexadecimal zeros.
- 3. Numbers with zero fractions compare equal even when they differ in sign or characteristic.

CONVERT FROM FIXED



Mnemonic	Op Code	Operands
CEFR	'B3B4'	32-bit binary-integer operand, short HFP result
CDFR	'B3B5'	32-bit binary-integer operand, long HFP result
CXFR	'B3B6'	32-bit binary-integer operand, extended HFP result
CEGR	'B3C4'	64-bit binary-integer operand, short HFP result
CDGR	'B3C5'	64-bit binary-integer operand, long HFP result
CXGR	'B3C6'	64-bit binary-integer operand, extended HFP result

The fixed-point second operand is converted to the HFP format, and the normalized result is placed at the first-operand location.

A nonzero result is normalized. A zero result is made a positive true zero.

The second operand is a signed binary integer that is located in the general register designated by R_2 . A 32-bit operand is in bit positions 32-63 of the register.

The result is normalized and rounded toward zero (truncated) before it is placed at the first-operand location.

For CXFR and CXGR, the R_1 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 1, AFP register
- Specification (CXFR and CXGR)

CONVERT TO FIXED

Mnemonic R_1, M_3, R_2 [RRF-e]

Op Code	M_3	////	R_1	R_2
0	16	20	24	28 31

Mnemonic	Op Code	Operands
CFER	'B3B8'	Short HFP operand, 32-bit binary-integer result
CFDR	'B3B9'	Long HFP operand, 32-bit binary-integer result
CFXR	'B3BA'	Extended HFP operand, 32-bit binary-integer result
CGER	'B3C8'	Short HFP operand, 64-bit binary-integer result
CGDR	'B3C9'	Long HFP operand, 64-bit binary-integer result
CGXR	'B3CA'	Extended HFP operand, 64-bit binary-integer result

The HFP second operand is rounded to an integer value and then converted to the fixed-point format. The result is placed at the first-operand location.

The result is a signed binary integer that is placed in the general register designated by R_1 . A 32-bit result replaces bits 32-63 of the register, and bits 0-31 of the register remain unchanged.

The second operand is rounded to an integer value by rounding as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

- 0 Round toward 0
- 1 Round to nearest with ties away from 0
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$

For details on rounding see the section "Rounding Methods" on page 9-13.

A modifier other than 0, 1, or 4-7 is invalid.

The sign of the result is the sign of the second operand, except that a zero result has a plus sign.

If the rounded result would have a value exceeding the range that can be represented in the result format, the largest (in magnitude) representable number of the same sign as the source is placed at the target location, and condition code 3 is set.

HFP exponent underflow is not recognized because small values are rounded to one (with the appropriate sign) or to zero, depending on the rounding mode.

The M_3 field must designate a valid modifier; otherwise, a specification exception is recognized. For CFXR and CGXR, the R_2 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Source was zero
- 1 Source was less than zero
- 2 Source was greater than zero
- 3 Special case

Program Exceptions:

- Data with DXC 1, AFP register
- Specification

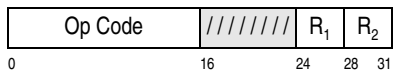
DIVIDE

Mnemonic1 R_1, R_2 [RR]

Op Code	R_1	R_2
0	8	12 15

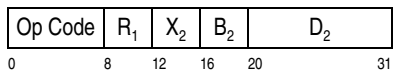
Mnemonic1	Op Code	Operands
DER	'3D'	Short HFP
DDR	'2D'	Long HFP

Mnemonic2 R₁,R₂ [RRE]



Mnemonic2 Op Code Operands
DXR 'B22D' Extended HFP

Mnemonic3 R₁,D₂(X₂,B₂) [RX-a]



Mnemonic3 Op Code Operands
DE '7D' Short HFP
DD '6D' Long HFP

The first operand (the dividend) is divided by the second operand (the divisor), and the normalized quotient is placed at the first-operand location. No remainder is preserved.

HFP division consists in characteristic subtraction and fraction division. The operands are first normalized to eliminate leading hexadecimal zeros. The difference between the dividend and divisor characteristics of the normalized operands, plus 64, is used as the characteristic of an intermediate quotient.

All dividend and divisor fraction digits participate in forming the fraction of the intermediate quotient. The intermediate-quotient fraction can have no leading hexadecimal zeros, but a right shift of one digit position may be necessary, with this causing an increase of the characteristic by one. The fraction is then truncated to the proper result-fraction length.

An HFP-exponent-overflow exception exists when the characteristic of the final quotient would exceed 127 and the fraction is not zero. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct. If, for extended results, the low-order characteristic would also exceed 127, it too is decreased by 128.

An HFP-exponent-underflow exception exists when the characteristic of the final quotient would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the

operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero. For extended results, HFP exponent underflow is not recognized when the low-order characteristic is less than zero but the high-order characteristic is equal to or greater than zero.

HFP exponent underflow does not occur when the characteristic of an operand becomes less than zero during normalization of the operands or when the intermediate-quotient characteristic is less than zero, as long as the final quotient can be represented with the correct characteristic.

When the divisor fraction is zero, an HFP-divide exception is recognized. This includes the case of division of zero by zero.

When the dividend fraction is zero but the divisor fraction is nonzero, the quotient is made a positive true zero. No HFP exponent overflow or HFP exponent underflow occurs.

The sign of the quotient is the exclusive or of the operand signs, except that the sign is always plus when the quotient is made a positive true zero.

For DXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

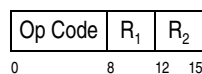
Program Exceptions:

- Access (fetch, operand 2 of DE and DD only)
- Data with DXC 1, AFP register
- HFP divide
- HFP exponent overflow
- HFP exponent underflow
- Specification (DXR only)

Programming Note: Examples of the use of the DIVIDE instruction (DER) are given in Appendix A.

HALVE

Mnemonic R_1, R_2 [RR]



Mnemonic	Op Code	Operands
HER	'34'	Short HFP
HDR	'24'	Long HFP

The second operand is divided by 2, and the normalized quotient is placed at the first-operand location.

The fraction of the second operand is shifted right one bit position, placing the contents of the rightmost bit position in the leftmost bit position of the guard digit, and a zero is supplied to the leftmost bit position of the fraction. The intermediate result, including the guard digit, is then normalized, and the final result is truncated to the proper length.

An HFP-exponent-underflow exception exists when the characteristic of the final result would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero.

When the fraction of the second operand is zero, the result is made a positive true zero, and no HFP exponent underflow occurs.

The sign of the result is the same as that of the second operand, except that the sign is always plus when the quotient is made a positive true zero.

Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 1, AFP register
- HFP exponent underflow

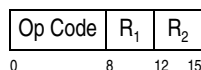
Programming Notes:

1. An example of the use of the HALVE instruction (HDR) is given in Appendix A.

2. With short and long operands, the halve operation is identical to a divide operation with the number 2 as divisor. Similarly, the result of HDR is identical to that of MD or MDR with one-half as a multiplier, and the result of HER is identical to that of MEE or MEER with one-half as a multiplier.

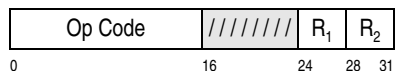
LOAD AND TEST

Mnemonic1 R_1, R_2 [RR]



Mnemonic1	Op Code	Operands
LTER	'32'	Short HFP
LTDR	'22'	Long HFP

Mnemonic2 R_1, R_2 [RRE]



Mnemonic2	Op Code	Operands
LTXR	'B362'	Extended HFP

The second operand is placed at the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code. The condition code is set the same as for a comparison of the second operand with zero.

For short and long operands, the second operand is placed unchanged in the first-operand location.

For extended operands, the high-order sign and the entire fraction of the source are placed unchanged in the result, and the low-order sign is set equal to the high-order sign. If the extended-operand fraction is nonzero, the high-order characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the extended-operand fraction is zero, the result is made a true zero with the same sign as the source (the high-order and low-order sign bits of the result are the same as the high-order sign bit of the source).

For LTXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

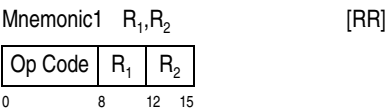
- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 --

Program Exceptions:

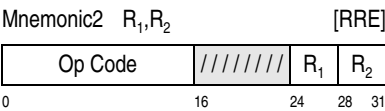
- Data with DXC 1, AFP register
- Specification (LTXR only)

Programming Note: When, for LTER and LTDR, the same register is designated as the first-operand and second-operand location, the operation is equivalent to a test without data movement.

LOAD COMPLEMENT



Mnemonic1	Op Code	Operands
LCER	'33'	Short HFP
LCDR	'23'	Long HFP



Mnemonic2	Op Code	Operands
LCXR	'B363'	Extended HFP

The second operand is placed at the first-operand location with the sign bit inverted.

The sign bit is inverted even if the operand is zero. For all operand lengths, the source fraction is placed unchanged in the result.

For short and long operands, the source characteristic is placed unchanged in the result.

For extended operands, the low-order sign is set equal to the high-order sign. If the extended-operand fraction is nonzero, the high-order characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the extended-operand fraction is zero, the result is made a true zero with the sign inverted from the source (the high-order and low-order sign bits of the result are inverted from the high-order sign bit of the source).

For LCXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

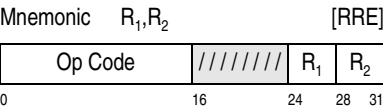
Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 --

Program Exceptions:

- Data with DXC 1, AFP register
- Specification (LCXR only)

LOAD FP INTEGER



Mnemonic	Op Code	Operands
FIER	'B377'	Short HFP
FIDR	'B37F'	Long HFP
FIXR	'B367'	Extended HFP

The second operand is truncated (rounded toward zero) to an integer value in the same floating-point format, and the normalized result is placed at the first-operand location.

A nonzero result is normalized. A zero result is made a positive true zero.

For FIXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Data with DXC 1, AFP register
- Specification (FIXR only)

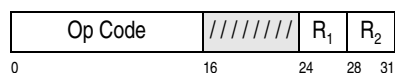
Programming Notes:

1. LOAD FP INTEGER truncates (rounds toward zero) an HFP number to an integer value. These integers, which remain in the HFP format, should not be confused with binary integers, which use a fixed-point format.

2. If the HFP operand has a large enough exponent so that it is already an integer, the result value remains the same, except that an unnormalized operand is normalized, and an operand with a zero fraction is changed to a positive true zero.

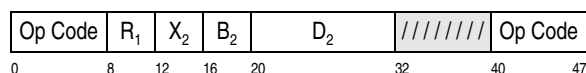
LOAD LENGTHENED

Mnemonic1 R₁,R₂ [RRE]



Mnemonic1	Op Code	Operands
LDER	'B324'	Short HFP operand 2, long HFP operand 1
LXDR	'B325'	Long HFP operand 2, extended HFP operand 1
LXER	'B326'	Short HFP operand 2, extended HFP operand 1

Mnemonic2 R₁,D₂(X₂,B₂) [RXE]



Mnemonic2	Op Code	Operands
LDE	'ED24'	Short HFP operand 2, long HFP operand 1
LXD	'ED25'	Long HFP operand 2, extended HFP operand 1
LXE	'ED26'	Short HFP operand 2, extended HFP operand 1

The second operand is extended to a longer format, and the result is placed at the first-operand location.

For all operand lengths, the source fraction is extended with zeros and placed in the result. The sign bit of the result is set the same as the sign of the source even when the result is made a true zero.

For long results, the source characteristic is placed unchanged in the result.

For extended results, the low-order sign is set equal to the high-order sign. If the fraction is nonzero, the source characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the fraction is zero, the result is made a true zero with the same sign as the source (the high-order and low-order sign bits of the result are the same as the sign bit of the source).

For LXD, LXDR, LXE, and LXER, the R₁ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

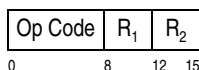
Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of LDE, LXE, and LXD only)
- Data with DXC 1, AFP register
- Specification (LXE, LXER, LXD, LXDR)

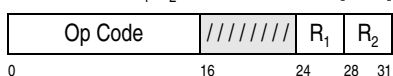
LOAD NEGATIVE

Mnemonic1 R₁,R₂ [RR]



Mnemonic1	Op Code	Operands
LNDR	'31'	Short HFP
LXDR	'21'	Long HFP

Mnemonic2 R₁,R₂ [RRE]



Mnemonic2	Op Code	Operands
LNXR	'B361'	Extended HFP

The second operand is placed at the first-operand location with the sign bit made one.

The sign bit is made one even if the operand is zero. For all operand lengths, the source fraction is placed unchanged in the result.

For short and long operands, the source characteristic is placed unchanged in the result.

For extended operands, the low-order sign is set equal to the high-order sign. If the extended-operand fraction is nonzero, the high-order characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the extended-operand fraction is zero, the result is made a negative true zero (the high-order and low-order sign bits of the result are set to one).

For LNXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

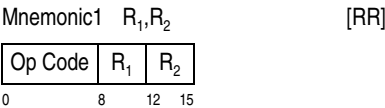
- 0 Result is zero

- 1 Result is less than zero
- 2 --
- 3 --

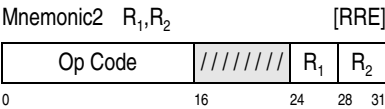
Program Exceptions:

- Data with DXC 1, AFP register
- Specification (LNXR only)

LOAD POSITIVE



Mnemonic1	Op Code	Operands
LPER	'30'	Short HFP
LPDR	'20'	Long HFP



Mnemonic2	Op Code	Operands
LPXR	'B360'	Extended HFP

The second operand is placed at the first-operand location with the sign bit made zero.

For all operand lengths, the sign bit is made zero, and the source fraction is placed unchanged in the result.

For short and long operands, the source characteristic is placed unchanged in the result.

For extended operands, the low-order sign is set equal to the high-order sign. If the extended-operand fraction is nonzero, the high-order characteristic is placed unchanged in the result high-order characteristic, and the low-order characteristic is set to 14 less than the high-order characteristic, modulo 128. If the extended-operand fraction is zero, the result is made a positive true zero (the high-order and low-order sign bits of the result are set to zero).

For LPXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

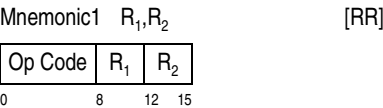
- 0 Result is zero

- 1 --
- 2 Result is greater than zero
- 3 --

Program Exceptions:

- Data with DXC 1, AFP register
- Specification (LPXR only)

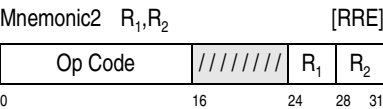
LOAD ROUNDED



Mnemonic1	Op Code	Operands
LEDR	'35'	Long HFP operand 2, short HFP operand 1
LDXR	'25'	Extended HFP operand 2, long HFP operand 1

The above mnemonics are alternatives to the following older mnemonics that are less descriptive of operand lengths:

LRER	'35'	Long HFP operand 2, short HFP operand 1
LRDR	'25'	Extended HFP operand 2, long HFP operand 1



Mnemonic2	Op Code	Operands
LEXR	'B366'	Extended HFP operand 2, short HFP operand 1

The second operand is rounded to a shorter format, and the result is placed at the first-operand location.

Rounding consists in adding a one to the leftmost bit position of the second operand that is to be dropped and propagating any carry through the fraction. The sign of the second operand is ignored, and addition is performed as if the fraction were positive.

If rounding causes a carry out of the leftmost hexadecimal digit position of the fraction, the fraction is shifted right one digit position so that the carry becomes the leftmost digit of the fraction, and the characteristic is increased by one.

The intermediate fraction is then truncated to the proper result-fraction length. For LEDR and LEXR, the result replaces the leftmost 32 bits of the target register, and the rightmost 32 bit positions of the target register remain unchanged. For LDXR, the 64-bit

result is placed in a floating-point register, not a floating-point register pair.

The sign of the result is the same as the sign of the second operand. There is no normalization to eliminate leading zeros.

An HFP-exponent-overflow exception exists when shifting the fraction right would cause the characteristic to exceed 127. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct.

HFP-exponent-underflow and HFP-significance exceptions cannot occur.

For LDXR (or LRDR) and LEXR, the R_2 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

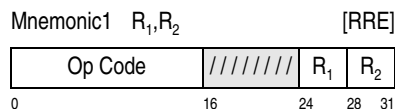
Condition Code: The code remains unchanged.

Program Exceptions:

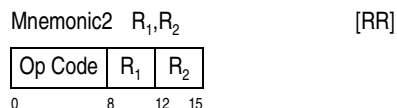
- Data with DXC 1, AFP register
- HFP exponent overflow
- Specification (LDXR, LEXR, LRDR)

Programming Note: The sign of the rounded result is the same as the sign of the operand, even when the result is zero.

MULTIPLY



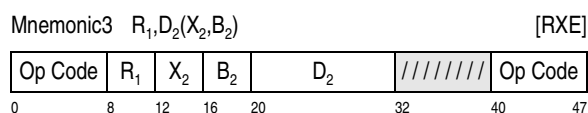
Mnemonic1	Op Code	Operands
MEER	'B337'	Short HFP



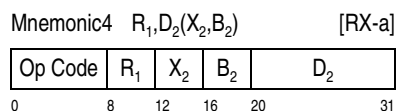
Mnemonic2	Op Code	Operands
MDR	'2C'	Long HFP
MXR	'26'	Extended HFP
MDER	'3C'	Short HFP multiplier and multiplicand, long HFP product
MXDR	'27'	Long HFP multiplier and multiplicand, extended HFP product

The above mnemonic MDER is an alternative to the following older mnemonic that is less descriptive of operand lengths:

MER	'3C'	Short HFP multiplier and multiplicand, long HFP product
-----	------	---



Mnemonic3	Op Code	Operands
MEE	'ED37'	Short HFP



Mnemonic4	Op Code	Operands
MD	'6C'	Long HFP
MDE	'7C'	Short HFP multiplier and multiplicand, long HFP product
MXD	'67'	Long HFP multiplier and multiplicand, extended HFP product

The above mnemonic MDE is an alternative to the following older mnemonic that is less descriptive of operand lengths:

ME	'7C'	Short HFP multiplier and multiplicand, long HFP product
----	------	---

The normalized product of the second operand (the multiplier) and the first operand (the multiplicand) is placed at the first-operand location.

Multiplication of two HFP numbers consists in exponent addition and fraction multiplication. The operands are first normalized to eliminate leading hexadecimal zeros. The sum of the characteristics of the normalized operands, less 64, is used as the characteristic of the intermediate product.

The fraction of the intermediate product is the exact product of the normalized operand fractions. If the intermediate-product fraction has one leading hexadecimal zero digit, the fraction is shifted left one digit position, bringing the contents of the guard-digit position into the rightmost position of the result fraction,

and the intermediate-product characteristic is reduced by one. The fraction is then truncated to the proper result-fraction length.

For MDE and MDER, the multiplier and multiplicand fractions have six hexadecimal digits; the product fraction has the full 14 digits of the long format, with the two rightmost fraction digits always zeros. For MEE and MEER, the multiplier and multiplicand fractions have six digits, and the final product fraction is truncated to six digits; the result, as for all short-format results, replaces the leftmost 32 bits of the target register, and the rightmost 32 bit positions of the target register remain unchanged.

For MD and MDR, the multiplier and multiplicand fractions have 14 digits, and the final product fraction is truncated to 14 digits. For MXD and MXDR, the multiplier and multiplicand fractions have 14 digits, with the multiplicand occupying the high-order part of the first operand; the final product fraction contains 28 digits and is an exact product of the operand fractions. For MXR, the multiplier and multiplicand fractions have 28 digits, and the final product fraction is truncated to 28 digits.

An HFP-exponent-overflow exception exists when the characteristic of the final product would exceed 127 and the fraction is not zero. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct. If, for extended results, the low-order characteristic would also exceed 127, it too is decreased by 128.

HFP exponent overflow is not recognized when the intermediate-product characteristic is initially 128 but is brought back within range by normalization.

An HFP-exponent-underflow exception exists when the characteristic of the final product would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero.

For extended results, HFP exponent underflow is not recognized when the low-order characteristic is less than zero but the high-order characteristic is equal to or greater than zero.

HFP exponent underflow does not occur when the characteristic of an operand becomes less than zero during normalization of the operands, as long as the final product can be represented with the correct characteristic.

If either or both operand fractions are zero, the result is made a positive true zero, and no HFP exponent overflow or HFP exponent underflow occurs.

The sign of the product is the exclusive or of the operand signs, except that the sign is always plus when the result is made a true zero.

The R_1 field for MXD, MXDR, and MXR, and the R_2 field for MXR must designate valid floating-point-register pairs. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

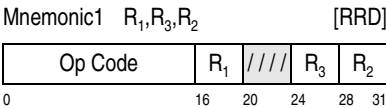
Program Exceptions:

- Access (fetch, operand 2 of MDE, MEE, MD, and MXD only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP exponent underflow
- Specification (MXD, MXDR, MXR)

Programming Notes:

1. An example of the use of the MULTIPLY instruction (MDR) is given in Appendix A.
2. Interchanging the two operands in an HFP multiplication does not affect the value of the product.

MULTIPLY AND ADD



Mnemonic1	Op Code	Operands
MAER	'B32E'	Short HFP
MADR	'B33E'	Long HFP

Mnemonic2 $R_1, R_3, D_2(X_2, B_2)$ [RXF]

Op Code	R ₃	X ₂	B ₂	D ₂		R ₁	////	Op Code
0	8	12	16	20	32	36	40	47

Mnemonic2 Op Code Operands

MAE 'ED2E' Short HFP

MAD 'ED3E' Long HFP

MULTIPLY AND SUBTRACT

Mnemonic1 R_1, R_3, R_2 [RRD]

Op Code		R ₁	////	R ₃	R ₂
0		16	20	24	28 31

Mnemonic1 Op Code Operands

MSER 'B32F' Short HFP

MSDR 'B33F' Long HFP

Mnemonic2 $R_1, R_3, D_2(X_2, B_2)$ [RXF]

Op Code	R ₃	X ₂	B ₂	D ₂		R ₁	////	Op Code
0	8	12	16	20	32	36	40	47

Mnemonic2 Op Code Operands

MSE 'ED2F' Short HFP

MSD 'ED3F' Long HFP

The third operand is multiplied by the second operand, and then the first operand is added to or subtracted from the product. The sum or difference is placed at the first-operand location. The MULTIPLY AND ADD and MULTIPLY AND SUBTRACT operations may be summarized as:

$$op_1 = op_3 \cdot op_2 \pm op_1$$

The third and second HFP operands are multiplied, forming an intermediate product, and the first operand is then added (or subtracted) algebraically to (or from) the intermediate product, forming an intermediate result. The exponent and fraction of the intermediate product and intermediate result are maintained exactly. The intermediate result, if nonzero, is normalized and truncated to the operand format and then placed at the first-operand location.

The sign of the result is determined by the rules of algebra, unless the intermediate-result fraction is zero, in which case the result is made a positive true zero.

An HFP-exponent-overflow exception exists when the characteristic of the normalized result would exceed 127 and the fraction is not zero. The opera-

tion is completed by making the result characteristic 128 less than the correct value, and a program interruption for HFP exponent overflow occurs. The result is normalized, and the sign and fraction remain correct.

HFP exponent overflow is not recognized on intermediate values, provided the normalized result can be represented with the correct characteristic.

An HFP-exponent-underflow exception exists when the characteristic of the normalized result would be less than zero and the fraction is not zero. If the HFP-exponent-underflow mask bit in the PSW is one, the operation is completed by making the result characteristic 128 greater than the correct value, and a program interruption for HFP exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the HFP-exponent-underflow mask bit in the PSW is zero, a program interruption does not occur; instead, the operation is completed by making the result a positive true zero.

HFP exponent underflow is not recognized on input operands and intermediate values, provided the normalized result can be represented with the correct characteristic.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of MAE, MAD, MSE, MSD)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP exponent underflow
- Operation (if the HFP multiply-add/subtract facility is not installed)

Programming Note: MULTIPLY AND ADD (SUBTRACT) differs from MULTIPLY followed by ADD (SUBTRACT) NORMALIZED in the following ways:

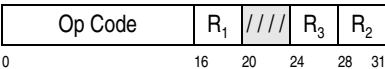
1. The product is maintained to full precision, and overflow and underflow are not recognized on the product.
2. The HFP-significance exception is not recognized for MULTIPLY AND ADD (SUBTRACT).
3. ADD (SUBTRACT) NORMALIZED maintains only a single guard digit and does not prenormalize input operands; thus, in some cases, an

unnormalized input operand may cause loss of precision in the result. MULTIPLY AND ADD (SUBTRACT) maintains the entire intermediate sum (difference), which is normalized before the truncation operation is performed; thus, unnorm-alized operands do not cause any additional loss of precision.

- 4. On most models, the execution time of MULTI-PLY AND ADD (SUBTRACT) is less than the combined execution time of MULTIPLY followed by ADD (SUBTRACT) NORMALIZED. The per-formance of MULTIPLY AND ADD (SUBTRACT) may be severely degraded in the case of unnor-malized input operands.

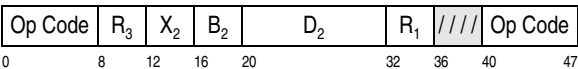
MULTIPLY AND ADD UNNORMALIZED

Mnemonic1 R₁,R₃,R₂ [RRD]



Mnemonic1	Op Code	Operands
MAYR	'B33A'	Long HFP sources, extended HFP result
MAYHR	'B33C'	Long HFP sources, high-order part of extended HFP result
MAYLR	'B338'	Long HFP sources, low-order part of extended HFP result

Mnemonic2 R₁,R₃,D₂(X₂,B₂) [RXF]



Mnemonic2	Op Code	Operands
MAY	'ED3A'	Long HFP sources, extended HFP result
MAYH	'ED3C'	Long HFP sources, high-order part of extended HFP result
MAYL	'ED38'	Long HFP sources, low-order part of extended HFP result

The second and third HFP operands are multiplied, forming an intermediate product; the first operand (addend) is then added algebraically to the interme-diate product to form an intermediate sum; the inter-mediate-sum fraction is truncated on the left or on the right, if need be, to form an intermediate extended result. All (or a part) of the intermediate extended result is placed in the floating-point-register pair (or floating-point register) designated by the R₁ field. The operands, intermediate values, and results are not normalized to eliminate leading hexadecimal zeros.

The MULTIPLY AND ADD UNNORMALIZED opera-tions may be summarized as:

t₁ ← op₃•op₂+op₁

Multiplication of two HFP numbers consists in expo-nent addition and fraction multiplication. The sum of the characteristics of the second and third operands, less 64, is used as the characteristic of the high-order part of the intermediate product; this value is independent of whether the result fraction is zero. The characteristic of the intermediate product is maintained correctly and does not wrap.

In all cases, the second- and third-operand fractions have 14 digits; the intermediate-product fraction con-tains 28 digits and is an exact product of the operand fractions. The intermediate-product fraction is not inspected for leading hexadecimal zero digits and is used without shifting in the subsequent addition.

In all cases, the first operand is located in the float-ing-point register designated by the R₁ field and the first-operand fraction has 14 digits.

Addition of two HFP numbers consists in characteris-tic comparison, fraction alignment, and signed frac-tion addition. The characteristics of the intermediate product and the addend are compared. If the charac-teristics are equal, no alignment is required. If the characteristic of the addend is smaller than the char-acteristic of the product, the fraction of the addend is aligned with the product fraction by a right shift, with its characteristic increased by one for each hexadeci-mal digit of shift. If the characteristic of the addend is larger than the characteristic of the product, the frac-tion of the addend is aligned with the product fraction by a left shift, with its characteristic decreased by one for each hexadecimal digit of shift. Shifting continues until the two characteristics agree. All hexadecimal digits shifted out are preserved and participate in the subsequent addition.

After alignment, the fractions with signs are then added algebraically to form a signed intermediate sum. The fraction of the intermediate sum is main-tained exactly. The intermediate-sum fraction is not inspected for leading hexadecimal zero digits and is not shifted. Only those 28 hexadecimal digits of the intermediate-sum fraction which are aligned with the 28 hexadecimal digits of the intermediate-product fraction are used as the fraction of the intermediate extended-result.

The high-order characteristic of the intermediate extended result is set to the characteristic of the intermediate product, modulo 128. The low-order characteristic of the intermediate extended result is set to 14 less than the high-order characteristic, modulo 128. Wrap-around of the characteristic is independent of whether the result fraction is zero.

The sign of the result is determined by the rules of algebra, unless the entire intermediate-sum fraction is zero, in which case the sign of the result is made positive.

For MAY and MAYR, the entire intermediate extended result is placed in the floating-point register-pair designated by the R_1 field; the R_1 field may designate either the lower-numbered or higher-numbered register of a floating-point-register pair. For MAYH and MAYHR, the high-order part of the intermediate extended result is placed in the floating-point register designated by the R_1 field and the low-order part is discarded. For MAYL and MAYLR, the low-order part of the intermediate extended result is placed in the floating-point register designated by the R_1 field and the high-order part is discarded.

HFP-exponent-overflow and HFP-exponent-underflow exceptions are not recognized. Characteristics of the intermediate extended result wrap-around modulo 128 and no exception is reported.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of MAYH, MAY, and MAYL only)
- Data with DXC 1, AFP register
- Operation (if the HFP-unnormalized-extensions facility is not installed)

Programming Notes:

1. While performance of MULTIPLY AND ADD may be severely degraded for the case of unnormalized input operands, this is not the case for MULTIPLY AND ADD UNNORMALIZED.
2. MULTIPLY AND ADD UNNORMALIZED can be used to efficiently perform multiple precision arithmetic on numbers of any arbitrary size. This is accomplished by organizing the numbers into big digits of 52 bits each, with each big digit

maintained as an integer in the HFP long format. Using a radix of 2^{52} and big digits which can hold up to 56 bits provides a redundant representation. This redundant representation permits multiplication and addition using a “carry save” technique and permits maximum utilization of the floating-point pipeline.

3. By setting the multiplier to an integer value of 1 with the proper characteristic, the multiplicand can be scaled by any power of 16 and then added to the addend. This permits, for example, adding the “carry” from one stage of a multiplication to the “sum” of the next stage to the left. In the same manner, the “sum” of one stage can be scaled to be added to the “carry” of the stage to the right.
4. In the first round of a multiply and accumulate, the step of clearing the accumulated value to zero, may be avoided by using the MULTIPLY UNNORMALIZED instead of MULTIPLY AND ADD UNNORMALIZED.
5. For additional notes concerning the unnormalized operations see the programming notes under the MULTIPLY UNNORMALIZED instruction.

MULTIPLY UNNORMALIZED

Mnemonic1 R_1, R_3, R_2 [RRD]

Op Code	R_1	////	R_3	R_2
0	16	20	24	28 31

Mnemonic1	Op Code	Operands
MYR	'B33B'	Long HFP multiplier and multiplicand, extended HFP product
MYHR	'B33D'	Long HFP multiplier and multiplicand, high-order part of extended HFP product
MYLR	'B339'	Long HFP multiplier and multiplicand, low-order part of extended HFP product

Mnemonic2 $R_1, R_3, D_2(X_2, B_2)$ [RXF]

Op Code	R_3	X_2	B_2	D_2	R_1	////	Op Code
0	8	12	16	20	32	36	40 47

Mnemonic2	Op Code	Operands
MY	'ED3B'	Long HFP multiplier and multiplicand, extended HFP product
MYH	'ED3D'	Long HFP multiplier and multiplicand, high-order part of extended HFP product
MYL	'ED39'	Long HFP multiplier and multiplicand, low-order part of extended HFP product

The second and third HFP operands are multiplied, forming an intermediate product, which, in turn, is used to form an intermediate extended result. All (or a part) of the intermediate extended result is placed in the floating-point-register pair (or floating-point register) designated by the R_1 field. The operands, intermediate values, and results are not normalized to eliminate leading hexadecimal zeros.

Multiplication of two HFP numbers consists in exponent addition and fraction multiplication. The sum of the characteristics of the second and third operands, less 64, is used as the characteristic of the high-order part of the intermediate product; this value is independent of whether the result fraction is zero. The characteristic of the intermediate product is maintained correctly and does not wrap.

The high-order characteristic of the intermediate extended result is set to the characteristic of the intermediate product, modulo 128. The low-order characteristic of the intermediate extended result is set to 14 less than the high-order characteristic, modulo 128. Wrap-around of the characteristic is independent of whether the result fraction is zero.

In all cases, the second- and third-operand fractions have 14 digits; the intermediate-product fraction contains 28 digits and is an exact product of the operand fractions. The intermediate-product fraction is not inspected for leading hexadecimal zero digits and is used without shifting as the fraction of the intermediate extended result.

The sign of the result is the exclusive or of the operand signs, including the case when the result fraction is zero.

For MY and MYR, the entire intermediate extended result is placed in the floating-point-register pair designated by the R_1 field. For MYH and MYHR, the high-order part of the intermediate extended result is placed in the floating-point register designated by the R_1 field and the low-order part is discarded. For MYL and MYLR, the low-order part of the intermediate extended result is placed in the floating-point register designated by the R_1 field and the high-order part is discarded.

HFP-exponent-overflow and HFP-exponent-underflow exceptions are not recognized. Characteristics of the intermediate extended result wrap-around modulo 128 and no exception is reported.

The R_1 field for MY and MYR must designate a valid floating-point-register pair. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of MYH, MY, and MYL only)
- Data with DXC 1, AFP register
- Operation (if the HFP-unnormalized-extensions facility is not installed)
- Specification (MY and MYR)

Programming Notes:

1. The formal definition of Long HFP numbers is in terms of a 14-digit fraction and a signed exponent in “excess-64 notation”. But, without change in function, these numbers can also be described as having a 14-digit integer and a signed exponent in “excess-78 notation”. When using the unnormalized instructions, the programmer may find it more convenient to consider these values in the later terms rather than the former. Thus, in these terms, MULTIPLY UNNORMALIZED multiplies two 14-digit integers to produce a 28-digit integer result. The characteristic of the low-order part is the sum of the operand characteristics less 78 and the high-order characteristic is set 14 larger than the low-order part. When considered as described above, MULTIPLY UNNORMALIZED provides the exact 28-digit integer product of two 14-digit integers.
2. Rather than using MY (or MYR) to provide the entire 28-digits in a floating-point-register pair, the programmer may find it more convenient to issue two separate instructions, MYH and MYL (or MYHR and MYLR), to place the high-order part and low-order part in any floating-point register. It is expected that on most machines, the performance of these two approaches is essentially the same.
3. The MULTIPLY UNNORMALIZED instruction format designates three operand locations and can be used to produce a product without overlaying the source operands. In many cases this saves an extra load.

SQUARE ROOT

Mnemonic1 R₁,R₂ [RRE]

Op Code	////////	R ₁	R ₂
0	16	24	28 31

Mnemonic1	Op Code	Operands
SQER	'B245'	Short HFP
SQDR	'B244'	Long HFP
SQXR	'B336'	Extended HFP

Mnemonic2 R₁,D₂(X₂,B₂) [RXE]

Op Code	R ₁	X ₂	B ₂	D ₂	////////	Op Code
0	8	12	16	20	32	40 47

Mnemonic2	Op Code	Operands
SQE	'ED34'	Short HFP
SQD	'ED35'	Long HFP

The normalized and rounded square root of the second operand is placed at the first-operand location.

When the fraction of the second operand is zero, the sign and characteristic of the second operand are ignored, and the operation is completed by placing a positive true zero at the first-operand location.

If the second operand is less than zero, an HFP-square-root exception is recognized.

If the second operand is normalized and greater than zero, the characteristic, fraction, and sign of the result are produced as follows:

- The result characteristic is one-half of the sum of the operand characteristic and either 64, if the operand characteristic is even, or 65, if it is odd.
- If the operand characteristic is odd, the operand fraction is shifted right one digit position, the rightmost digit entering the guard-digit position.
- An intermediate-result fraction is produced by computing without rounding the square root of the operand fraction, after any right shift as described. The intermediate-result fraction consists of the 29 most significant hexadecimal digits of the square-root result in the extended format, 15 in the long format, or seven in the short format, where all three formats include a guard digit on the right.
- A one is added to the leftmost bit of the guard digit of the intermediate result, any carry is prop-

agated to the left, and the guard digit is dropped to produce the result fraction.

- The result sign is made plus.

If the second operand is unnormalized and greater than zero, the operand is first normalized. The operation then proceeds as for normalized operands.

For SQXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2 of SQE and SQD only)
- Data with DXC 1, AFP register
- HFP square root
- Specification (SQXR only)

Programming Notes:

1. The use of the SQUARE ROOT instruction with short operands (SQER) is illustrated by the examples in the following table:

Operand (hex)	Decimal Value	Result (hex)	Decimal Value
42 190000	25.0	41 500000	5.0
40 400000	0.250	40 800000	0.50
40 800000	0.50	40 B504F3	0.7071...
41 800000	8.0	41 2D413D	2.8284...

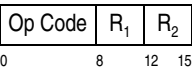
2. The result fraction is correctly normalized without any further left or right shifts of the intermediate-result fraction and without any further exponent adjustment. Rounding cannot cause a carry out of the leftmost digit.
3. Although a characteristic greater than 127 or less than zero may temporarily be generated during the operation, the result characteristic is always within the representable range, and no HFP exponent overflow or underflow occurs.

Specifically, the smallest nonzero operand in the long format consists of a one bit, preceded on the left by 63 zeros. This operand is an unnormalized number with a value of 16^{-78} , and its square root is 16^{-39} . The normalized representation of this result has a characteristic of 26 (decimal). Similarly, the square root of the largest representable operand has a characteristic of 96

(decimal). The instruction, therefore, cannot produce a nonzero result with a characteristic outside the range of 26 to 96.

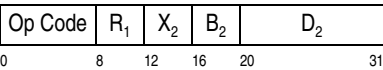
SUBTRACT NORMALIZED

Mnemonic1 R₁,R₂ [RR]



Mnemonic1	Op Code	Operands
SER	'3B'	Short HFP
SDR	'2B'	Long HFP
SXR	'37'	Extended HFP

Mnemonic2 R₁,D₂(X₂,B₂) [RX-a]



Mnemonic2	Op Code	Operands
SE	'7B'	Short HFP
SD	'6B'	Long HFP

The second operand is subtracted from the first operand, and the normalized difference is placed at the first-operand location.

The execution of SUBTRACT NORMALIZED is identical to that of ADD NORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

For SXR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 --

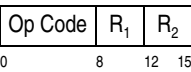
Program Exceptions:

- Access (fetch, operand 2 of SE and SD only)

- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP exponent underflow
- HFP significance
- Specification (SXR only)

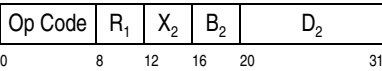
SUBTRACT UNNORMALIZED

Mnemonic1 R₁,R₂ [RR]



Mnemonic1	Op Code	Operands
SUR	'3F'	Short HFP
SWR	'2F'	Long HFP

Mnemonic2 R₁,D₂(X₂,B₂) [RX-a]



Mnemonic2	Op Code	Operands
SU	'7F'	Short HFP
SW	'6F'	Long HFP

The second operand is subtracted from the first operand, and the unnormalized difference is placed at the first-operand location.

The execution of SUBTRACT UNNORMALIZED is identical to that of ADD UNNORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

Resulting Condition Code:

- 0 Result fraction zero
- 1 Result less than zero
- 2 Result greater than zero
- 3 --

Program Exceptions:

- Access (fetch, operand 2 of SU and SW only)
- Data with DXC 1, AFP register
- HFP exponent overflow
- HFP significance

Chapter 19. Binary-Floating-Point Instructions

Binary-Floating-Point Facility	19-1	Instructions	19-8
Floating-Point-Control (FPC) Register	19-2	ADD	19-11
BFP Arithmetic	19-2	COMPARE	19-16
BFP Data Formats	19-2	COMPARE AND SIGNAL	19-17
BFP Short Format	19-2	CONVERT FROM FIXED	19-18
BFP Long Format	19-2	CONVERT FROM LOGICAL	19-20
BFP Extended Format	19-2	CONVERT TO FIXED	19-21
Biased Exponent	19-2	CONVERT TO LOGICAL	19-23
Significand	19-3	DIVIDE	19-25
Values of Nonzero Numbers	19-3	DIVIDE TO INTEGER	19-27
Classes of BFP Data	19-4	LOAD AND TEST	19-30
Zeros	19-4	LOAD COMPLEMENT	19-30
Subnormal Numbers	19-4	LOAD FP INTEGER	19-31
Normal Numbers	19-4	LOAD LENGTHENED	19-32
Infinities	19-4	LOAD NEGATIVE	19-33
Signaling and Quiet NaNs	19-4	LOAD POSITIVE	19-33
BFP-Format Conversion	19-5	LOAD ROUNDED	19-34
BFP Rounding	19-5	MULTIPLY	19-35
BFP Comparison	19-5	MULTIPLY AND ADD	19-37
Remainder	19-5	MULTIPLY AND SUBTRACT	19-37
IEEE Exceptions	19-7	SQUARE ROOT	19-38
Result Figures	19-7	SUBTRACT	19-39
Data-Exception Codes (DXC) and		TEST DATA CLASS	19-39
Abbreviations	19-7		

Binary-Floating-Point Facility

The binary-floating-point (BFP) facility provides instructions to operate on binary (radix-2) floating-point data.

BFP provides a number of important advantages over hexadecimal floating point (HFP):

- Greater precision and exponent range (except for numbers in the short format where HFP has the greater range).
- Automatic rounding to the nearest value for all arithmetic operations. There are directed-rounding options that may be used instead.
- Special entities of “infinity” and “Not-a-Number” (NaN), which are accepted and handled by arithmetic operations in a reasonable fashion. They provide better defaults for exponent overflow and invalid operations (such as division of zero by zero). This allows most programs to continue

running without hiding such errors and without using specialized exception handlers.

- Exponent underflow gives “subnormal” numbers as the default, which provides more consistent results than the abrupt result of zero produced by the HFP instructions.
- The greater exponent range makes exponent overflow and underflow in correctly written programs very unlikely, so that programmers may often be able to ignore these conditions.
- Both mask and flag bits are provided for all IEEE exceptions. The mask bits enable or disable interruptions. When interruptions are disabled, the flag bits keep track of exceptions during execution so that warning messages may be issued.
- Programs can be migrated from and to workstations and other systems using different architectures and still give consistent results, provided that floating-point operations on the other systems also conform to ANSI/IEEE Standard 754-1985. This does not mean, however, that bit-wise

compatible results can be guaranteed, because the standard allows implementation flexibility, especially in the presence of exceptions.

Programming Note: The bit representation of the BFP data formats in storage is defined to be left-to-right in a manner that is uniform for all numeric operands in the z/Architecture architecture. Although the format diagrams in ANSI/IEEE Standard 754-1985 appear to use the same left-to-right bit sequence, the standard only defines the meaning of the bits without specifying how they appear in storage; the storage arrangement is left to the implementation. Several implementations in fact use other sequences; this may affect programs which are dependent on the bit representation of floating-point data in storage.

Floating-Point-Control (FPC) Register

The floating-point-control (FPC) register is a 32-bit register that contains mask bits, flag bits, a data-exception code, and two rounding-mode fields. The FPC register is described in the section “Floating-Point-Control (FPC) Register” on page 9-9. An overview of the FPC register is shown in Figure 9-5 on page 9-9. Details are shown in Figure 9-6 on page 9-9, Figure 9-7 on page 9-10, and in Figure 9-8 on page 9-10. (In Figure 9-6 on page 9-9, the abbreviations “IM” and “SF” are based on the terms “interruption mask” and “status flag,” respectively.)

The bits of the FPC register are often referred to as, for example, FPC 1.0, meaning bit 0 of byte 1 of the register.

BFP Arithmetic

BFP Data Formats

Binary-floating-point data may be represented in any of three formats: short, long, or extended.

BFP Short Format

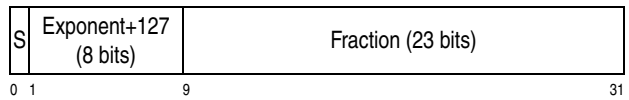


Figure 19-1. BFP Short Format (4 bytes)

When an operand in the BFP short format is loaded into a floating-point register, it occupies the left half of the register, and the right half remains unchanged.

BFP Long Format

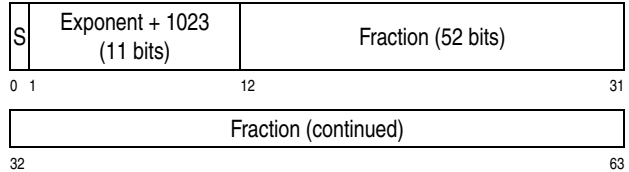


Figure 19-2. BFP Long Format (8 bytes)

When an operand in the BFP long format is loaded into a floating-point register, it occupies the entire register.

BFP Extended Format

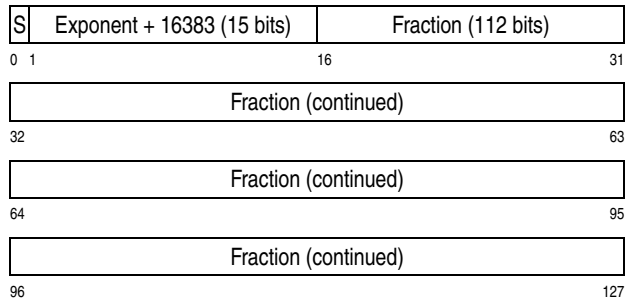


Figure 19-3. BFP Extended Format (16 bytes)

An operand in the BFP extended format occupies a register pair. The sign and biased exponent are in the leftmost 16 bits of the lower-numbered register of the pair and are followed by the leftmost 48 bits of the fraction. The rightmost 64 bits of the fraction are in the higher-numbered register of the pair.

The properties of the three formats are tabulated in Figure 19-4 on page 19-3.

Biased Exponent

For each format, the bias that is used to allow all exponents to be expressed as unsigned numbers is shown in the Figure 19-4 on page 19-3. Biased exponents are similar to the characteristics of the HFP format, except that special meanings are attached to

Property	Format		
	Short	Long	Extended
Format length (bits)	32	64	128
Biased-exponent length (bits)	8	11	15
Fraction length (bits)	23	52	112
Precision (p)	24	53	113
Maximum left-units-view (LUV) exponent (Emax)	127	1023	16383
Minimum left-units-view (LUV) exponent (Emin)	-126	-1022	-16382
Left-units-view (LUV) bias	127	1023	16383
Nmax	$(1-2^{-24}) \times 2^{128}$ $\approx 3.4 \times 10^{38}$	$(1-2^{-53}) \times 2^{1024}$ $\approx 1.8 \times 10^{308}$	$(1-2^{-113}) \times 2^{16384}$ $\approx 1.2 \times 10^{4932}$
Nmin	1.0×2^{-126} $\approx 1.2 \times 10^{-38}$	1.0×2^{-1022} $\approx 2.2 \times 10^{-308}$	1.0×2^{-16382} $\approx 3.4 \times 10^{-4932}$
Dmin	1.0×2^{-149} $\approx 1.4 \times 10^{-45}$	1.0×2^{-1074} $\approx 4.9 \times 10^{-324}$	1.0×2^{-16494} $\approx 6.5 \times 10^{-4966}$
Explanation: \approx Value is approximate. Dmin Smallest (in magnitude) representable subnormal number. Nmax Largest (in magnitude) representable finite number. Nmin Smallest (in magnitude) representable normal number.			

Figure 19-4. Summary of BFP Data Formats

biased exponents of all zeros and all ones, which are discussed in the section “Classes of BFP Data” on page 19-4.

Significand

In each format, the binary point of a BFP number is considered to be to the left of the leftmost fraction bit. To the left of the binary point there is an implied unit bit, which is considered to be one for normal numbers and zero for zeros and subnormal numbers. The fraction with the implied unit bit appended on the left is the *significand* of the number.

The value of a normal BFP number is the significand multiplied by the radix 2 raised to the power of the unbiased exponent. The value of a subnormal BFP number is the significand multiplied by the radix 2 raised to the power of the minimum exponent.

A value of one in the rightmost bit position of the significand in each format is sometimes referred to as one ulp (unit in the last place).

Values of Nonzero Numbers

The values of nonzero numbers in the various formats are shown in Figure 19-5.

Number Class	Format	Value
Normal	Short	$\pm 2^{e-127} \times (1.f)$
	Long	$\pm 2^{e-1023} \times (1.f)$
	Extended	$\pm 2^{e-16383} \times (1.f)$
Subnormal	Short	$\pm 2^{-126} \times (0.f)$
	Long	$\pm 2^{-1022} \times (0.f)$
	Extended	$\pm 2^{-16382} \times (0.f)$
Explanation: e Biased exponent (shown in decimal). f Fraction (in binary).		

Figure 19-5. Values of Nonzero Numbers

Programming Note: ANSI/IEEE Standard 754-1985 specifies minimum requirements for the extended format but does not include details. The BFP extended format meets these requirements, far exceeding them in the area of precision.

Classes of BFP Data

There are six classes of BFP data, which include numeric and related nonnumeric entities. Each data item consists of a sign, an exponent, and a significand. The exponent is biased such that all biased exponents are nonnegative unsigned numbers and the minimum biased exponent is zero. The significand consists of an explicit fraction and an implicit unit bit to the left of the binary point. The sign bit is zero for plus and one for minus.

All nonzero finite numbers permitted by a given format have a unique BFP representation. There are no unnormalized numbers, which numbers might allow multiple representations for the same values, and there are no unnormalized arithmetic operations. Tiny numbers of a magnitude below the minimum normal number in a given format are represented as *subnormal* numbers, but those values are also represented uniquely. The implied unit bit of a normal number is one, and that of a subnormal number or a zero is zero.

The six classes of BFP data are summarized in Figure 19-6 on page 19-4.

Data Class	Sign	Biased Exponent	Unit Bit*	Fraction
Zero	\pm	0	0	0
Subnormal numbers	\pm	0**	0	Not 0
Normal numbers	\pm	Not 0, not all ones	1	Any
Infinity	\pm	All ones	—	0
Quiet NaN	\pm	All ones	—	F0=1, Fr=any
Signaling NaN	\pm	All ones	—	F0=0, Fr \neq 0
Explanation:				
—	Does not apply.			
*	The unit bit is implied.			
**	The biased exponent is treated arithmetically as if it had the value one.			
F0	Leftmost bit of fraction.			
Fr	Remaining bits of fraction.			
NaN	Not-a-number.			

Figure 19-6. Classes of BFP Data

The instruction TEST DATA CLASS may be used to determine the class of a BFP operand.

Zeros

Zeros have a biased exponent of zero and a zero fraction. The implied unit bit is zero. A +0 is distinct from -0, except that comparison treats them as equal.

Subnormal Numbers

Subnormal numbers are numbers which are smaller than the smallest normal number and greater than zero in magnitude. They have a biased exponent of zero and a nonzero fraction. The biased exponent is treated arithmetically as if it were one, which causes the exponent to be the minimum exponent. The implied unit bit is zero. (In ANSI/IEEE Standard 754-1985, subnormal numbers were called “denormalized numbers”.)

Normal Numbers

Normal numbers have a biased exponent greater than zero but less than all ones. The implied unit bit is one, and the fraction may have any value. (In some early documentation, normal numbers were called “normalized numbers”.)

Infinities

An infinity is represented by a biased exponent of all ones and a zero fraction. Infinities can participate in most arithmetic operations and give a consistent result, usually infinity. In comparisons, $+\infty$ compares greater than any finite number, and $-\infty$ compares less than any finite number.

Signaling and Quiet NaNs

A NaN (not-a-number) entity is represented by a biased exponent of all ones and a nonzero fraction. NaNs are produced in place of a numeric result after an invalid operation when there is no interruption. NaNs may also be used by the program to flag special operands, such as the contents of an uninitialized storage area.

There are two types of NaNs, signaling and quiet. A signaling NaN (SNaN) is distinguished from the corresponding quiet NaN (QNaN) by the leftmost fraction bit: zero for the SNaN and one for the QNaN. A special QNaN is supplied as the default result for an IEEE-invalid-operation exception; it has a plus sign and a leftmost fraction bit of one, with the remaining fraction bits being set to zeros.

Normally, QNaNs are just propagated during computations so that they will remain visible at the end. An

SNaN operand causes an IEEE-invalid-operation exception. If the IEEE-invalid-operation mask (FPC 0.0) is zero, the result is the corresponding QNaN, which is produced by setting the leftmost fraction bit to one, and the IEEE-invalid-operation flag (FPC 1.0) is set to one. If the IEEE-invalid-operation mask (FPC 0.0) is one, the operation is suppressed, and a data exception for IEEE-invalid operation occurs.

Programming Notes:

1. The program can generate and assign meanings to any nonzero fraction values of a NaN. The CPU propagates those values unchanged, except that an SNaN is changed to the corresponding QNaN if the IEEE-invalid-operation mask bit is zero, and conversion to a narrower format may truncate significant bits on the right.
2. ANSI/IEEE Standard 754-1985 requires SNaNs to signal the invalid-operation exception for the arithmetic, comparison, and conversion operations that are part of the standard, but it makes it an implementation option whether copying an SNaN without a change of format signals the exception. In the appendix, the standard also makes it an implementation option whether SNaNs should signal the invalid-operation exception for the recommended functions of copying the sign, taking the absolute value, reversing the sign, and testing the data class of a datum.

The above functions generally correspond to the instructions LOAD, LOAD COMPLEMENT, LOAD NEGATIVE, LOAD POSITIVE, and TEST DATA CLASS. These instructions do not signal the invalid-operation exception but, instead, treat SNaNs like any other data; giving an exception would be disruptive when the intention is to include SNaNs. TEST DATA CLASS does not give an exception since it is the instruction with which to test for the presence of SNaNs.

3. LOAD AND TEST signals the invalid-operation exception when the operand is an SNaN. This instruction, in conjunction with the above instructions, gives the program the choice of either option permitted by ANSI/IEEE Standard 754-1985.
4. Load-type instructions which change the precision signal the invalid-operation exception when the operand is an SNaN, as this is required by ANSI/IEEE Standard 754-1985.

BFP-Format Conversion

BFP format conversion is described in the section “IEEE Same-Radix Format Conversion” on page 9-23.

BFP Rounding

BFP rounding is described in the section “IEEE Rounding” on page 9-13.

BFP Comparison

BFP comparison is described in the section “IEEE Comparison” on page 9-24.

Remainder

The instruction DIVIDE TO INTEGER produces two floating-point results, an exact integer quotient and the corresponding remainder. The remainder is defined as follows:

Let

a = Dividend
 b = Divisor
 q = Exact quotient ($a \div b$)
 r = Remainder

in the selected floating-point format. Then

$$r = a - b \cdot n$$

where n is an integer. If q is an integer, then n equals q. Otherwise, n is obtained by rounding q according to a final-quotient-rounding method.

When the final-quotient-rounding method is round to nearest with ties to even or round toward zero, the remainder is exact for any finite dividend and any nonzero divisor. The remainder cannot overflow.

If the integer quotient has a value that lies outside the range of the operand format, a scaled result is provided.

In certain cases where the number of bits in the integer quotient exceeds or may exceed the maximum number of bits provided in the precision of the operand and format, partial results are produced, and more

than one execution of the instruction is required to obtain the final result; this may be done with a simple instruction loop.

Partial results are produced when the precise quotient is not an integer and the two integers closest to this precise quotient cannot both be represented exactly in the precision of the quotient. This situation exists when the precise quotient is greater than 2^P , where P is the precision of the operand format, and the remainder is not zero. When the remainder is zero, then the quotient is an integer, and the number of bits required to represent the quotient is never more than the precision of the target.

Programming Note: The remainder result of DIVIDE TO INTEGER with a final-quotient-rounding method of round to nearest with ties to even corresponds to the *Remainder* function in ANSI/IEEE Standard 754-1985. This function is similar to the *MOD* function found in some languages and to the mathematical

modulo function, but they are not the same. They differ in the definition of n:

Remainder	n is q rounded to nearest with ties to even.
modulo	n is q rounded toward $-\infty$.
MOD	n is q rounded toward 0.

Another important difference is that implementations of modulo and MOD may put range restrictions on the result because they may simply use the DIVIDE instruction and accept its range restrictions.

The MOD definition provides an exact result, as does Remainder, but the modulo definition may result in rounding errors.

The differences between the various methods may be illustrated by the simple example of computing a divided by b to obtain an integer quotient n, where a is a series of integers, and b is +4 or -4. Figure 19-7 shows the results for the three definitions.

	a																	
	-8	-7	-6	-5	-4	-3	-2	-1	-0	+0	+1	+2	+3	+4	+5	+6	+7	+8
Remainder																		
b=+4: n	-2	-2	-2	-1	-1	-1	-0	-0	-0	+0	+0	+0	+1	+1	+1	+2	+2	+2
r	-0	+1	+2	-1	-0	1	-2	-1	-0	+0	+1	+2	-1	+0	+1	-2	-1	+0
b=-4: n	+2	+2	+2	+1	+1	+1	+0	+0	+0	-0	-0	-0	-1	-1	-1	-2	-2	-2
r	-0	+1	+2	-1	-0	+1	-2	-1	-0	+0	+1	+2	-1	+0	+1	-2	-1	+0
MOD																		
b=+4: n	-2	-1	-1	-1	-1	0	0	0	0	0	0	0	+1	+1	+1	+1	+2	
r	0	-3	-2	-1	0	-3	-2	-1	0	+1	+2	+3	0	+1	+2	+3	0	
b=-4: n	+2	+1	+1	+1	+1	0	0	0	0	0	0	0	-1	-1	-1	-1	-2	
r	0	-3	-2	-1	0	-3	-2	-1	0	+1	+2	+3	0	+1	+2	+3	0	
modulo																		
b=+4: n	-2	-2	-2	-2	-1	-1	-1	-1	0	0	0	0	+1	+1	+1	+1	+2	
r	0	+1	+2	+3	0	+1	+2	+3	0	+1	+2	+3	0	+1	+2	+3	0	
b=-4: n	+2	+1	+1	+1	+1	0	0	0	0	-1	-1	-1	-1	-2	-2	-2	-2	
r	0	-3	-2	-1	0	-3	-2	-1	0	-3	-2	-1	0	-3	-2	-1	0	
Explanation:																		
a	Dividend.																	
b	Divisor.																	
n	Integer quotient.																	
r	Result (Remainder, MOD, or modulo).																	

Figure 19-7. Comparison of Remainder with MOD and Modulo

The result of Remainder lies in the range of zero to one-half the divisor, inclusive, in magnitude. A zero

result is defined to have the sign of the dividend. A zero divisor is invalid.

The modulo and MOD results can both be computed from the Remainder result; the reverse may not be true, because of rounding errors and, depending on the implementation, range restrictions.

An extreme example of the rounding error that can occur with the modulo definition is the following, where the result is restricted to two significant decimal digits:

$\text{modulo}(0.01, -95) = -94.99$, which rounds to -95

$\text{Remainder}(0.01, -95) = 0.01$

The properly rounded modulo result is completely wrong since it is equal to the divisor instead of being smaller in magnitude. The Remainder result is exact and can be used to compute the theoretical result of modulo.

Remainder is included as an arithmetic operation because of its usefulness in argument reduction when computing elementary transcendental functions. Thus, $\text{SIN}(X)$ can be computed to full precision for any value of X in degrees by first reducing the argument to $\text{Remainder}(X, 360)$.

IEEE Exceptions

The IEEE exceptions for the BFP instructions are described in the section “IEEE Exceptions” on page 9-17.

Result Figures

Concise descriptions of the results produced by many of the BFP instructions are made by means of figures which contain columns and rows representing all possible combinations of BFP data class for the source operands of an instruction. The information shown at the intersection of a row and a column is one or more symbols representing the result or results produced for that particular combination of source-operand data classes. Explanations of the symbols used are contained in each figure. In many cases, the explanation of a particular result is in the form of a cross reference to another figure. In many cases, the information shown at the intersection consists of several symbols separated by commas. All such results are produced unless one of the results is

a program interruption. In the case of a program interruption, the operation is suppressed or completed as shown in Figure 19-9 on page 19-8.

Data-Exception Codes (DXC) and Abbreviations

Figure 19-8 shows IEEE exceptions and flag abbreviations that are used in the result figures, and it explains the symbols “Xi:” and “Xz:” that are used in the figures. Bits 0-4 (i,z,o,u,x) of the eight-bit data-exception code (DXC) in byte 2 of the FPC register are trap flags and correspond to the same bits in bytes 0 and 1 of the register (IEEE masks and IEEE flags). The trap flag for an exception, instead of the IEEE flag, is set to one when an interruption for the exception is enabled by the corresponding IEEE mask bit. Bit 5 of byte 2 (y) is used in conjunction with bit 4, inexact (x), to indicate that the result has been incremented in magnitude.

Exception		FPC IEEE Mask Bit	IEEE Flag	
Name	Abbr.		FPC Bit	Abbr.
IEEE invalid operation	Xi^1	0.0	1.0	SFi
IEEE division by zero	Xz^2	0.1	1.1	SFz
IEEE overflow	Xo	0.2	1.2	SFo
IEEE underflow	Xu	0.3	1.3	SFu
IEEE inexact	Xx	0.4	1.4	SFx

Explanation:

¹ The symbol “Xi:” followed by a list of results in a figure indicates that, when FPC 0.0 is zero, then instruction execution is completed by setting SFi (FPC 1.0) to one and producing the indicated results; and when FPC 0.0 is one, then instruction execution is suppressed, the data exception code (DXC) is set to 80 hex, and a program interruption for a data exception occurs.

² The symbol “Xz:” followed by a list of results in a figure indicates that, when FPC 0.1 is zero, then instruction execution is completed by setting SFz (FPC 1.1) to one and producing the indicated results; and when FPC 0.1 is one, then instruction execution is suppressed, the data exception code (DXC) is set to 40 hex, and a program interruption for a data exception occurs.

Figure 19-8. IEEE Exception and Flag Abbreviations

Figure 19-9 on page 19-8 shows the various DXCs that can be indicated, the associated instruction endings, and abbreviations that are used for the DXCs in

the result figures. (The abbreviation “PID” stands for “program interruption for a data exception.”)

Abbr.	DXC (Hex)	Data-Exception-Code Name	Instruction Ending
PIDx	08	IEEE inexact and truncated	Complete
PIDy	0C	IEEE inexact and incremented	Complete
PIDu	10	IEEE underflow, exact	Complete, scale exponent
PIDux	18	IEEE underflow, inexact and truncated	Complete, scale exponent
PIDuy	1C	IEEE underflow, inexact and incremented	Complete, scale exponent
PIDo	20	IEEE overflow, exact	Complete, scale exponent
PIDox	28	IEEE overflow, inexact and truncated	Complete, scale exponent
PIDoy	2C	IEEE overflow, inexact and incremented	Complete, scale exponent
PIDz	40	IEEE division by zero	Suppress
PIDi	80	IEEE invalid operation	Suppress

Figure 19-9. IEEE Data-Exception Codes (DXC) and Abbreviations

Instructions

The BFP instructions and their mnemonics and operation codes are listed in Figure 19-10 on page 19-9.

The figure indicates, in the column labeled “Characteristics”, the instruction format, when the condition code is set, the instruction fields that designate access registers, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

All BFP instructions are subject to the AFP-register-control bit, bit 45 of control register 0. For the BFP instructions to be executed successfully, the AFP-register-control bit must be one; otherwise, a BFP-instruction data exception, DXC 2, is recognized.

Mnemonics for the BFP instructions are distinguished from the corresponding HFP instructions by a B in the mnemonic. Mnemonics for the BFP instructions have an R as the last letter or the letter next to the last letter when the instruction is in the RRE or RRF format. Mnemonics for the BFP instructions have an A as the last letter when the instruction is an alternate instruction, which uses additional modifier fields not available to the original instruction. Certain letters are used for BFP instructions to represent operand-format length, as follows:

F Thirty-two-bit signed fixed point
 G Sixty-four-bit signed fixed point
 LF Thirty-two-bit unsigned fixed point
 LG Sixty-four-bit unsigned fixed point
 D Long
 E Short
 X Extended

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For a register-to-register operation using COMPARE (short), for example, CEBR is the mnemonic and R₁,R₂ the operand designation.

Programming Note: The floating-point extension facility includes the following BFP features:

- The following new BFP instructions are added:
 - CONVERT FROM LOGICAL (CXLFBR, CDLFBR, CELFBR, CXLGBR, CDLGBR, and CELGBR).
 - CONVERT TO LOGICAL (CLFXBR, CLFDBR, CLFEBR, CLGXBR, CLGDBR, and CLGEBR)
- One new value of the effective rounding method field is assigned to support the round to prepare for shorter precision rounding method for CONVERT TO FIXED, DIVIDE TO INTEGER, and LOAD FP INTEGER

- An IEEE-inexact-exception control (XxC) is added to CONVERT TO FIXED and LOAD FP INTEGER.

- An effective rounding method field and an IEEE-inexact-exception control (XxC) are added to CONVERT FROM FIXED and LOAD ROUNDED.

Name	Mnemonic	Characteristics						Op Code	Page
ADD (extended BFP)	AXBR	RRE C	SP	Db Xi	Xo Xu Xx			B34A	19-11
ADD (long BFP)	ADB	RXE C	A	Db Xi	Xo Xu Xx		B ₂	ED1A	19-11
ADD (long BFP)	ADBR	RRE C		Db Xi	Xo Xu Xx			B31A	19-11
ADD (short BFP)	AEB	RXE C	A	Db Xi	Xo Xu Xx		B ₂	ED0A	19-11
ADD (short BFP)	AEBR	RRE C		Db Xi	Xo Xu Xx			B30A	19-11
COMPARE (extended BFP)	CXBR	RRE C	SP	Db Xi				B349	19-16
COMPARE (long BFP)	CDB	RXE C	A	Db Xi			B ₂	ED19	19-16
COMPARE (long BFP)	CDBR	RRE C		Db Xi				B319	19-16
COMPARE (short BFP)	CEB	RXE C	A	Db Xi			B ₂	ED09	19-16
COMPARE (short BFP)	CEBR	RRE C		Db Xi				B309	19-16
COMPARE AND SIGNAL (extended BFP)	KXBR	RRE C	SP	Db Xi				B348	19-17
COMPARE AND SIGNAL (long BFP)	KDB	RXE C	A	Db Xi			B ₂	ED18	19-17
COMPARE AND SIGNAL (long BFP)	KDBR	RRE C		Db Xi				B318	19-17
COMPARE AND SIGNAL (short BFP)	KEB	RXE C	A	Db Xi			B ₂	ED08	19-17
COMPARE AND SIGNAL (short BFP)	KEBR	RRE C		Db Xi				B308	19-17
CONVERT FROM FIXED (32 to extended BFP)	CXFBR	RRE	SP	Db				B396	19-18
CONVERT FROM FIXED (32 to extended BFP)	CXFBRA	RRF-e F	SP	Db				B396	19-18
CONVERT FROM FIXED (32 to long BFP)	CDFBR	RRE		Db				B395	19-18
CONVERT FROM FIXED (32 to long BFP)	CDFBRA	RRF-e F	SP	Db				B395	19-18
CONVERT FROM FIXED (32 to short BFP)	CEFBR	RRE		Db	Xx			B394	19-18
CONVERT FROM FIXED (32 to short BFP)	CEFBRA	RRF-e F	SP	Db	Xx			B394	19-18
CONVERT FROM FIXED (64 to extended BFP)	CXGBR	RRE N	SP	Db				B3A6	19-18
CONVERT FROM FIXED (64 to extended BFP)	CXGBRA	RRF-e F	SP	Db				B3A6	19-18
CONVERT FROM FIXED (64 to long BFP)	CDGBR	RRE N		Db	Xx			B3A5	19-18
CONVERT FROM FIXED (64 to long BFP)	CDGBRA	RRF-e F	SP	Db	Xx			B3A5	19-18
CONVERT FROM FIXED (64 to short BFP)	CEGBR	RRE N		Db	Xx			B3A4	19-18
CONVERT FROM FIXED (64 to short BFP)	CEGBRA	RRF-e F	SP	Db	Xx			B3A4	19-18
CONVERT FROM LOGICAL (32 to extended BFP)	CXLFBR	RRF-e F	SP	Db				B392	19-20
CONVERT FROM LOGICAL (32 to long BFP)	CDLFBR	RRF-e F	SP	Db				B391	19-20
CONVERT FROM LOGICAL (32 to short BFP)	CELFBR	RRF-e F	SP	Db	Xx			B390	19-20
CONVERT FROM LOGICAL (64 to extended BFP)	CXLGBR	RRF-e F	SP	Db				B3A2	19-20
CONVERT FROM LOGICAL (64 to long BFP)	CDLGBR	RRF-e F	SP	Db	Xx			B3A1	19-20
CONVERT FROM LOGICAL (64 to short BFP)	CELGBR	RRF-e F	SP	Db	Xx			B3A0	19-20
CONVERT TO FIXED (extended BFP to 32)	CFXBR	RRF-e C	SP	Db Xi	Xx			B39A	19-21
CONVERT TO FIXED (extended BFP to 32)	CFXBRA	RRF-e C F	SP	Db Xi	Xx			B39A	19-21
CONVERT TO FIXED (extended BFP to 64)	CGXBR	RRF-e C N	SP	Db Xi	Xx			B3AA	19-21
CONVERT TO FIXED (extended BFP to 64)	CGXBRA	RRF-e C F	SP	Db Xi	Xx			B3AA	19-21
CONVERT TO FIXED (long BFP to 32)	CFDBR	RRF-e C	SP	Db Xi	Xx			B399	19-21
CONVERT TO FIXED (long BFP to 32)	CFDBRA	RRF-e C F	SP	Db Xi	Xx			B399	19-21
CONVERT TO FIXED (long BFP to 64)	CGDBR	RRF-e C N	SP	Db Xi	Xx			B3A9	19-21
CONVERT TO FIXED (long BFP to 64)	CGDBRA	RRF-e C F	SP	Db Xi	Xx			B3A9	19-21
CONVERT TO FIXED (short BFP to 32)	CFEBR	RRF-e C	SP	Db Xi	Xx			B398	19-21
CONVERT TO FIXED (short BFP to 32)	CFEBRA	RRF-e C F	SP	Db Xi	Xx			B398	19-21
CONVERT TO FIXED (short BFP to 64)	CGEBR	RRF-e C N	SP	Db Xi	Xx			B3A8	19-21
CONVERT TO FIXED (short BFP to 64)	CGEBRA	RRF-e C F	SP	Db Xi	Xx			B3A8	19-21
CONVERT TO LOGICAL (extended BFP to 32)	CLFXBR	RRF-e C F	SP	Db Xi	Xx			B39E	19-23
CONVERT TO LOGICAL (extended BFP to 64)	CLGXBR	RRF-e C F	SP	Db Xi	Xx			B3AE	19-23

Figure 19-10. Summary of BFP Instructions (Part 1 of 3)

Name	Mne- monic	Characteristics						Op Code	Page
CONVERT TO LOGICAL (long BFP to 32)	CLFDBR	RRF-e C F		SP	Db Xi	Xx		B39D	19-23
CONVERT TO LOGICAL (long BFP to 64)	CLGDBR	RRF-e C F		SP	Db Xi	Xx		B3AD	19-23
CONVERT TO LOGICAL (short BFP to 32)	CLFEBr	RRF-e C F		SP	Db Xi	Xx		B39C	19-23
CONVERT TO LOGICAL (short BFP to 64)	CLGEBR	RRF-e C F		SP	Db Xi	Xx		B3AC	19-23
DIVIDE (extended BFP)	DXBR	RRE		SP	Db Xi	Xz Xo Xu Xx		B34D	19-26
DIVIDE (long BFP)	DDB	RXE	A		Db Xi	Xz Xo Xu Xx	B ₂	ED1D	19-26
DIVIDE (long BFP)	DDBR	RRE			Db Xi	Xz Xo Xu Xx		B31D	19-26
DIVIDE (short BFP)	DEB	RXE	A		Db Xi	Xz Xo Xu Xx	B ₂	ED0D	19-26
DIVIDE (short BFP)	DEBR	RRE			Db Xi	Xz Xo Xu Xx		B30D	19-26
DIVIDE TO INTEGER (long BFP)	DIDBR	RRF-b C		SP	Db Xi	Xu Xx		B35B	19-27
DIVIDE TO INTEGER (short BFP)	DIEBR	RRF-b C		SP	Db Xi	Xu Xx		B353	19-27
LOAD AND TEST (extended BFP)	LTXBR	RRE C		SP	Db Xi			B342	19-30
LOAD AND TEST (long BFP)	LTDBR	RRE C			Db Xi			B312	19-30
LOAD AND TEST (short BFP)	LTEBR	RRE C			Db Xi			B302	19-30
LOAD COMPLEMENT (extended BFP)	LCXBR	RRE C		SP	Db			B343	19-30
LOAD COMPLEMENT (long BFP)	LCDBR	RRE C			Db			B313	19-30
LOAD COMPLEMENT (short BFP)	LCEBR	RRE C			Db			B303	19-30
LOAD FP INTEGER (extended BFP)	FIXBR	RRF-e		SP	Db Xi	Xx		B347	19-31
LOAD FP INTEGER (extended BFP)	FIXBRA	RRF-e F		SP	Db Xi	Xx		B347	19-31
LOAD FP INTEGER (long BFP)	FIDBR	RRF-e		SP	Db Xi	Xx		B35F	19-31
LOAD FP INTEGER (long BFP)	FIDBRA	RRF-e F		SP	Db Xi	Xx		B35F	19-31
LOAD FP INTEGER (short BFP)	FIEBR	RRF-e		SP	Db Xi	Xx		B357	19-31
LOAD FP INTEGER (short BFP)	FIEBRA	RRF-e F		SP	Db Xi	Xx		B357	19-31
LOAD LENGTHENED (long to extended BFP)	LXDB	RXE	A	SP	Db Xi		B ₂	ED05	19-33
LOAD LENGTHENED (long to extended BFP)	LXDBR	RRE		SP	Db Xi			B305	19-32
LOAD LENGTHENED (short to extended BFP)	LXEB	RXE	A	SP	Db Xi		B ₂	ED06	19-33
LOAD LENGTHENED (short to extended BFP)	LXEBr	RRE		SP	Db Xi			B306	19-32
LOAD LENGTHENED (short to long BFP)	LDEB	RXE	A		Db Xi		B ₂	ED04	19-33
LOAD LENGTHENED (short to long BFP)	LDEBR	RRE			Db Xi			B304	19-32
LOAD NEGATIVE (extended BFP)	LNxBR	RRE C		SP	Db			B341	19-33
LOAD NEGATIVE (long BFP)	LNDBR	RRE C			Db			B311	19-33
LOAD NEGATIVE (short BFP)	LNEBR	RRE C			Db			B301	19-33
LOAD POSITIVE (extended BFP)	LPxBR	RRE C		SP	Db			B340	19-33
LOAD POSITIVE (long BFP)	LPDBR	RRE C			Db			B310	19-33
LOAD POSITIVE (short BFP)	LPEBR	RRE C			Db			B300	19-33
LOAD ROUNDED (extended to long BFP)	LDXBR	RRE		SP	Db Xi	Xo Xu Xx		B345	19-34
LOAD ROUNDED (extended to long BFP)	LDXBRA	RRF-e F		SP	Db Xi	Xo Xu Xx		B345	19-34
LOAD ROUNDED (extended to short BFP)	LEXBR	RRE		SP	Db Xi	Xo Xu Xx		B346	19-34
LOAD ROUNDED (extended to short BFP)	LEXBRA	RRF-e F		SP	Db Xi	Xo Xu Xx		B346	19-34
LOAD ROUNDED (long to short BFP)	LEDBR	RRE			Db Xi	Xo Xu Xx		B344	19-34
LOAD ROUNDED (long to short BFP)	LEDBRA	RRF-e F		SP	Db Xi	Xo Xu Xx		B344	19-34
MULTIPLY (extended BFP)	MXBR	RRE		SP	Db Xi	Xo Xu Xx		B34C	19-35
MULTIPLY (long BFP)	MDB	RXE	A		Db Xi	Xo Xu Xx	B ₂	ED1C	19-35
MULTIPLY (long BFP)	MDBR	RRE			Db Xi	Xo Xu Xx		B31C	19-35
MULTIPLY (long to extended BFP)	MXDB	RXE	A	SP	Db Xi		B ₂	ED07	19-35
MULTIPLY (long to extended BFP)	MXDBR	RRE		SP	Db Xi			B307	19-35
MULTIPLY (short BFP)	MEEB	RXE	A		Db Xi	Xo Xu Xx	B ₂	ED17	19-35
MULTIPLY (short BFP)	MEEBR	RRE			Db Xi	Xo Xu Xx		B317	19-35
MULTIPLY (short to long BFP)	MDEB	RXE	A		Db Xi		B ₂	ED0C	19-35
MULTIPLY (short to long BFP)	MDEBR	RRE			Db Xi			B30C	19-35

Figure 19-10. Summary of BFP Instructions (Part 2 of 3)

Name	Mne- monic	Characteristics							Op Code	Page	
MULTIPLY AND ADD (long BFP)	MADB	RXF	A	Db	Xi	Xo	Xu	Xx	B ₂	ED1E	19-37
MULTIPLY AND ADD (long BFP)	MADBR	RRD		Db	Xi	Xo	Xu	Xx		B31E	19-37
MULTIPLY AND ADD (short BFP)	MAEB	RXF	A	Db	Xi	Xo	Xu	Xx	B ₂	ED0E	19-37
MULTIPLY AND ADD (short BFP)	MAEBR	RRD		Db	Xi	Xo	Xu	Xx		B30E	19-37
MULTIPLY AND SUBTRACT (long BFP)	MSDB	RXF	A	Db	Xi	Xo	Xu	Xx	B ₂	ED1F	19-37
MULTIPLY AND SUBTRACT (long BFP)	MSDBR	RRD		Db	Xi	Xo	Xu	Xx		B31F	19-37
MULTIPLY AND SUBTRACT (short BFP)	MSEB	RXF	A	Db	Xi	Xo	Xu	Xx	B ₂	ED0F	19-37
MULTIPLY AND SUBTRACT (short BFP)	MSEBR	RRD		Db	Xi	Xo	Xu	Xx		B30F	19-37
SQUARE ROOT (extended BFP)	SQXBR	RRE	SP	Db	Xi			Xx		B316	19-38
SQUARE ROOT (long BFP)	SQDB	RXE	A	Db	Xi			Xx	B ₂	ED15	19-38
SQUARE ROOT (long BFP)	SQDBR	RRE		Db	Xi			Xx		B315	19-38
SQUARE ROOT (short BFP)	SQEB	RXE	A	Db	Xi			Xx	B ₂	ED14	19-38
SQUARE ROOT (short BFP)	SQEBR	RRE		Db	Xi			Xx		B314	19-38
SUBTRACT (extended BFP)	SXBR	RRE C	SP	Db	Xi	Xo	Xu	Xx		B34B	19-39
SUBTRACT (long BFP)	SDB	RXE C	A	Db	Xi	Xo	Xu	Xx	B ₂	ED1B	19-39
SUBTRACT (long BFP)	SDBR	RRE C		Db	Xi	Xo	Xu	Xx		B31B	19-39
SUBTRACT (short BFP)	SEB	RXE C	A	Db	Xi	Xo	Xu	Xx	B ₂	ED0B	19-39
SUBTRACT (short BFP)	SEBR	RRE C		Db	Xi	Xo	Xu	Xx		B30B	19-39
TEST DATA CLASS (extended BFP)	TCXB	RXE C	SP	Db						ED12	19-39
TEST DATA CLASS (long BFP)	TCDB	RXE C		Db						ED11	19-39
TEST DATA CLASS (short BFP)	TCEB	RXE C		Db						ED10	19-39
Explanation:											
A	Access exceptions for logical addresses.										
B ₂	B ₂ field designates an access register in the access-register mode.										
C	Condition code is set.										
Db	BFP-instruction data exception.										
F	Floating-point extension facility										
N	Instruction is new in z/Architecture as compared to ESA/390.										
RRD	RRE instruction format.										
RRE	RRE instruction format.										
RRF	RRF instruction format.										
RXE	RXE instruction format.										
RXF	RXF instruction format.										
SP	Specification exception.										
Xi	IEEE invalid-operation exception.										
Xo	IEEE overflow exception.										
Xu	IEEE underflow exception.										
Xx	IEEE inexact exception.										
Xz	IEEE division-by-zero exception.										

Figure 19-10. Summary of BFP Instructions (Part 3 of 3)

ADDMnemonic1 R₁,R₂ [RRE]

Op Code	////////	R ₁	R ₂
0	16	24	28 31

Mnemonic1	Op Code	Operands
AEBR	'B30A'	Short BFP
ADBR	'B31A'	Long BFP
AXBR	'B34A'	Extended BFP

Mnemonic2 R₁,D₂(X₂,B₂)

[RXE]

Op Code	R ₁	X ₂	B ₂	D ₂	////////	Op Code
0	8	12	16	20	32	40 47

Mnemonic2	Op Code	Operands
AEB	'ED0A'	Short BFP
ADB	'ED1A'	Long BFP

The second operand is added to the first operand, and the sum is placed at the first-operand location.

If both operands are finite numbers, they are added algebraically, forming an intermediate sum. The intermediate sum, if nonzero, is rounded to the operand format according to the current BFP rounding mode. The sum is then placed at the result location.

The sign of the sum is determined by the rules of algebra. This also applies to a result of zero:

- If the result of rounding a nonzero intermediate sum is zero, the sign of the zero result is the sign of the intermediate sum.
- If the sum of two operands with opposite signs is exactly zero, the sign of the result is plus in all rounding methods except round toward $-\infty$, in which method the sign is minus.
- The sign of the sum x plus x is the sign of x , even when x is zero.

If one operand is an infinity and the other is a finite number, the result is that infinity. If both operands are infinities of the same sign, the result is the same infinity. If the two operands are infinities of opposite signs, an IEEE-invalid-operation exception is recognized.

See Figure 19-12 on page 19-12 for a detailed description of the results of this instruction. (Figure 19-11 is referred to by Figure 19-12.)

For AXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero

- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Result is a NaN

IEEE Exceptions:

- Invalid operation
- Overflow
- Underflow
- Inexact

Program Exceptions:

- Access (fetch, operand 2 of AEB and ADB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (AXBR only)

Programming Note: Interchanging the two operands in a BFP addition does not affect the value of the sum when the result is numeric. This is not true, however, when both operands are QNaNs, in which case the result is the first operand; or when both operands are SNaNs and the IEEE-invalid-operation mask bit in the FPC register is zero, in which case the result is the QNaN derived from the first operand.

Value of Result (r)	Condition Code
$r=0$	cc0
$r<0$	cc1
$r>0$	cc2
Explanation:	
ccn	Condition code is set to n.

Figure 19-11. Condition Code for Resultant Sum

First Operand (a) Is	Results for ADD (a+b) when Second Operand (b) Is									
	$-\infty$	-Nn	-Dn	-0	+0	+Dn	+Nn	$+\infty$	QNaN	SNaN
$-\infty$	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	Xi: T(dNaN), cc3	T(b), cc3	Xi: T(b*), cc3
-Nn	T($-\infty$), cc1	R(a+b), cc1	R(a+b), cc1	T(a), cc1	T(a), cc1	R(a+b), cc1	R(a+b), ccrs	T($+\infty$), cc2	T(b), cc3	Xi: T(b*), cc3
-Dn	T($-\infty$), cc1	R(a+b), cc1	R(a+b), cc1	R(a), cc1	R(a), cc1	R(a+b), ccrs	R(a+b), cc2	T($+\infty$), cc2	T(b), cc3	Xi: T(b*), cc3
-0	T($-\infty$), cc1	T(b), cc1	R(b), cc1	T(-0), cc0	Rezd, cc0	R(b), cc2	T(b), cc2	T($+\infty$), cc2	T(b), cc3	Xi: T(b*), cc3
+0	T($-\infty$), cc1	T(b), cc1	R(b), cc1	Rezd, cc0	T(+0), cc0	R(b), cc2	T(b), cc2	T($+\infty$), cc2	T(b), cc3	Xi: T(b*), cc3

Figure 19-12. Results: ADD

First Operand (a) Is	Results for ADD (a+b) when Second Operand (b) Is									
	$-\infty$	-Nn	-Dn	-0	+0	+Dn	+Nn	$+\infty$	QNaN	SNaN
+Dn	T($-\infty$), cc1	R(a+b), cc1	R(a+b), ccrs	R(a), cc2	R(a), cc2	R(a+b), cc2	R(a+b), cc2	T($+\infty$), cc2	T(b), cc3	Xi: T(b*), cc3
+Nn	T($-\infty$), cc1	R(a+b), ccrs	R(a+b), cc2	T(a), cc2	T(a), cc2	R(a+b), cc2	R(a+b), cc2	T($+\infty$), cc2	T(b), cc3	Xi: T(b*), cc3
$+\infty$	Xi: T(dNaN), cc3	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T(b), cc3	Xi: T(b*), cc3
QNaN	T(a), cc3	T(a), cc3	T(a), cc3	T(a), cc3	T(a), cc3	T(a), cc3	T(a), cc3	T(a), cc3	T(a), cc3	Xi: T(b*), cc3
SNaN	Xi: T(a*), cc3	Xi: T(a*), cc3	Xi: T(a*), cc3	Xi: T(a*), cc3	Xi: T(a*), cc3	Xi: T(a*), cc3	Xi: T(a*), cc3	Xi: T(a*), cc3	Xi: T(a*), cc3	Xi: T(a*), cc3

Explanation:

*	The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.
ccn	Condition code is set to n.
ccrs	Condition code is set according to the resultant sum. See Figure 19-11 on page 19-12.
dNaN	Default NaN.
Dn	Subnormal number.
Nn	Normal number.
R(v)	Rounding and range action is performed on the value v. See Figure 19-13 on page 19-14.
Rezd	Exact zero-difference result. See Figure 19-13 on page 19-14.
T(x)	The value x is placed at the target operand location.
Xi:	IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 19-12. Results: ADD

Range of v	Case	Nontrap Result (r) when Effective Rounding Method Is				
		To Nearest with ties to even	Toward 0	Toward $+\infty$	Toward $-\infty$	Prepare for shorter precision
$v < -N_{\max}$, $g < -N_{\max}$	Overflow	$-\infty^1$	$-N_{\max}$	$-N_{\max}$	$-\infty^1$	$-N_{\max}$
$v < -N_{\max}$, $g = -N_{\max}$	Normal	$-N_{\max}$	$-N_{\max}$	$-N_{\max}$	—	$-N_{\max}$
$-N_{\max} \leq v \leq -N_{\min}$	Normal	g	g	g	g	g
$-N_{\min} < v \leq -D_{\min}$	Tiny	d^*	d	d	d^*	d
$-D_{\min} < v < -D_{\min}/2$	Tiny	$-D_{\min}$	-0	-0	$-D_{\min}$	$-D_{\min}$
$-D_{\min}/2 \leq v < 0$	Tiny	-0	-0	-0	$-D_{\min}$	$-D_{\min}$
$v = 0$	Exact zero difference ²	$+0$	$+0$	$+0$	-0	$+0$
$0 < v \leq +D_{\min}/2$	Tiny	$+0$	$+0$	$+D_{\min}$	$+0$	$+D_{\min}$
$+D_{\min}/2 < v < +D_{\min}$	Tiny	$+D_{\min}$	$+0$	$+D_{\min}$	$+0$	$+D_{\min}$
$+D_{\min} \leq v < +N_{\min}$	Tiny	d^*	d	d^*	d	d
$+N_{\min} \leq v \leq +N_{\max}$	Normal	g	g	g	g	g
$+N_{\max} < v$, $g = +N_{\max}$	Normal	$+N_{\max}$	$+N_{\max}$	—	$+N_{\max}$	$+N_{\max}$
$+N_{\max} < v$, $+N_{\max} < g$	Overflow	$+\infty^1$	$+N_{\max}$	$+\infty^1$	$+N_{\max}$	$+N_{\max}$

Explanation:

— This situation cannot occur.

* The rounded value, in the extreme case, may be N_{\min} . In this case, the exceptions are underflow, inexact and incremented.

¹ The nontrap result r is considered to have been incremented.

² The exact-zero-difference case applies only to ADD, SUBTRACT, MULTIPLY AND ADD, and MULTIPLY AND SUBTRACT. For all other BFP operations, a zero result is detected by inspection of the source operands without use of the $R(v)$ function.

d The denormalized value. The value derived when the precise intermediate value (v) is rounded to the format of the target, including both precision and bounded exponent range. Except as explained in note *, this is a subnormal number.

g The precision-rounded value. The value derived when the precise intermediate value (v) is rounded to the precision of the target, but assuming an unbounded exponent range.

v Precise intermediate value. This is the value, before rounding, assuming unbounded precision and an unbounded exponent range. For LOAD ROUNDED, v is the source value (a).

D_{\min} Smallest (in magnitude) representable subnormal number in the target format.

N_{\max} Largest (in magnitude) representable finite number in the target format.

N_{\min} Smallest (in magnitude) representable normal number in the target format.

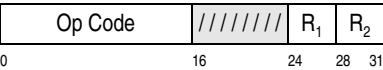
Figure 19-13. Action for $R(v)$: Rounding and Range Function (Part 1 of 2)

Case	Is r Inexact (r≠v)	Overflow Mask (FPC 0.2)	Underflow Mask (FPC 0.3)	IEEE Inexact Exception Control (XxC) ²	Inexact Mask (FPC 0.4)	Is r Incremented (r > v)	Is g Inexact (g≠v)	Is g Incremented (g > v)	Results
Overflow	Yes ¹	0	—	1	—	—	—	—	T(r), SFo←1
Overflow	Yes ¹	0	—	0	0	—	—	—	T(r), SFo←1, SFx←1
Overflow	Yes ¹	0	—	0	1	No	—	—	T(r), SFo←1, PIDx(08)
Overflow	Yes ¹	0	—	0	1	Yes	—	—	T(r), SFo←1, PIDy(0C)
Overflow	Yes ¹	1	—	—	—	—	No	No ¹	Tw(g÷Ψ), PIDo(20)
Overflow	Yes ¹	1	—	—	—	—	Yes	No	Tw(g÷Ψ), PIDox(28)
Overflow	Yes ¹	1	—	—	—	—	Yes	Yes	Tw(g÷Ψ), PIDoy(2C)
Normal	No	—	—	—	—	—	—	—	T(r)
Normal	Yes	—	—	1	—	—	—	—	T(r)
Normal	Yes	—	—	0	0	—	—	—	T(r), SFx←1
Normal	Yes	—	—	0	1	No	—	—	T(r), PIDx(08)
Normal	Yes	—	—	0	1	Yes	—	—	T(r), PIDy(0C)
Tiny	No	—	0	—	—	—	—	—	T(r)
Tiny	No	—	1	—	—	—	No ¹	No ¹	Tw(g÷Ψ), PIDu(10)
Tiny	Yes	—	0	1	—	—	—	—	T(r), SFu←1
Tiny	Yes	—	0	0	0	—	—	—	T(r), SFu←1, SFx←1
Tiny	Yes	—	0	0	1	No	—	—	T(r), SFu←1, PIDx(08)
Tiny	Yes	—	0	0	1	Yes	—	—	T(r), SFu←1, PIDy(0C)
Tiny	Yes	—	1	—	—	—	No	No ¹	Tw(g÷Ψ), PIDu(10)
Tiny	Yes	—	1	—	—	—	Yes	No	Tw(g÷Ψ), PIDux(18)
Tiny	Yes	—	1	—	—	—	Yes	Yes	Tw(g÷Ψ), PIDuy(1C)
Explanation: — The results do not depend on this condition or mask bit. ¹ This condition is true by virtue of the state of some condition to the left of this column. ² The IEEE-inexact-exception control (XxC) is defined only if the floating-point extension facility is installed. Ψ Scale factor. For overflow, $\Psi = 2^{+\alpha}$. For underflow, $\Psi = 2^{-\alpha}$. The unsigned scaling exponent (α) depends on the type of operation and operand format. For all BFP operations except LOAD ROUNDED, α depends on the target format and is 192 for short, 1536 for long, and 24576 for extended. For LOAD ROUNDED, α depends on the source format, and is 512 for long and 8192 for extended. g The precision-rounded value. The value derived when the precise intermediate value (v) is rounded to the precision of the target, but assuming an unbounded exponent range. r Nontrap result as defined in Part 1 of this figure. v Precise intermediate value. This is the value, before rounding, assuming unbounded precision and unbounded exponent range. PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. See Figure 19-9 on page 19-8. SFo IEEE overflow flag, FPC 1.2. SFu IEEE underflow flag, FPC 1.3. SFx IEEE inexact flag, FPC 1.4. T(x) The value x is placed at the target operand location. Tw(x) The scaled result x is placed at the target operand location. For all operations except BFP LOAD ROUNDED, the scaled result is in the format and length of the target and rounded to the precision of the target. For LOAD ROUNDED, the scaled result is in the format and length of the source, but rounded to the precision of the target.									

Figure 19-13. Action for R(v): Rounding and Range Function (Part 2 of 2)

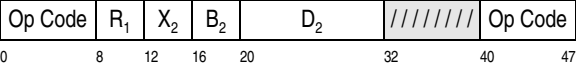
COMPARE

Mnemonic1 R₁,R₂ [RRE]



Mnemonic1	Op Code	Operands
CEBR	'B309'	Short BFP
CDBR	'B319'	Long BFP
CXBR	'B349'	Extended BFP

Mnemonic2 R₁,D₂(X₂,B₂) [RXE]



Mnemonic2	Op Code	Operands
CEB	'ED09'	Short BFP
CDB	'ED19'	Long BFP

The first operand is compared with the second operand, and the condition code is set to indicate the result.

If both operands are finite numbers, the comparison is algebraic and follows the procedure for BFP subtraction, except that the difference is discarded after setting the condition code, and both operands remain unchanged. If the difference is exactly zero with either sign, the operands are equal; this includes zero operands (so +0 equals -0). If a nonzero difference is positive or negative, the first operand is high or low, respectively.

+∞ compares greater than any finite number, and all finite numbers compare greater than -∞. Two infinity operands of like sign compare equal.

Numeric comparison is exact, and the condition code is determined for finite operands as if range and precision were unlimited. No overflow or underflow exception can occur.

If either or both operands are QNaNs and neither operand is an SNaN, the comparison result is unordered, and condition code 3 is set.

If either or both operands are SNaNs, an IEEE-invalid-operation exception is recognized. If the IEEE invalid-operation mask bit is one, a program interrup-

tion for a data exception with DXC 80 hex (IEEE invalid operation) occurs. If the IEEE-invalid-operation mask bit is zero, the IEEE-invalid-operation flag bit is set to one, and instruction execution is completed by setting condition code 3.

See Figure 19-14 on page 19-17 for a detailed description of the results of this instruction.

For CXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 Operands unordered

IEEE Exceptions:

- Invalid operation

Program Exceptions:

- Access (fetch, operand 2 of CEB and CDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (CXBR only)

Programming Notes:

- COMPARE may be used to implement those comparisons which are required by ANSI/IEEE Standard 754-1985 to not recognize an exception when the result is unordered due to a QNaN.
- ANSI/IEEE Standard 754-1985 requires that it be possible to compare BFP operands in different formats. To accomplish this, LOAD LENGTHENED may be used before COMPARE to convert the shorter operand to the same format as the longer.

First Operand (a) Is	Results for COMPARE (a:b) when Second Operand (b) Is							
	$-\infty$	-Fn	-0	+0	+Fn	$+\infty$	QNaN	SNaN
$-\infty$	cc0	cc1	cc1	cc1	cc1	cc1	cc3	Xi: cc3
-Fn	cc2	C(a:b)	cc1	cc1	cc1	cc1	cc3	Xi: cc3
-0	cc2	cc2	cc0	cc0	cc1	cc1	cc3	Xi: cc3
+0	cc2	cc2	cc0	cc0	cc1	cc1	cc3	Xi: cc3
+Fn	cc2	cc2	cc2	cc2	C(a:b)	cc1	cc3	Xi: cc3
$+\infty$	cc2	cc2	cc2	cc2	cc2	cc0	cc3	Xi: cc3
QNaN	cc3	cc3	cc3	cc3	cc3	cc3	cc3	Xi: cc3
SNaN	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3

Explanation:

ccn Condition code is set to n.
C(a:b) Basic compare results. See Figure 19-15.
Fn Finite nonzero number (includes both subnormal and normal).
Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 19-14. Results: COMPARE

Relation of Value (a) to Value (b)	Condition Code for C(a:b)
a=b	cc0
a<b	cc1
a>b	cc2

Explanation:

ccn Condition code is set to n.

Figure 19-15. Basic Compare Results

COMPARE AND SIGNAL

Mnemonic1 R₁,R₂ [RRE]

Op Code	////////	R ₁	R ₂
0	16	24	28 31

Mnemonic1	Op Code	Operands
KEBR	'B308'	Short BFP
KDBR	'B318'	Long BFP
KXBR	'B348'	Extended BFP

Mnemonic2 R₁,D₂(X₂,B₂) [RXE]

Op Code	R ₁	X ₂	B ₂	D ₂	////////	Op Code
0	8	12	16	20	32	40 47

Mnemonic2	Op Code	Operands
KEB	'ED08'	Short BFP
KDB	'ED18'	Long BFP

The first operand is compared with the second operand, and the condition code is set to indicate the result. The operation is the same as for COMPARE

except that QNaN operands cause an IEEE-invalid-operation exception to be recognized. Thus, QNaN operands are treated as if they were SNaNs.

See Figure 19-16 on page 19-18 for a detailed description of the results of this instruction.

For KXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Operands equal
- 1 First operand low
- 2 First operand high
- 3 Operands unordered

IEEE Exceptions:

- Invalid operation

Program Exceptions:

- Access (fetch, operand 2 of KEB and KDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (KXBR only)

Programming Notes:

1. COMPARE AND SIGNAL may be used to implement those comparisons which are required by

ANSI/IEEE Standard 754-1985 to recognize an exception when the result is unordered due to a QNaN.

2. ANSI/IEEE Standard 754-1985 requires that it be possible to compare BFP operands in different

formats. To accomplish this, LOAD LENGTHENED may be used before COMPARE AND SIGNAL to convert the shorter operand to the same format as the longer.

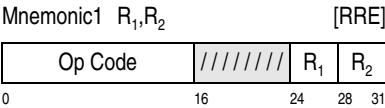
First Operand (a) Is	Results for COMPARE AND SIGNAL (a:b) when Second Operand (b) Is						
	$-\infty$	-Fn	-0	+0	+Fn	$+\infty$	NaN
$-\infty$	cc0	cc1	cc1	cc1	cc1	cc1	Xi: cc3
-Fn	cc2	C(a:b)	cc1	cc1	cc1	cc1	Xi: cc3
-0	cc2	cc2	cc0	cc0	cc1	cc1	Xi: cc3
+0	cc2	cc2	cc0	cc0	cc1	cc1	Xi: cc3
+Fn	cc2	cc2	cc2	cc2	C(a:b)	cc1	Xi: cc3
$+\infty$	cc2	cc2	cc2	cc2	cc2	cc0	Xi: cc3
NaN	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3

Explanation:

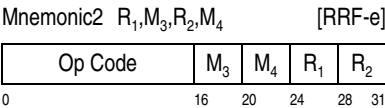
ccn Condition code is set to n.
C(a:b) Basic compare results. See Figure 19-15 on page 19-17
Fn Finite nonzero number (includes both subnormal and normal).
Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 19-16. Results: COMPARE AND SIGNAL

CONVERT FROM FIXED



Mnemonic1	Op Code	Operands
CEFBR	'B394'	32-bit binary-integer operand, short BFP result
CDFBR	'B395'	32-bit binary-integer operand, long BFP result
CXFBR	'B396'	32-bit binary-integer operand, extended BFP result
CEGBR	'B3A4'	64-bit binary-integer operand, short BFP result
CDGBR	'B3A5'	64-bit binary-integer operand, long BFP result
CXGBR	'B3A6'	64-bit binary-integer operand, extended BFP result



Mnemonic2	Op Code	Operands
CEFBRA	'B394'	32-bit binary-integer operand, short BFP result
CDFBRA	'B395'	32-bit binary-integer operand, long BFP result
CXFBRA	'B396'	32-bit binary-integer operand, extended BFP result
CEGBRA	'B3A4'	64-bit binary-integer operand, short BFP result
CDGBRA	'B3A5'	64-bit binary-integer operand, long BFP result
CXGBRA	'B3A6'	64-bit binary-integer operand, extended BFP result

The fixed-point second operand is converted to the BFP format, and the result is placed at the first-operand location.

The second operand is a signed binary integer that is located in the general register designated by R_2 . A 32-bit operand is in bit positions 32-63 of the register.

When the floating-point extension facility is installed, the converted result is rounded by rounding as specified by the modifier in the M_3 field:

M_3	Effective Rounding Method
0	According to current BFP rounding mode
1	Round to nearest with ties away from 0

- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$

An M_3 modifier other than 0, 1, or 3-7 is invalid. The M_3 field must designate a valid modifier; otherwise, a specification exception is recognized.

When the M_3 modifier field is zero, rounding is controlled by the current BFP rounding mode specified in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current BFP rounding mode.

When the floating-point extension facility is not installed and bits 16-19 of the instruction contain a value of zero, the converted result is rounded according to the current BFP rounding mode. When the floating-point extension facility is not installed and bits 16-19 of the instruction contain a nonzero value, it is undefined whether a specification exception is recognized or an unpredictable rounding method is performed.

When the floating-point extension facility is installed, bit 1 of the M_4 field is the IEEE-inexact-exception control (XxC). Bits 0, 2, and 3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of IEEE-inexact exception is suppressed.

When the floating-point extension facility is not installed and if bits 20-23 of the instruction are zeros, recognition of IEEE-inexact exception is not suppressed; when the floating-point extension facility is not installed and if bits 20-23 of the instruction contain a nonzero value, then it is unpredictable whether recognition of IEEE-inexact exception is suppressed.

See Figure 19-17 on page 19-20 for a detailed description of the results of this instruction.

For CXFBR, CXFBRA, CXGBR, and CXGBRA, the R_1 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Inexact (CEFBR, CEFBRA, CDGBR, CDGBRA, CEGBR, CEGBRA)

Program Exceptions:

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification

Programming Note: Unassigned bits in the M_4 field are reserved for future extensions and should be set to zeros; otherwise the program may not operate compatibly in the future.

Instruction	Results for Instructions with a Single Operand (a) when Operand (a) Is							
	$-\infty$	-Fn	-0	+0	+Fn	$+\infty$	QNaN	SNaN
CONVERT FROM FIXED	–	Rf(a)	–	T(+0)	Rf(a)	–	–	–
LOAD AND TEST	T($-\infty$)	T(a)	T(-0)	T(+0)	T(a)	T(+ ∞)	T(a)	Xi: T(a*)
LOAD LENGTHENED	T($-\infty$)	T(a) ¹	T(-0)	T(+0)	T(a) ¹	T(+ ∞)	T(a) ¹	Xi: T(a*) ¹
LOAD ROUNDED	T($-\infty$)	R(a)	T(-0)	T(+0)	R(a)	T(+ ∞)	T(a) ²	Xi: T(a*) ²
SQUARE ROOT	Xi: T(dNaN)	Xi: T(dNaN)	T(-0)	T(+0)	R(\sqrt{a})	T(+ ∞)	T(a)	Xi: T(a*)

Explanation:

– This situation cannot occur.

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

¹ The operand is extended to the longer format by appending zeros on the right before it is placed at the target operand location.

² The NaN is shortened to the target format by truncating the rightmost bits.

dNaN Default NaN.

Fn Finite nonzero number (includes both subnormal and normal).

R(v) Rounding and range action is performed on the value v. See Figure 19-13 on page 19-15.

Rf(a) The value a is converted to the precise intermediate value floating-point number v, and then action R(v) is performed.

T(x) The value x is placed in the target operand location.

Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 19-17. Results: Single-Operand Instructions

CONVERT FROM LOGICAL

Mnemonic	R ₁ ,M ₃ ,R ₂ ,M ₄	[RRF-e]
Op Code	M ₃	M ₄ R ₁ R ₂
0	16	20 24 28 31

Mnemonic	Op Code	Operands
CELFBR	'B390'	32-bit binary-integer operand, short BFP result
CDLFBR	'B391'	32-bit binary-integer operand, long BFP result
CXLFBF	'B392'	32-bit binary-integer operand, extended BFP result
CELGBR	'B3A0'	64-bit binary-integer operand, short BFP result
CDLGBR	'B3A1'	64-bit binary-integer operand, long BFP result
CXLGBR	'B3A2'	64-bit binary-integer operand, extended BFP result

The fixed-point second operand is converted to the BFP format, and the result is placed at the first-operand location.

The second operand is an unsigned binary integer that is located in the general register designated by R₂. A 32-bit operand is in bit positions 32-63 of the register.

The converted result is rounded by rounding as specified by the modifier in the M₃ field:

M₃ Effective Rounding Method

- 0 According to current BFP rounding mode
- 1 Round to nearest with ties away from 0
- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward + ∞
- 7 Round toward $-\infty$

An M₃ modifier other than 0, 1, or 3-7 is invalid.

When the M₃ modifier field is zero, rounding is controlled by the current BFP rounding mode specified in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current BFP rounding mode.

The M₃ field must designate a valid modifier; otherwise, a specification exception is recognized. For CXLFBF and CXLGBR, the R₁ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Bit 1 of the M₄ field is the IEEE-inexact-exception control (XxC). Bits 0, 2, and 3 are ignored. If XxC is

zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of IEEE-inexact exception is suppressed.

The result always has a plus sign.

See Figure 19-17 on page 19-20 for a detailed description of the results of this instruction.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Inexact (CELFBR, CDLGBR, CELGBR)

Program Exceptions:

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Operation (if the floating-point extension facility is not installed)
- Specification

Programming Notes:

1. Unassigned bits in the M_4 field are reserved for future extensions and should be set to zeros; otherwise the program may not operate compatibly in the future
2. Using either the round toward 0 or the round toward $-\infty$ rounding method produces the same result.

CONVERT TO FIXED

Mnemonic1 R_1, M_3, R_2 [RRF-e]

Op Code	M_3	////	R_1	R_2
0	16	20	24	28 31

Mnemonic1	Op Code	Operands
CFEBR	'B398'	Short BFP operand, 32-bit binary-integer result
CFDBR	'B399'	Long BFP operand, 32-bit binary-integer result
CFXBR	'B39A'	Extended BFP operand, 32-bit binary-integer result
CGEBR	'B3A8'	Short BFP operand, 64-bit binary-integer result
CGDBR	'B3A9'	Long BFP operand, 64-bit binary-integer result
CGXBR	'B3AA'	Extended BFP operand, 64-bit binary-integer result

Mnemonic2 R_1, M_3, R_2, M_4 [RRF-e]

Op Code	M_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic2	Op Code	Operands
CFEBRA	'B398'	Short BFP operand, 32-bit binary-integer result
CFDBRA	'B399'	Long BFP operand, 32-bit binary-integer result
CFXBRA	'B39A'	Extended BFP operand, 32-bit binary-integer result
CGEBRA	'B3A8'	Short BFP operand, 64-bit binary-integer result
CGDBRA	'B3A9'	Long BFP operand, 64-bit binary-integer result
CGXBRA	'B3AA'	Extended BFP operand, 64-bit binary-integer result

The BFP second operand is rounded to an integer value and then converted to the fixed-point format. The result is placed at the first-operand location.

The result is a signed binary integer that is placed in the general register designated by R_1 . A 32-bit result replaces bits 32-63 of the register, and bits 0-31 of the register remain unchanged.

If the second operand is a finite number, it is rounded to an integer value by rounding as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

- 0 According to current BFP rounding mode
- 1 Round to nearest with ties away from 0
- 4 3Round to prepare for shorter precision
- Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$

An M_3 modifier other than 0, 1, or 3-7 is invalid. If the floating-point extension facility is not installed, an M_3 modifier of 3 is also invalid.

When the M_3 modifier field is zero, rounding is controlled by the current BFP rounding mode specified in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current BFP rounding mode.

The sign of the result is the sign of the second operand, except that a zero result has a plus sign.

When the floating-point extension facility is installed, bit 1 of the M_4 field is the IEEE-inexact-exception

control (XxC). Bits 0, 2, and 3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of IEEE-inexact exception is suppressed.

When the floating-point extension facility is not installed and if bits 20-23 of the instruction are zeros, recognition of IEEE-inexact exception is not sup-

pressed; when the floating-point extension facility is not installed and if bits 20-23 of the instruction contain a nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.

See Figure 19-18 for a detailed description of the results of this instruction.

Operand (a)	Is n Inexact (n≠a)	Inv.-Op. Mask (FPC 0.0)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is n Incre- mented (n > a)	Results
$-\infty \leq a < \text{MN}, n < \text{MN}$	—	0	1 ³	—	—	T(MN), SFi←1, cc3
$-\infty \leq a < \text{MN}, n < \text{MN}$	—	0	0 ⁴	0	—	T(MN), SFi←1, SFx←1, cc3 †
$-\infty \leq a < \text{MN}, n < \text{MN}$	—	0	0 ⁴	1	—	T(MN), SFi←1, cc3, PIDx(08) †
$-\infty \leq a < \text{MN}, n < \text{MN}$	—	1	—	—	—	PIDi(80)
$-\infty < a < \text{MN}, n = \text{MN}$	—	—	1 ³	—	—	T(MN), cc1
$-\infty < a < \text{MN}, n = \text{MN}$	—	—	0 ⁴	0	—	T(MN), SFx←1, cc1
$-\infty < a < \text{MN}, n = \text{MN}$	—	—	0 ⁴	1	—	T(MN), cc1, PIDx(08)
$\text{MN} \leq a < 0$	No	—	—	—	—	T(f), cc1
$\text{MN} \leq a < 0$	Yes	—	1 ³	—	—	T(f), cc1
$\text{MN} \leq a < 0$	Yes	—	0 ⁴	0	—	T(f), SFx←1, cc1
$\text{MN} \leq a < 0$	Yes	—	0 ⁴	1	No	T(f), cc1, PIDx(08)
$\text{MN} \leq a < 0$	Yes	—	0 ⁴	1	Yes	T(f), cc1, PIDy(0C)
-0	No ¹	—	—	—	—	T(0), cc0
+0	No ¹	—	—	—	—	T(0), cc0
$0 < a \leq \text{MP}$	No	—	—	—	—	T(f), cc2
$0 < a \leq \text{MP}$	Yes	—	1 ³	—	—	T(f), cc2
$0 < a \leq \text{MP}$	Yes	—	0 ⁴	0	—	T(f), SFx←1, cc2
$0 < a \leq \text{MP}$	Yes	—	0 ⁴	1	No	T(f), cc2, PIDx(08)
$0 < a \leq \text{MP}$	Yes	—	0 ⁴	1	Yes	T(f), cc2, PIDy(0C)
$\text{MP} < a < +\infty, n = \text{MP}^2$	—	—	1 ³	—	—	T(MP), cc2
$\text{MP} < a < +\infty, n = \text{MP}^2$	—	—	0 ⁴	0	—	T(MP), SFx←1, cc2
$\text{MP} < a < +\infty, n = \text{MP}^2$	—	—	0 ⁴	1	—	T(MP), cc2, PIDx(08)
$\text{MP} < a \leq +\infty, n > \text{MP}$	—	0	1 ³	—	—	T(MP), SFi←1, cc3
$\text{MP} < a \leq +\infty, n > \text{MP}$	—	0	0 ⁴	0	—	T(MP), SFi←1, SFx←1, cc3 †
$\text{MP} < a \leq +\infty, n > \text{MP}$	—	0	0 ⁴	1	—	T(MP), SFi←1, cc3, PIDx(08) †
$\text{MP} < a \leq +\infty, n > \text{MP}$	—	1	—	—	—	PIDi(80)
NaN	—	0	1 ³	—	—	T(MN), SFi←1, cc3
NaN	—	0	0 ⁴	0	—	T(MN), SFi←1, SFx←1, cc3 †
NaN	—	0	0 ⁴	1	—	T(MN), SFi←1, cc3, PIDx(08) †

Figure 19-18. Results: CONVERT TO FIXED

Operand (a)	Is n Inexact (n≠a)	Inv.-Op. Mask (FPC 0.0)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is n Incre- mented (n > a)	Results
NaN	—	1	—	—	—	PIDi(80)
Explanation: — The results do not depend on this condition or mask bit. 1 This condition is true by virtue of the state of some condition to the left of this column. 2 The n=MP condition only applies to CFDBR, CFDBRA, CFXBR, CFXBRA, CGXBR, and CGXBRA. 3 Floating-point extension facility is installed and XxC bit in the M ₄ field is 1. 4 Bit 21 of the instruction is 0, regardless of whether the floating-point extension facility is installed. † Result differs from DFP CONVERT TO FIXED. The DFP instruction combines two cases as it does not recognize inexact for this case and does not test the inexact mask. ccn Condition code is set to n. f The value n converted to a fixed-point result. n The value derived when the source value (a) is rounded to a floating-point integer using the effective rounding method. MN Maximum negative number representable in the target fixed-point format. MP Maximum positive number representable in the target fixed-point format. PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. Figure 19-9 on page 19-8. SFi IEEE invalid-operation flag, FPC 1.0. SFx IEEE inexact flag, FPC 1.4. T(x) The value x is placed at the target operand location.						

Figure 19-18. Results: CONVERT TO FIXED

If the M₃ field designates any of the following invalid modifier values: 2 and 8-15, then a specification exception is recognized. When the floating-point extension facility is not installed, if the M₃ field designates the invalid value 3, it is undefined whether a specification exception is recognized or an unpredictable rounding method is performed.

For CFXBR, CFXBRA, CGXBR and CGXBRA, the R₂ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Source was zero
- 1 Source was less than zero
- 2 Source was greater than zero
- 3 Special case

IEEE Exceptions:

- Invalid operation
- Inexact

Program Exceptions:

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification

Programming Note: Unassigned bits in the M₄ field are reserved for future extensions and should be set to zeros; otherwise the program may not operate compatibly in the future.

CONVERT TO LOGICAL

Mnemonic R₁,M₃,R₂,M₄ [RRF-e]

Op Code	M ₃	M ₄	R ₁	R ₂
0	16	20	24	28 31

Mnemonic	Op Code	Operands
CLFEBR	'B39C'	Short BFP operand, 32-bit binary-integer result
CLFDBR	'B39D'	Long BFP operand, 32-bit binary-integer result
CLFXBR	'B39E'	Extended BFP operand, 32-bit binary-integer result
CLGEBR	'B3AC'	Short BFP operand, 64-bit binary-integer result
CLGDBR	'B3AD'	Long BFP operand, 64-bit binary-integer result
CLGXBR	'B3AE'	Extended BFP operand, 64-bit binary-integer result

The BFP second operand is rounded to an integer value and then converted to the fixed-point format. The result is placed at the first-operand location.

The result is an unsigned binary integer that is placed in the general register designated by R_1 . A 32-bit result replaces bits 32-63 of the register, and bits 0-31 of the register remain unchanged.

If the second operand is a finite number, it is rounded to an integer value by rounding as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

- 0 According to current BFP rounding mode
- 1 Round to nearest with ties away from 0
- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$

An M_3 modifier other than 0, 1, or 3-7 is invalid.

When the M_3 modifier field is zero, rounding is controlled by the current BFP rounding mode specified in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current BFP rounding mode.

Bit 1 of the M_4 field is the IEEE-inexact-exception control (XxC). Bits 0, 2, and 3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of IEEE-inexact exception is suppressed.

See Figure 19-19 for a detailed description of the results of this instruction.

The M_3 field must designate a valid modifier; otherwise, a specification exception is recognized. For CLFXBR and CLGXBR, the R_2 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Source was zero
- 1 Source was less than zero
- 2 Source was greater than zero
- 3 Special case

IEEE Exceptions:

- Invalid operation
- Inexact

Program Exceptions:

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Operation (if the floating-point extension facility is not installed)
- Specification

Programming Note: Unassigned bits in the M_4 field are reserved for future extensions and should be set to zeros; otherwise the program may not operate compatibly in the future.

Operand (a)	Is n Inexact (n≠a)	Inv.-Op. Mask (FPC 0.0)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is n Incremented (n > a)	Results
$-\infty \leq a < 0, n < 0$	—	0	1	—	—	T(0), SFi←1, cc3
$-\infty \leq a < 0, n < 0$	—	0	0	0	—	T(0), SFi←1, SFx←1, cc3 †
$-\infty \leq a < 0, n < 0$	—	0	0	1	—	T(0), SFi←1, cc3, PIDx(08) †
$-\infty \leq a < 0, n < 0$	—	1	—	—	—	PIDi(80)
$-\infty < a < 0, n = 0$	—	—	1	—	—	T(0), cc1
$-\infty < a < 0, n = 0$	—	—	0	0	—	T(0), SFx←1, cc1
$-\infty < a < 0, n = 0$	—	—	0	1	—	T(0), cc1, PIDx(08)
-0	No ¹	—	—	—	—	T(0), cc0
+0	No ¹	—	—	—	—	T(0), cc0
$0 < a \leq \text{MU}$	No	—	—	—	—	T(f), cc2
$0 < a \leq \text{MU}$	Yes	—	1	—	—	T(f), cc2
$0 < a \leq \text{MU}$	Yes	—	0	0	—	T(f), SFx←1, cc2
$0 < a \leq \text{MU}$	Yes	—	0	1	No	T(f), cc2, PIDx(08)
$0 < a \leq \text{MU}$	Yes	—	0	1	Yes	T(f), cc2, PIDy(0C)
$\text{MU} < a < +\infty, n = \text{MU}$	—	—	1	—	—	T(MU), cc2
$\text{MU} < a < +\infty, n = \text{MU}$	—	—	0	0	—	T(MU), SFx←1, cc2
$\text{MU} < a < +\infty, n = \text{MU}$	—	—	0	1	—	T(MU), cc2, PIDx(08)
$\text{MU} < a \leq +\infty, n > \text{MU}$	—	0	1	—	—	T(MU), SFi←1, cc3
$\text{MU} < a \leq +\infty, n > \text{MU}$	—	0	0	0	—	T(MU), SFi←1, SFx←1, cc3 †
$\text{MU} < a \leq +\infty, n > \text{MU}$	—	0	0	1	—	T(MU), SFi←1, cc3, PIDx(08) †
$\text{MU} < a \leq +\infty, n > \text{MU}$	—	1	—	—	—	PIDi(80)
NaN	—	0	1	—	—	T(0), SFi←1, cc3
NaN	—	0	0	0	—	T(0), SFi←1, SFx←1, cc3 †
NaN	—	0	0	1	—	T(0), SFi←1, cc3, PIDx(08) †
NaN	—	1	—	—	—	PIDi(80)

Explanation:

— The results do not depend on this condition or mask bit.

¹ This condition is true by virtue of the state of some condition to the left of this column.

† Result differs from DFP CONVERT TO LOGICAL. The DFP instruction combines two cases as it does not recognize inexact for this case and does not test the inexact mask.

ccn Condition code is set to n.

f The value n converted to a fixed-point result.

n The value derived when the source value (a) is rounded to a floating-point integer using the effective rounding method.

MU Maximum unsigned number representable in the target fixed-point format.

PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. Figure 19-9 on page 19-8.

SFi IEEE invalid-operation flag, FPC 1.0.

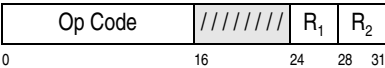
SFx IEEE inexact flag, FPC 1.4.

T(x) The value x is placed at the target operand location.

Figure 19-19. Results: CONVERT TO LOGICAL

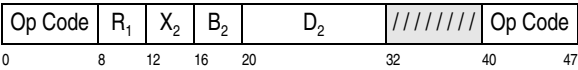
DIVIDE

Mnemonic1 R₁,R₂ [RRE]



Mnemonic1	Op Code	Operands
DEBR	'B30D'	Short BFP
DDBR	'B31D'	Long BFP
DXBR	'B34D'	Extended BFP

Mnemonic2 R₁,D₂(X₂,B₂) [RXE]



Mnemonic2	Op Code	Operands
DEB	'ED0D'	Short BFP
DDB	'ED1D'	Long BFP

The first operand (the dividend) is divided by the second operand (the divisor), and the quotient is placed at the first-operand location. No remainder is preserved.

If the divisor is nonzero and both the dividend and divisor are finite numbers, the first operand is divided by the second operand to form an intermediate quotient. The intermediate quotient, if nonzero, is

rounded to the target format according to the current BFP rounding mode.

When the dividend is a finite number and the divisor is infinity, the result is zero.

The sign of the quotient, if the quotient is numeric, is the exclusive or of the operand signs. This includes the sign of a zero or infinite quotient.

If the divisor is zero but the dividend is a finite number, an IEEE-division-by-zero exception is recognized. If the dividend and divisor are both zero, or if both are infinity, regardless of sign, an IEEE-invalid-operation exception is recognized.

See Figure 19-20 on page 19-26 for a detailed description of the results of this instruction.

For DXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Dividend (a)	Results for DIVIDE (a÷b) when Divisor (b) Is							
	-∞	-Fn	-0	+0	+Fn	+∞	QNaN	SNaN
-∞	Xi: T(dNaN)	T(+∞)	T(+∞)	T(-∞)	T(-∞)	Xi: T(dNaN)	T(b)	Xi: T(b*)
-Fn	T(+0)	R(a÷b)	Xz: T(+∞)	Xz: T(-∞)	R(a÷b)	T(-0)	T(b)	Xi: T(b*)
-0	T(+0)	T(+0)	Xi: T(dNaN)	Xi: T(dNaN)	T(-0)	T(-0)	T(b)	Xi: T(b*)
+0	T(-0)	T(-0)	Xi: T(dNaN)	Xi: T(dNaN)	T(+0)	T(+0)	T(b)	Xi: T(b*)
+Fn	T(-0)	R(a÷b)	Xz: T(-∞)	Xz: T(+∞)	R(a÷b)	T(+0)	T(b)	Xi: T(b*)
+∞	Xi: T(dNaN)	T(-∞)	T(-∞)	T(+∞)	T(+∞)	Xi: T(dNaN)	T(b)	Xi: T(b*)
QNaN	T(a)	T(a)	T(a)	T(a)	T(a)	T(a)	T(a)	Xi: T(b*)
SNaN	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)

Explanation:

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

Fn Finite nonzero number (includes both subnormal and normal).

R(v) Rounding and range action is performed on the value v. See Figure 19-13 on page 19-14.

T(x) The value x is placed at the target operand location.

Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Xz: IEEE division-by-zero exception. The results shown are produced only when FPC 0.1 is zero.

Figure 19-20. Results: DIVIDE

IEEE Exceptions:

- Invalid operation
- Division by zero
- Overflow

- Underflow
- Inexact

Program Exceptions:

- Access (fetch, operand 2 of DEB and DDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (DXBR only)

DIVIDE TO INTEGER

Mnemonic R_1, R_3, R_2, M_4 [RRF-b]

Op Code	R_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic	Op Code	Operands
DIEBR	'B353'	Short BFP
DIDBR	'B35B'	Long BFP

The first operand (the dividend) is divided by the second operand (the divisor). An integer quotient in BFP form is produced and placed at the third-operand location. The remainder replaces the dividend at the first-operand location. The first, second, and third operands must be in different registers. The condition code indicates whether partial or final results have been produced.

The remainder result is

$$r = a - b \cdot n$$

where a is the dividend, b the divisor, and n an integer obtained by rounding the precise quotient

$$q = a \div b.$$

The first-operand result is r with the sign determined by the above expression. The third-operand result is n with a sign that is the exclusive or of the dividend and divisor signs.

If the precise quotient is not an integer and the two integers closest to this precise quotient cannot both be represented exactly in the precision of the quotient, then a partial quotient and partial remainder are formed. (In all other cases, a final quotient and final remainder are formed.) This partial quotient n and the corresponding partial remainder

$$r = a - b \cdot n$$

are used as the results. The sign of a partial remainder is the same as the sign of the dividend. The sign

of a partial quotient is the exclusive or of the dividend and divisor signs.

If the remainder is zero, then the precise quotient is an integer and can be represented exactly in the precision of the quotient.

The M_4 field, called the modifier field, specifies rounding of the final quotient. This rounding is called the "final-quotient-rounding method" as contrasted to the "current BFP rounding mode" specified by the BFP rounding-mode bits in the FPC register. The final quotient is rounded according to the final-quotient-rounding method. The final-quotient-rounding method affects only the final quotient; partial quotients are rounded toward zero.

Since the partial quotient is rounded toward zero, the partial remainder is always exact. For the final-quotient-rounding methods of round toward 0, round to nearest with ties to even, and round to nearest with ties away from 0, the final remainder is exact. For the final-quotient-rounding methods of round toward $+\infty$ and round toward $-\infty$, the final remainder may not be exact.

The final quotient is rounded to an integer by rounding as specified by the modifier in the M_4 field:

M_4 Final-Quotient-Rounding Method

- 0 According to current BFP rounding mode
- 1 Round to nearest with ties away from 0
- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$

When the floating-point extension facility is installed, a modifier of 2 and 8-15 is invalid. If the floating-point extension facility is not installed, a modifier of 2-3 and 8-15 is invalid.

When the modifier field is zero, rounding of the final quotient is controlled by the current BFP rounding mode specified in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current BFP rounding mode.

Underflow is recognized only on the final remainder, not on the partial remainder.

For the final-quotient-rounding methods of round toward $+\infty$, round toward $-\infty$, and round to prepare for shorter precision, the final remainder may not be exact. When, in these cases, the final remainder is inexact, it is rounded according to the current BFP rounding mode specified in the FPC register.

The sign of a zero quotient is the exclusive or of the divisor and dividend signs.

A zero remainder has the sign of the dividend.

See Figure 19-21 on page 19-28 for a detailed description of the results of this instruction.

If the quotient exponent is greater than the largest exponent that can be represented in the operand format, the correct remainder or partial remainder still is produced, and the third-operand result is the correct value, but with the exponent reduced by 192 or 1536 for short or long operands, respectively. The condition code indicates this out-of-range condition.

If the M_4 field designates any of the following invalid modifier values: 2 and 8-15, then a specification exception is recognized. When the floating-point extension facility is not installed, if the M_4 field designates the invalid value 3, it is undefined whether a specification exception is recognized or an unpredictable rounding method is performed.

The R_1 , R_2 , and R_3 fields must designate different registers; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Remainder final; normal quotient
- 1 Remainder final; quotient overflow or NaN
- 2 Remainder partial; normal quotient
- 3 Remainder partial; quotient overflow

IEEE Exceptions:

- Invalid operation
- Underflow
- Inexact

Dividend (a)	Results for DIVIDE TO INTEGER (a+b) when Divisor (b) Is							QNaN	SNaN
	$-\infty$	-Fn	-0	+0	+Fn	$+\infty$			
$-\infty$	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	T(b), cc1		Xi: T(b*), cc1
-Fn	T(a,+0), cc0	D(a,b)	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	D(a,b)	T(a,-0), cc0	T(b), cc1		Xi: T(b*), cc1
-0	T(-0,+0), cc0	T(-0,+0), cc0	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	T(-0,-0), cc0	T(-0,-0), cc0	T(b), cc1		Xi: T(b*), cc1
+0	T(+0,-0), cc0	T(+0,-0), cc0	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	T(+0,+0), cc0	T(+0,+0), cc0	T(b), cc1		Xi: T(b*), cc1
+Fn	T(a,-0), cc0	D(a,b)	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	D(a,b)	T(a,+0), cc0	T(b), cc1		Xi: T(b*), cc1
$+\infty$	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	Xi: T(dNaN), cc1	T(b), cc1		Xi: T(b*), cc1
QNaN	T(a), cc1	T(a), cc1	T(a), cc1	T(a), cc1	T(a), cc1	T(a), cc1	T(a), cc1		Xi: T(b*), cc1
SNaN	Xi: T(a*), cc1	Xi: T(a*), cc1	Xi: T(a*), cc1	Xi: T(a*), cc1	Xi: T(a*), cc1	Xi: T(a*), cc1	Xi: T(a*), cc1		Xi: T(a*), cc1

Explanation:

- * The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.
- ccn Condition code is set to n.
- D(a,b) Basic divide-to-integer results. See Part 2 of this figure.
- Fn Finite nonzero number (includes both subnormal and normal).
- T(r,q) Results r (the remainder) and q (the quotient) are placed in target operands 1 and 3, respectively.
- T(x) Value x is placed in both target operands 1 and 3.
- Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 19-21. Results: DIVIDE TO INTEGER (Part 1 of 2)

$ q < 2^P$	$r = 0$	Case [#]	Is r Tiny	Is r Inexact	Underflow Mask (FPC 0.3)	Inexact mask (FPC 0.4)	Quotient Overflow	Is r Incremented	Results for D(a,b)
Yes	Yes	Final	No ¹	No ¹	—	—	No ¹	—	T(r,n), cc0
Yes	No	Final	No	No	—	—	No ¹	—	T(r,n), cc0
Yes	No	Final	Yes	No ¹	0	—	No ¹	—	T(r,n), cc0
Yes	No	Final	Yes	No ¹	1	—	No ¹	No ¹	T(r \div Ψ , n), cc0, PIDu(10)
Yes	No	Final	No	Yes	—	0	No ¹	—	T(r,n), SFx \leftarrow 1, cc0
Yes	No	Final	No	Yes	—	1	No ¹	No	T(r,n), cc0, PIDx(08)
Yes	No	Final	No	Yes	—	1	No ¹	Yes	T(r,n), cc0, PIDy(0C)
No	Yes	Final	No ¹	No ¹	—	—	No	—	T(r,n), cc0
No	Yes	Final	No ¹	No ¹	—	—	Yes	—	T(r, n \div Ψ), cc1
No	No	Partial	— ²	No ¹	—	—	No	—	T(r,n), cc2
No	No	Partial	— ²	No ¹	—	—	Yes	—	T(r, n \div Ψ), cc3

Explanation:

— The results do not depend on this condition or mask bit.

¹ This condition is true by virtue of the state of some condition to the left of this column. That is, when $|q| < 2^P$, there cannot be a quotient overflow; the cases of remainder is zero, tiny, or inexact are mutually exclusive; and when r is exact, it is not incremented.

² Underflow is not recognized for a partial remainder.

Each entry of this column is derived from the entries of the two left columns, and helps determine the specific rounding method used in some columns on the right and determine the resulting condition code setting.

Ψ Scale factor. For overflow, $\Psi = 2^{+\alpha}$. For underflow, $\Psi = 2^{-\alpha}$. The unsigned scaling exponent (α) depends on the operand format and is 192 for short and 1536 for long.

|q| The absolute value of q, where q is the precise intermediate value of a-b before rounding, assuming unbounded precision and unbounded exponent range.

cc0 Condition code is set to 0 (remainder final; normal quotient).

cc1 Condition code is set to 1 (remainder final; quotient overflow).

cc2 Condition code is set to 2 (remainder partial; normal quotient).

cc3 Condition code is set to 3 (remainder partial; quotient overflow).

n Integer quotient. $n = q$, rounded toward 0 for partial results and rounded according to the final-quotient-rounding method for final results. The sign of the integer quotient, including the cases of partial and final, scaled value, and zero, is the exclusive or of the signs of the dividend (a) and divisor (b).

r Remainder. $r = a - b \cdot n$. A partial remainder is always exact; no rounding is necessary. The sign of a partial remainder is always the same as the sign of the dividend (a). A final remainder is rounded according to the current BFP rounding mode (if necessary) specified in the FPC register. The sign of a zero remainder is the same as the sign of the dividend (a). The sign of a nonzero final remainder is determined by the rules of algebra.

P Precision of the operand, which depends on the target format: $P = 24$ for short and 53 for long.

PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. See Figure 19-9 on page 19-8.

SFi IEEE invalid-operation flag, FPC 1.0.

SFu IEEE underflow flag, FPC 1.3.

SFx IEEE inexact flag, FPC 1.4.

T(r,n) Results r (the remainder) and n (the integer quotient) are placed in target operands 1 and 3, respectively.

Figure 19-21. Results: DIVIDE TO INTEGER (Part 2 of 2)

Program Exceptions:

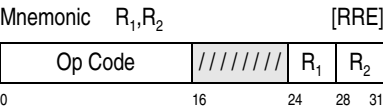
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification

Programming Notes:

1. The Remainder operation, as defined in ANSI/IEEE Standard 754-1985, is produced by issuing DIVIDE TO INTEGER in an iterative loop, with the M_4 field set to 4.

- 2. The rounding specifications of round to nearest with ties to even, round toward 0, and round toward $-\infty$ permit the instruction to be used directly to produce the Remainder, MOD, and modulo functions, respectively.
- 3. When DIVIDE TO INTEGER is used in an iterative loop, all quotients are produced in BFP format but may be considered as portions of a multiple-precision fixed-point number.
- 4. In the case when the resulting remainder is sub-normal, ANSI/IEEE Standard 754-1985 requires that if traps are implemented and the underflow mask is one, then an underflow trap must occur. To accomplish this, DIVIDE TO INTEGER recognizes underflow on the final remainder but not on the partial remainder. Since in all cases when underflow exists on the partial remainder it will also exist on the final remainder, recognizing underflow on only the final remainder avoids two underflow traps to be reported for what ANSI/IEEE Standard 754-1985 considers a single Remainder operation.

LOAD AND TEST



Mnemonic	Op Code	Operands
LTEBR	'B302'	Short BFP
LTDBR	'B312'	Long BFP
LTXBR	'B342'	Extended BFP

The second operand is placed at the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code. The condition code is set the same as for a comparison of the second operand with zero.

The second operand is placed unchanged at the first-operand location. If the second operand is an SNaN, an IEEE-invalid-operation exception is recognized; if there is no interruption, the result is the corresponding QNaN.

See Figure 19-17 on page 19-20 for a detailed description of the results of this instruction.

For LTXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Result is a NaN

IEEE Exceptions:

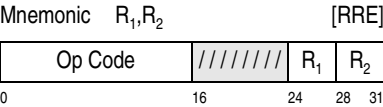
- Invalid operation

Program Exceptions:

- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (LTXBR only)

Programming Note: LOAD AND TEST signals invalid operation when the operand is an SNaN. TEST DATA CLASS may be used to test an operand if signaling is not desired.

LOAD COMPLEMENT



Mnemonic	Op Code	Operands
LCEBR	'B303'	Short BFP
LCDBR	'B313'	Long BFP
LCXBR	'B343'	Extended BFP

The second operand is placed at the first-operand location with the sign bit inverted.

The sign bit is inverted even if the operand is zero. The rest of the second operand is placed unchanged at the first-operand location. The sign is inverted for any operand, including a QNaN or SNaN, without causing an IEEE exception.

For LCXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Result is a NaN

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 2, BFP instruction
- Specification (LCXBR only)

Programming Note: LOAD COMPLEMENT does not signal invalid operation when the operand is an SNaN. LOAD AND TEST may be used in conjunction with this instruction if signaling is desired.

LOAD FP INTEGER

Mnemonic1 R_1, M_3, R_2 [RRF-e]

Op Code	M_3	////	R_1	R_2
0	16	20	24	28 31

Mnemonic1	Op Code	Operands
FIEBR	'B357'	Short BFP
FIDBR	'B35F'	Long BFP
FIXBR	'B347'	Extended BFP

Mnemonic2 R_1, M_3, R_2, M_4 [RRF-e]

Op Code	M_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic2	Op Code	Operands
FIEBRA	'B357'	Short BFP
FIDBRA	'B35F'	Long BFP
FIXBRA	'B347'	Extended BFP

The second operand is rounded to an integer value in the same floating-point format, and the result is placed at the first-operand location.

The second operand, if a finite number, is rounded to an integer value as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

- 0 According to current BFP rounding mode
- 1 Round to nearest with ties away from 0
- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$

An M_3 modifier other than 0, 1, or 3-7 is invalid. If the floating-point extension facility is not installed, an M_3 modifier of 3 is also invalid.

When the M_3 modifier field is zero, rounding is controlled by the current BFP rounding mode in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current BFP rounding mode.

In the absence of an interruption, if the second operand is an infinity or a QNaN, the result is that operand; if the second operand is an SNaN, the result is the corresponding QNaN.

The sign of the result is the sign of the second operand, even when the result is zero.

When the floating-point extension facility is installed, bit 1 of the M_4 field is the IEEE-inexact-exception control (XxC). Bits 0, 2, and 3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of IEEE-inexact exception is suppressed.

When the floating-point extension facility is not installed and if bits 20-23 of the instruction are zeros, recognition of IEEE-inexact exception is not suppressed; when the floating-point extension facility is not installed and if bits 20-23 of the instruction contain a nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.

See Figure 19-22 on page 19-32 for a detailed description of the results of this instruction.

If the M_3 field designates any of the following invalid modifier values: 2 and 8-15, then a specification exception is recognized. When the floating-point extension facility is not installed, if the M_3 field designates the invalid value 3, it is undefined whether a specification exception is recognized or an unpredictable rounding method is performed.

For FIXBR and FIXBRA, the R fields must designate valid floating-point-register pairs. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Inexact

Program Exceptions:

- Data with DXC 2, BFP instruction

- Data with DXC for IEEE exception
 - Specification
- Programming Notes:**
1. LOAD FP INTEGER rounds a BFP number to an integer value. These integers, which remain in the BFP format, should not be confused with binary integers, which have a fixed-point format.
 2. If the BFP operand is a finite number with a large enough exponent so that it is already an integer, the result value remains the same.
 3. Unassigned bits in the M₄ field are reserved for future extensions and should be set to zeros; otherwise the program may not operate compatibly in the future.

Operand (a)	Is n Inexact (n≠a)	Inv.-Op. Mask (FPC 0.0)	IEEE Inexact Exception Control (XxC) ²	Inexact Mask (FPC 0.4)	Is n Incremented (n > a)	Results
-∞	No ¹	—	—	—	—	T(-∞)
-Fn	No	—	—	—	—	T(n)
-Fn	Yes	—	1	—	—	T(n)
-Fn	Yes	—	0	0	—	T(n), SFx←1
-Fn	Yes	—	0	1	No	T(n), PIDx(08)
-Fn	Yes	—	0	1	Yes	T(n), PIDy(0C)
-0	No ¹	—	—	—	—	T(-0)
+0	No ¹	—	—	—	—	T(+0)
+Fn	No	—	—	—	—	T(n)
+Fn	Yes	—	1	—	—	T(n)
+Fn	Yes	—	0	0	—	T(n), SFx←1
+Fn	Yes	—	0	1	No	T(n), PIDx(08)
+Fn	Yes	—	0	1	Yes	T(n), PIDy(0C)
+∞	No ¹	—	—	—	—	T(+∞)
QNaN	No ¹	—	—	—	—	T(a)
SNaN	No ¹	0	—	—	—	T(a*), SFi←1
SNaN	No ¹	1	—	—	—	PIDi(80)

Explanation:

— The results do not depend on this condition or mask bit.

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

¹ This condition is true by virtue of the state of some condition to the left of this column.

² XxC is defined only if the Floating-point extension facility is installed.

n The value derived when the source value, a, is rounded to a floating-point integer using the effective rounding mode.

Fn Finite nonzero number (includes both subnormal and normal).

PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. See Figure 19-9 on page 19-8.

SFi IEEE invalid-operation flag, FPC 1.0.

SFx IEEE inexact flag, FPC 1.4.

T(x) The value x is placed at the target operand location.

Figure 19-22. Results: LOAD FP INTEGER

LOAD LENGTHENED

Mnemonic1	R ₁ ,R ₂	[RRE]
Op Code	//////	R ₁ R ₂
0	16	24 28 31

Mnemonic1	Op Code	Operands
LDEBR	'B304'	Short BFP operand 2, long BFP operand 1
LXDBR	'B305'	Long BFP operand 2, extended BFP operand 1
LXEBR	'B306'	Short BFP operand 2, extended BFP operand 1

Mnemonic2 $R_1, D_2(X_2, B_2)$ [RXE]

Op Code	R ₁	X ₂	B ₂	D ₂	////////	Op Code	
0	8	12	16	20	32	40	47

Mnemonic2	Op Code	Operands
LDEB	'ED04'	Short BFP operand 2, long BFP operand 1
LXDB	'ED05'	Long BFP operand 2, extended BFP operand 1
LXEB	'ED06'	Short BFP operand 2, extended BFP operand 1

The second operand is converted to a longer format, and the result is placed at the first-operand location.

When the second operand is a finite number, the value of the second operand is placed in the target format. The exponent of the second operand is converted to the corresponding exponent in the result format, and the fraction is extended by appending zeros on the right.

If the second operand is an infinity, the result is an infinity of the same sign. If the second operand is an SNaN, an IEEE-invalid-operation exception is recognized; if there is no interruption, the result is the corresponding QNaN with the fraction extended.

The sign of the result is the same as the sign of the source.

See Figure 19-17 on page 19-20 for a detailed description of the results of this instruction.

For LXDB, LXDBR, LXEB, and LXEBR, the R₁ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation

Program Exceptions:

- Access (fetch, operand 2 of LDEB, LXEB, and LXDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (LXEB, LXEBR, LXDB, LXDBR)

LOAD NEGATIVE

Mnemonic R_1, R_2 [RRE]

Op Code		////////	R ₁	R ₂
0	16	24	28	31

Mnemonic	Op Code	Operands
LNEBR	'B301'	Short BFP
LNDBR	'B311'	Long BFP
LNxBR	'B341'	Extended BFP

The second operand is placed at the first-operand location with the sign bit made one.

The sign bit is made one even if the operand is zero. The rest of the second operand is placed unchanged at the first-operand location. The sign is set for any operand, including a QNaN or SNaN, without causing an IEEE exception.

For LNxBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 --
- 3 Result is a NaN

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 2, BFP instruction
- Specification (LNxBR only)

Programming Note: LOAD NEGATIVE does not signal invalid operation when the operand is an SNaN. LOAD AND TEST may be used in conjunction with this instruction if signaling is desired.

LOAD POSITIVE

Mnemonic R_1, R_2 [RRE]

Op Code		////////	R ₁	R ₂
0	16	24	28	31

Mnemonic	Op Code	Operands
LPEBR	'B300'	Short BFP
LPDBR	'B310'	Long BFP
LPxBR	'B340'	Extended BFP

The second operand is placed at the first-operand location with the sign bit made zero.

The sign bit is made zero, and the rest of the second operand is placed unchanged at the first-operand location. The sign is set for any operand, including a QNaN or SNaN, without causing an IEEE exception.

For LPXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 --
- 2 Result is greater than zero
- 3 Result is a NaN

IEEE Exceptions: None.

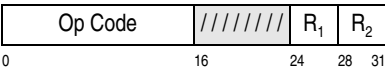
Program Exceptions:

- Data with DXC 2, BFP instruction
- Specification (LPXBR only)

Programming Note: LOAD POSITIVE does not signal invalid operation when the operand is an SNaN. LOAD AND TEST may be used in conjunction with this instruction if signaling is desired.

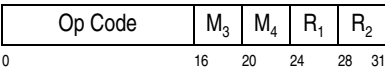
LOAD ROUNDED

Mnemonic1 R₁,R₂ [RRE]



Mnemonic1	Op Code	Operands
LEDBR	'B344'	Long BFP source, short BFP target, short or long BFP result
LDXBR	'B345'	Extended BFP source, long BFP target, long or extended BFP result
LEXBR	'B346'	Extended BFP source, short BFP target, short or extended BFP result

Mnemonic2 R₁,M₃,R₂,M₄ [RRF-e]



Mnemonic2	Op Code	Operands
LEDBRA	'B344'	Long BFP source, short BFP target, short or long BFP result
LDXBRA	'B345'	Extended BFP source, long BFP target, long or extended BFP result
LEXBRA	'B346'	Extended BFP source, short BFP target, short or extended BFP result

The second operand, in the format of the source, is rounded to the precision of the target, and the result is placed at the first-operand location.

When the floating-point extension facility is installed, the second operand, if a finite number, is rounded by rounding as specified by the modifier in the M₃ field:

M₃ Effective Rounding Method

- 0 According to current BFP rounding mode
- 1 Round to nearest with ties away from 0
- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward +∞
- 7 Round toward -∞

An M₃ modifier other than 0, 1, or 3-7 is invalid. The M₃ field must designate a valid modifier; otherwise, a specification exception is recognized.

When the M₃ modifier field is zero, rounding is controlled by the current BFP rounding mode specified in the FPC register. When the field is not zero, rounding is performed as specified by the modifier, regardless of the current BFP rounding mode.

When the floating-point extension facility is not installed and bits 16-19 of the instruction contain a value of zero, the second operand is rounded according to the current BFP rounding mode. When the floating-point extension facility is not installed and bits 16-19 of the instruction contain a nonzero value, it is undefined whether a specification exception is recognized or an unpredictable rounding method is performed.

In the absence of IEEE-trap action for overflow or underflow, the result is in the format and length of the target. However, when an IEEE overflow or an IEEE underflow is recognized and the corresponding mask bit is one, the operation is completed by producing a scaled result in the same format and length as the source but rounded to the precision of the target. For LEDBR, LEDBRA, LEXBR and LEXBRA, the result has at most 23 significand fraction bits; for LDXBR

and LDXBRA, the result has at most 52 significand fraction bits.

A short-format result replaces the leftmost 32 bits of the target register, and the rightmost 32 bit positions of the target register remain unchanged. A long-format result is placed in a floating-point register, and the other register of the floating-point register pair, if any, remains unchanged. An extended-format result is placed in a floating-point register pair.

The sign of the result is the same as the sign of the second operand.

When the floating-point extension facility is installed, bit 1 of the M_4 field is the IEEE-inexact-exception control (XxC). Bits 0, 2, and 3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of IEEE-inexact exception is suppressed.

When the floating-point extension facility is not installed and if bit 21 of the instruction is zero, recognition of IEEE-inexact exception is not suppressed; when the floating-point extension facility is not installed and if bit 21 of the instruction is one, then it is unpredictable whether recognition of IEEE-inexact exception is suppressed.

See Figure 19-17 on page 19-20 for a detailed description of the results of this instruction.

For LDXBR, LDXBRA, LEXBR, and LEXBRA, the R_1 and R_2 fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Overflow
- Underflow
- Inexact

Program Exceptions:

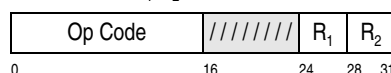
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification

Programming Notes:

1. The sign of the rounded result is the same as the sign of the operand, even when the result is zero.
2. The R_1 field for LDXBR, LDXBRA, LEXBR, and LEXBRA must designate a valid floating-point-register pair since in certain cases the result is in the extended format. In normal operation for LDXBR, LDXBRA, LEXBR and LEXBRA, the result format is long or short, respectively, and this result replaces the leftmost 32 bits or 64 bits of the target-register pair. However, when an IEEE overflow or an IEEE underflow is recognized and the corresponding mask bit is one, the operation is completed by placing a result in the extended format at the target location. Thus, the program must take into account the fact that these instructions sometimes update both registers of the pair.
3. Unassigned bits in the M_4 field are reserved for future extensions and should be set to zeros; otherwise the program may not operate compatibly in the future

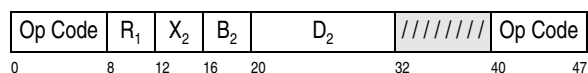
MULTIPLY

Mnemonic1 R_1, R_2 [RRE]



Mnemonic1	Op Code	Operands
MEEBR	'B317'	Short BFP
MDBR	'B31C'	Long BFP
MXBR	'B34C'	Extended BFP
MDEBR	'B30C'	Short BFP multiplier and multiplicand, long BFP product
MXDBR	'B307'	Long BFP multiplier and multiplicand, extended BFP product

Mnemonic2 $R_1, D_2(X_2, B_2)$ [RXE]



Mnemonic2	Op Code	Operands
MEEB	'ED17'	Short BFP
MDB	'ED1C'	Long BFP
MDEB	'ED0C'	Short BFP multiplier and multiplicand, long BFP product
MXDB	'ED07'	Long BFP multiplier and multiplicand, extended BFP product

The product of the second operand (the multiplier) and the first operand (the multiplicand) is placed at the first-operand location.

The two BFP operands, if finite numbers, are multiplied, forming an intermediate product. For MDEB, MDEBR, MXDB, and MXDBR, the intermediate product is converted to the longer target format; the result cannot overflow or underflow and is exact. For MDB, MDBR, MEEB, MEEBR, and MXBR, the result is rounded to the operand format according to the current BFP rounding mode. For MEEB and MEEBR, the result, as for all short-format results, replaces the leftmost 32 bits of the target register, and the rightmost 32 bit positions of the target register remain unchanged.

The sign of the product, if the product is numeric, is the exclusive or of the operand signs. This includes the sign of a zero or infinite product.

If one operand is a zero and the other an infinity, an IEEE-invalid-operation exception is recognized.

See Figure 19-23 on page 19-36 for a detailed description of the results of this instruction.

The R_1 field for MXDB, MXDBR, and MXBR, and the R_2 field for MXBR, must designate valid floating-

point-register pairs. Otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Overflow (MDB, MDBR, MEEB, MEEBR, MXBR)
- Underflow (MDB, MDBR, MEEB, MEEBR, MXBR)
- Inexact (MDB, MDBR, MEEB, MEEBR, MXBR)

Program Exceptions:

- Access (fetch, operand 2 of MDEB, MEEB, MDB, and MXDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (MXDB, MXDBR, MXBR)

Programming Note: Interchanging the two operands in a BFP multiplication does not affect the value of the product when the result is numeric. This is not true, however, when both operands are QNaNs, in which case the result is the first operand; or when both operands are SNaNs and the IEEE-invalid-operation mask bit in the FPC register is zero, in which case the result is the QNaN derived from the first operand.

First Operand (a) Is	Results for MULTIPLY ($a \cdot b$) when Second Operand (b) Is							
	$-\infty$	$-Fn$	-0	$+0$	$+Fn$	$+\infty$	QNaN	SNaN
$-\infty$	$T(+\infty)$	$T(+\infty)$	Xi: T(dNaN)	Xi: T(dNaN)	$T(-\infty)$	$T(-\infty)$	T(b)	Xi: T(b*)
$-Fn$	$T(+\infty)$	$R(a \cdot b)$	$T(+0)$	$T(-0)$	$R(a \cdot b)$	$T(-\infty)$	T(b)	Xi: T(b*)
-0	Xi: T(dNaN)	$T(+0)$	$T(+0)$	$T(-0)$	$T(-0)$	Xi: T(dNaN)	T(b)	Xi: T(b*)
$+0$	Xi: T(dNaN)	$T(-0)$	$T(-0)$	$T(+0)$	$T(+0)$	Xi: T(dNaN)	T(b)	Xi: T(b*)
$+Fn$	$T(-\infty)$	$R(a \cdot b)$	$T(-0)$	$T(+0)$	$R(a \cdot b)$	$T(+\infty)$	T(b)	Xi: T(b*)
$+\infty$	$T(-\infty)$	$T(-\infty)$	Xi: T(dNaN)	Xi: T(dNaN)	$T(+\infty)$	$T(+\infty)$	T(b)	Xi: T(b*)
QNaN	T(a)	T(a)	T(a)	T(a)	T(a)	T(a)	T(a)	Xi: T(b*)
SNaN	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)	Xi: T(a*)

Explanation:

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

dNaN Default NaN.

Fn Finite nonzero number (includes both subnormal and normal).

R(v) Rounding and range action is performed on the value v. See Figure 19-13 on page 19-14.

T(x) The value x is placed at the target operand location.

Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 19-23. Results: MULTIPLY

MULTIPLY AND ADD

Mnemonic1 R_1, R_3, R_2 [RRD]

Op Code	R_1	////	R_3	R_2
0	16	20	24	28 31

Mnemonic1 Op Code Operands
 MAEBR 'B30E' Short BFP
 MADBR 'B31E' Long BFP

Mnemonic2 $R_1, R_3, D_2(X_2, B_2)$ [RXF]

Op Code	R_3	X_2	B_2	D_2	R_1	////	Op Code
0	8	12	16	20	32	36	40 47

Mnemonic2 Op Code Operands
 MAEB 'ED0E' Short BFP
 MADB 'ED1E' Long BFP

Mnemonic2 $R_1, R_3, D_2(X_2, B_2)$ [RXF]

Op Code	R_3	X_2	B_2	D_2	R_1	////	Op Code
0	8	12	16	20	32	36	40 47

Mnemonic2 Op Code Operands
 MSEB 'ED0F' Short BFP
 MSDB 'ED1F' Long BFP

The third operand is multiplied by the second operand, and then the first operand is added to or subtracted from the product. The sum or difference is placed at the first-operand location. The MULTIPLY AND ADD and MULTIPLY AND SUBTRACT operations may be summarized as:

$$op_1 = op_3 \cdot op_2 \pm op_1$$

When the operands are finite numbers, the third and second BFP operands are multiplied, forming an intermediate product, and the first operand is then added (or subtracted) algebraically to (or from) the intermediate product, forming an intermediate sum. The intermediate sum, if nonzero, is rounded to the operand format according to the current BFP rounding mode and then placed at the first-operand location. The exponent and fraction of the intermediate product are maintained exactly; rounding and range checking occur only on the intermediate sum.

MULTIPLY AND SUBTRACT

Mnemonic1 R_1, R_3, R_2 [RRD]

Op Code	R_1	////	R_3	R_2
0	16	20	24	28 31

Mnemonic1 Op Code Operands
 MSEBR 'B30F' Short BFP
 MSDBR 'B31F' Long BFP

Third Operand (a) Is	Results, Part 1, for MULTIPLY AND ADD ($a \cdot b + c$) when Second Operand (b) Is							
	$-\infty$	$-Fn$	-0	$+0$	$+Fn$	$+\infty$	QNaN	SNaN
$-\infty$	P($+\infty$)	P($+\infty$)	Xi: T(dNaN)	Xi: T(dNaN)	P($-\infty$)	P($-\infty$)	P(b)	Xi: T(b^*)
$-Fn$	P($+\infty$)	P($a \cdot b$)	P($+0$)	P(-0)	P($a \cdot b$)	P($-\infty$)	P(b)	Xi: T(b^*)
-0	Xi: T(dNaN)	P($+0$)	P($+0$)	P(-0)	P(-0)	Xi: T(dNaN)	P(b)	Xi: T(b^*)
$+0$	Xi: T(dNaN)	P(-0)	P(-0)	P($+0$)	P($+0$)	Xi: T(dNaN)	P(b)	Xi: T(b^*)
$+Fn$	P($-\infty$)	P($a \cdot b$)	P(-0)	P($+0$)	P($a \cdot b$)	P($+\infty$)	P(b)	Xi: T(b^*)
$+\infty$	P($-\infty$)	P($-\infty$)	Xi: T(dNaN)	Xi: T(dNaN)	P($+\infty$)	P($+\infty$)	P(b)	Xi: T(b^*)
QNaN	P(a)	P(a)	P(a)	P(a)	P(a)	P(a)	P(a)	Xi: T(b^*)
SNaN	Xi: T(a^*)	Xi: T(a^*)	Xi: T(a^*)	Xi: T(a^*)	Xi: T(a^*)	Xi: T(a^*)	Xi: T(a^*)	Xi: T(a^*)

Figure 19-24. Results: MULTIPLY AND ADD (Part 1 of 2)

See Figure 19-24 for a detailed description of the results of MULTIPLY AND ADD. The results of MULTIPLY AND SUBTRACT are the same, except that the first operand, if numeric, participates in the operation with its sign bit inverted. When the first operand is a NaN, it participates in the operation with its sign bit unchanged.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Overflow
- Underflow
- Inexact

Program Exceptions:

Value from Part 1 (p) Is	Results, Part 2, for MULTIPLY AND ADD (a•b+c) when First Operand (c) Is							
	-∞	-Fn	-0	+0	+Fn	+∞	QNaN	SNaN
-∞	T(-∞)	T(-∞)	T(-∞)	T(-∞)	T(-∞)	Xi: T(dNaN)	T(c)	Xi: T(c*)
-Fn	T(-∞)	R(p+c)	R(p)	R(p)	R(p+c)	T(+∞)	T(c)	Xi: T(c*)
-0	T(-∞)	R(c)	T(-0)	Rezd	R(c)	T(+∞)	T(c)	Xi: T(c*)
+0	T(-∞)	R(c)	Rezd	T(+0)	R(c)	T(+∞)	T(c)	Xi: T(c*)
+Fn	T(-∞)	R(p+c)	R(p)	R(p)	R(p+c)	T(+∞)	T(c)	Xi: T(c*)
+∞	Xi: T(dNaN)	T(+∞)	T(+∞)	T(+∞)	T(+∞)	T(+∞)	T(c)	Xi: T(c*)
QNaN	T(p)	T(p)	T(p)	T(p)	T(p)	T(p)	T(p)	Xi: T(c*)

Explanation:

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

dNaN Default NaN.

Fn Finite nonzero number (includes both subnormal and normal).

P(x) The value x is passed to Part 2 of this figure.

R(v) Rounding and range action is performed on the value v. See Figure 19-13 on page 19-14.

Rezd Exact zero-difference result. See Figure 19-13 on page 19-14.

T(x) The value x is placed at the target operand location.

Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 19-24. Results: MULTIPLY AND ADD (Part 2 of 2)

- Access (fetch, operand 2 of MAEB, MADB, MSEB, MSDB)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception

Programming Note: MULTIPLY AND ADD and MULTIPLY AND SUBTRACT produce a precise intermediate value, and a single rounding operation is performed after the addition or subtraction. This definition is consistent with the Power architecture, and, in certain applications, can be used to great advantage, especially in algorithms used in math libraries.

SQUARE ROOT

Mnemonic1 R₁,R₂ [RRE]

Op Code	////////	R ₁	R ₂
0	16	24	28 31

Mnemonic1	Op Code	Operands
SQEBR	'B314'	Short BFP
SQDBR	'B315'	Long BFP
SQXBR	'B316'	Extended BFP

Mnemonic2 R₁,D₂(X₂,B₂) [RXE]

Op Code	R ₁	X ₂	B ₂	D ₂	////////	Op Code
0	8	12	16	20	32	40 47

Mnemonic2	Op Code	Operands
SQEB	'ED14'	Short BFP
SQDB	'ED15'	Long BFP

The square root of the second operand is placed at the first-operand location.

The result rounded according to the current BFP rounding mode is placed at the first-operand location.

If the second operand is a positive finite number, the result is the square root of that number with a plus sign. If the operand is a zero of either sign, the result is a zero of the same sign. If the operand is +∞, the result is +∞.

If the second operand is less than zero, an IEEE-invalid-operation exception is recognized.

See Figure 19-17 on page 19-20 for a detailed description of the results of this instruction.

For SQXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Inexact

Program Exceptions:

- Access (fetch, operand 2 of SQEB and SQDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (SQXBR only)

SUBTRACTMnemonic1 R₁,R₂ [RRE]

Op Code	////////	R ₁	R ₂
0	16	24	28 31

Mnemonic1	Op Code	Operands
SEBR	'B30B'	Short BFP
SDBR	'B31B'	Long BFP
SXBR	'B34B'	Extended BFP

Mnemonic2 R₁,D₂(X₂,B₂) [RXE]

Op Code	R ₁	X ₂	B ₂	D ₂	////////	Op Code
0	8	12	16	20	32	40 47

Mnemonic2	Op Code	Operands
SEB	'ED0B'	Short BFP
SDB	'ED1B'	Long BFP

The second operand is subtracted from the first operand, and the difference is placed at the first-operand location.

The execution of SUBTRACT is identical to that of ADD, except that the second operand, if numeric, participates in the operation with its sign bit inverted. When the second operand is a NaN, it participates in the operation with its sign bit unchanged. See Figure 19-12 on page 19-12 for the detailed results of ADD.

For SXBR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Result is a NaN

IEEE Exceptions:

- Invalid operation

- Overflow
- Underflow
- Inexact

Program Exceptions:

- Access (fetch, operand 2 of SEB and SDB only)
- Data with DXC 2, BFP instruction
- Data with DXC for IEEE exception
- Specification (SXBR only)

TEST DATA CLASSMnemonic R₁,D₂(X₂,B₂) [RXE]

Op Code	R ₁	X ₂	B ₂	D ₂	////////	Op Code
0	8	12	16	20	32	40 47

Mnemonic	Op Code	Operands
TCEB	'ED10'	Short BFP
TCDB	'ED11'	Long BFP
TCXB	'ED12'	Extended BFP

The class and sign of the first operand are examined to select one bit from the second-operand address. Condition code 0 or 1 is set according to whether the selected bit is zero or one, respectively.

The second-operand address is not used to address data; instead, the rightmost 12 bits of the address, bits 52-63, are used to specify 12 combinations of BFP data class and sign. Bits 0-51 of the second-operand address are ignored.

As shown in Figure 19-25, BFP operands are divided into six classes: zero, normal number, subnormal number, infinity, quiet NaN, and signaling NaN.

BFP Operand Class	Bit Used when Sign Is	
	+	-
Zero	52	53
Normal number	54	55
Subnormal number	56	57
Infinity	58	59
Quiet NaN	60	61
Signaling NaN	62	63

Figure 19-25. Second-Operand-Address Bits for TEST DATA CLASS

One or more of the second-operand-address bits may be set to one. If the second-operand-address bit corresponding to the class and sign of the first operand

and is one, condition code 1 is set; otherwise, condition code 0 is set.

Operands, including SNaNs and QNaNs, are examined without causing an IEEE exception.

For TCXB, the R_1 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Resulting Condition Code:

- | | |
|---|------------------------------|
| 0 | Selected bit is 0 (no match) |
| 1 | Selected bit is 1 (match) |
| 2 | -- |
| 3 | -- |

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 2, BFP instruction
- Specification (TCXB only)

Programming Notes:

1. TEST DATA CLASS provides a way to test an operand without risk of an exception or setting the IEEE flags.
2. The bits used to specify the combinations of data class and sign are not all the same for BFP and DFP. Specifically, DFP TEST DATA CLASS uses bits 54 and 55 for subnormal number, and bits 56 and 57 for normal number.

Chapter 20. Decimal-Floating-Point Instructions

Decimal-Floating-Point Facility	20-1	Summary of Rounding And Range Actions . .	20-9
DFP Arithmetic	20-2	Result Figures	20-14
Finite Floating-Point Number	20-2	Data-Exception Codes (DXC) and	
Cohort	20-2	Abbreviations	20-14
Quantum	20-2	Instructions	20-15
Preferred Quantum	20-2	ADD	20-19
Scaled Preferred Quantum	20-3	COMPARE	20-21
Delivered Quantum	20-3	COMPARE AND SIGNAL	20-22
Special Quantum-Handling Operations . .	20-3	COMPARE BIASED EXPONENT	20-23
DFP Data Formats	20-3	CONVERT FROM FIXED	20-24
DFP Short Format	20-3	CONVERT FROM LOGICAL	20-25
DFP Long Format	20-3	CONVERT FROM SIGNED PACKED	20-26
DFP Extended Format	20-4	CONVERT FROM UNSIGNED PACKED	20-27
Sign	20-4	CONVERT TO FIXED	20-27
Combination	20-4	CONVERT TO LOGICAL	20-29
Encoded Trailing Significand	20-5	CONVERT TO SIGNED PACKED	20-31
Values of Finite Numbers	20-5	CONVERT TO UNSIGNED PACKED	20-32
Significand	20-5	DIVIDE	20-32
DFP Significant Digits	20-5	EXTRACT BIASED EXPONENT	20-34
Canonical Declets	20-6	EXTRACT SIGNIFICANCE	20-35
DFP Canonical Data	20-6	INSERT BIASED EXPONENT	20-35
Classes of DFP Data	20-6	LOAD AND TEST	20-36
Zeros	20-6	LOAD FP INTEGER	20-37
Subnormal Numbers	20-6	LOAD LENGTHENED	20-40
Normal Numbers	20-7	LOAD ROUNDED	20-41
Infinities	20-7	MULTIPLY	20-43
Signaling and Quiet NaNs	20-7	QUANTIZE	20-45
Canonicalization	20-7	REROUND	20-47
DFP-Format Conversion	20-7	SHIFT SIGNIFICAND LEFT	20-49
DFP Rounding	20-7	SHIFT SIGNIFICAND RIGHT	20-50
DFP Comparison	20-7	SUBTRACT	20-51
DFP Formatting Instructions	20-7	TEST DATA CLASS	20-51
Signed-Packed-Decimal Format	20-8	TEST DATA GROUP	20-52
Unsigned-Packed-Decimal Format	20-8	Densely Packed Decimal (DPD)	20-53
IEEE Exceptions	20-8	Decimal-to-DPD Mapping	20-53
Summary of Preferred Quantum	20-9	DPD-to-Decimal Mapping	20-53

Decimal-Floating-Point Facility

The decimal-floating-point (DFP) facility provides instructions to operate on decimal (radix-10) floating-point data. The facility was designed in cooperation with the IEEE floating-point working group.

Additionally, this facility was designed to support a common library for various systems. This facility in both the z/architecture and Power architectures sup-

ports the same data formats, the same rounding methods, the same functions, and the same exceptions.

DFP has a number of important characteristics:

- It produces results consistent with decimal arithmetic as taught in elementary school.
- It recognizes the quantum of a finite number, maintains quantum information through arithmetic operations, and provides special opera-

tions to extract, compare, and adjust the quantum.

- It provides greater exponent range than hexadecimal-floating-point (HFP) and binary-floating-point (BFP) while maintaining comparable precision.
- It provides greater precision and better space efficiency than provided by the decimal-arithmetic instructions and the packed-decimal format as described in Chapter 8, “Decimal Instructions.”
- It supports IEEE flags, exceptions, and special entities of “infinity” and “Not-a-Number” (NaN).
- It supports eight automatic rounding methods. For DFP computational instructions, an explicit rounding method can be specified in a modifier field in the instruction.
- It facilitates production of DFP results with a precision other than any supported format precision and with the effect of single rounding.
- It provides operations for conversion between fixed-point and DFP formats.
- It provides operations for conversion between packed-decimal and DFP formats.
- It provides three-operand nondestructive arithmetic operations by permitting different target-operand registers from source-operand registers.

The DFP facility uses the same 16 floating-point registers as those used by HFP and BFP. A set of radix-independent floating-point-support instructions can be used to load, store, or change the sign of a DFP operand. These instructions are defined in Chapter 9, “Floating-Point Overview and Support Instructions.”

The DFP facility uses the same floating-point-control (FPC) register as that used by BFP. The bits of the FPC register are often referred to as, for example, FPC1.0, meaning bit 0 of byte 1 of the register. The description of the FPC register, including the DFP-rounding-mode field, and the instructions that operate on the FPC register are provided in Chapter 9, “Floating-Point Overview and Support Instructions.”

The DFP computational model, a high-level overview of the computational steps, is described in the section “IEEE Computational Operations” of Chapter 9,

“Floating-Point Overview and Support Instructions.” Detailed definitions of rounding methods and IEEE exceptions supported by DFP are also described in the section.

DFP Arithmetic

Finite Floating-Point Number

A finite floating-point number has three components: a sign, an exponent, and a significand. The magnitude of a finite number is the product of significand multiplied by the radix raised to the power of the exponent. The number is positive or negative depending on whether the sign bit is zero or one, respectively.

Cohort

Finite decimal-floating-point (DFP) numbers are not necessarily normalized; that is, the leftmost significand digit may be zero. This allows some values to have multiple representations; each with a different combination of the significand and the exponent. The set of different representations for a value of the same sign is called a cohort; and each representation in a cohort is called a cohort member. A plus zero and a minus zero have the same value but are in different cohorts.

Quantum

For a DFP finite number, the magnitude of a value of one in the rightmost digit position of the significand is called the quantum. Each cohort member of a cohort uses a different quantum to represent the same value. Regardless of whether the left-units view or right-units view is taken, the quantum of a cohort member is the same.

Preferred Quantum

For operations that produce a DFP result, when, in the absence of a trap overflow or a trap underflow, the delivered value is a finite number, a value, called the preferred quantum, is defined to select a cohort member to represent the delivered value. The preferred quantum for each of these operations is shown in Figure 20-4 on page 20-9. When the delivered value is exact, the preferred quantum depends on the operation. When the delivered value is inexact, the preferred quantum is the smallest quantum of the cohort members, unless otherwise stated.

Scaled Preferred Quantum

For operations that produce a DFP result, when a trap overflow or a trap underflow occurs, a scaled preferred quantum is used to select a cohort member to represent the delivered value. The scaled preferred quantum is obtained by scaling the preferred quantum using the same scale factor as that used to obtain the delivered value.

Delivered Quantum

Normally, when the delivered value is a finite number, the result is selected from the cohort for the delivered value in the target format. However, in case of a trap overflow or trap underflow for LOAD ROUNDED, the result is selected from a subset of the cohort for the delivered value in the source format. This subset of the cohort consists of cohort members that have the number of DFP significand digits equal to or less than the precision of the target format.

In the absence of a trap overflow or trap underflow, if the delivered value is a finite number, the cohort member with the quantum closest to the preferred quantum is selected.

In case of a trap overflow or trap underflow, the cohort member with the quantum closest to the scaled preferred quantum is selected.

Special Quantum-Handling Operations

Special operations are provided to handle the quantum: COMPARE BIASED EXPONENT can be used to compare two quanta; EXTRACT BIASED EXPONENT can be used to determine the quantum of a finite number; and QUANTIZE can be used to adjust the quantum of a finite number.

Programming Notes:

- 1. Many financial applications and commercial databases use scaled integer representation of decimal data. Calculations are carried out by using integer arithmetic and scaling factors are tracked separately; the scaling factor is often an attribute of a column in the database.

This computational model can be supported by DFP by allowing the integer part of decimal data

to map into the significand with a right-units view and the scaling factor to map into the quantum.

- 2. The term coefficient is used in some other documents to mean an integer value in significand; that is, significand in right-units view.

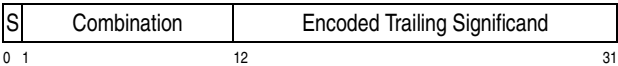
DFP Data Formats

Decimal-floating-point data may be represented in any of three data formats: short, long, or extended.

The contents of each data format represent encoded information. Special codes are assigned to distinguish finite numbers from NaNs and infinities.

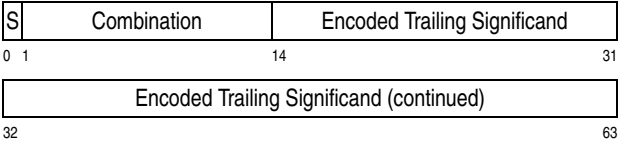
For finite numbers, a biased exponent is used in the format. For each format, a different bias is used for right-units-view (RUV) exponents from that for left-units-view (LUV) exponents. The biased exponents are unsigned numbers and all biases are shown in Figure 20-3 on page 20-6. The biased exponent is encoded with the leftmost digit (LMD) of the significand in the combination field. The remaining digits of the significand are encoded in the encoded trailing-significand field.

DFP Short Format



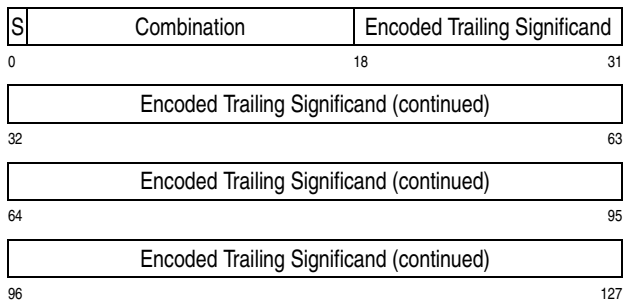
When an operand in the DFP short format is loaded into a floating-point register, it occupies the left half of the register, and the right half remains unchanged.

DFP Long Format



When an operand in the DFP long format is loaded into a floating-point register, it occupies the entire register.

DFP Extended Format



An operand in the DFP extended format occupies a register pair. The leftmost 64 bits occupy the entire lower-numbered register of the pair and the rightmost 64 bits occupy the entire higher-numbered register.

The properties of the three formats are tabulated in Figure 20-3 on page 20-6.

Sign

The sign bit is in bit 0 of each format, and is zero for plus and one for minus.

Combination

For finite numbers, this field contains the biased exponent and the leftmost digit of the significand; for NaNs and infinities, this field contains codes to identify them.

When bits 1-5 of the format are in the range of 00000 - 11101, the operand is a finite number. The two leftmost bits of the biased exponent and the leftmost digit of the significand are encoded in bits 1-5 of the format. Bit 6 through the end of the combination field contain the rest of the biased exponent.

When bits 1-5 of the format field are 11110, the operand is an infinity. All bits in the combination field to the right of bit 5 of the format constitute the reserved field for infinity. A nonzero value in the reserved field is accepted in a source infinity; the reserved field is set to zero in a resultant infinity.

When bits 1-5 of the format are 11111, the operand is a NaN and bit 6, called the SNaN bit, further distinguishes QNaN from SNaN. If bit 6 is zero, then it is QNaN; otherwise, it is SNaN. All bits in the combination field to the right of bit 6 of the format constitute the reserved field for NaN. A nonzero value in the reserved field is accepted in a source NaN; the reserved field is set to zero in a resultant NaN.

Figure 20-1 summarizes the encoding and layout of the combination field. In the figure, the biased exponent of a finite number is the concatenation of two parts: (1) two leftmost bits are derived from bits 1-5 of the format, and (2) the remaining bits in the combination field. For example, if the combination field of the DFP short format contains 101010101 binary, it represents a biased exponent of 10010101 binary and a leftmost significand digit of 5.

Bits 1 2 3 4 5	Bit 6	Type	Biased Exponent	LMD
00000	m	Finite number	00 RBE	0
00001	m	Finite number	00 RBE	1
00010	m	Finite number	00 RBE	2
00011	m	Finite number	00 RBE	3
00100	m	Finite number	00 RBE	4
00101	m	Finite number	00 RBE	5
00110	m	Finite number	00 RBE	6
00111	m	Finite number	00 RBE	7
01000	m	Finite number	01 RBE	0
01001	m	Finite number	01 RBE	1
01010	m	Finite number	01 RBE	2
01011	m	Finite number	01 RBE	3
01100	m	Finite number	01 RBE	4
01101	m	Finite number	01 RBE	5
01110	m	Finite number	01 RBE	6
01111	m	Finite number	01 RBE	7
10000	m	Finite number	10 RBE	0
10001	m	Finite number	10 RBE	1
10010	m	Finite number	10 RBE	2
10011	m	Finite number	10 RBE	3
10100	m	Finite number	10 RBE	4
10101	m	Finite number	10 RBE	5
10110	m	Finite number	10 RBE	6
10111	m	Finite number	10 RBE	7
11000	m	Finite number	00 RBE	8
11001	m	Finite number	00 RBE	9
11010	m	Finite number	01 RBE	8
11011	m	Finite number	01 RBE	9
11100	m	Finite number	10 RBE	8
11101	m	Finite number	10 RBE	9
11110	r	Infinity ¹	--	--
11111	0	QNaN ²	--	--

Figure 20-1. The Combination Field

Bits 1 2 3 4 5	Bit 6	Type	Biased Exponent	LMD
11111	1	SNaN ²	--	--
Explanation: -- Not applicable. Concatenation. 1 All bits in the combination field to the right of bit 5 of the format constitute the reserved field for infinity. 2 All bits in the combination field to the right of bit 6 of the format constitute the reserved field for NaN. LMD Leftmost digit of the significand. m Bit 6 is a part of the remaining biased exponent. RBE Remaining biased exponent. It includes all bits in the combination field to the right of bit 5 of the format. r Bit 6 is a reserved bit for infinity.				

Figure 20-1. The Combination Field (Continued)

Encoded Trailing Significand

This field contains an encoded decimal number, which represents digits in the trailing significand. The trailing significand contains all significand digits, except the leftmost digit. For infinities, nonzero trailing-significand digits are accepted in a source infinity; all trailing-significand digits in a resultant infinity are set to zeros, unless otherwise stated. For NaNs, this field contains diagnostic information called the payload.

The encoded trailing-significand field is a multiple of 10-bit blocks called declets. The number of declets depends on the format. Each declet represents three decimal digits in a 10-bit value. This is described in the section, “Densely Packed Decimal (DPD)” on page 20-53.

Values of Finite Numbers

The values of finite numbers in the various formats are shown in Figure 20-2.

Format	Value	
	Left-Units View	Right-Units View
Short	$\pm 10^{e-95} \times (d_0.d_1d_2\dots d_6)$	$\pm 10^{e-101} \times (d_0d_1d_2\dots d_6)$
Long	$\pm 10^{e-383} \times (d_0.d_1d_2\dots d_{15})$	$\pm 10^{e-398} \times (d_0d_1d_2\dots d_{15})$
Extended	$\pm 10^{e-6143} \times (d_0.d_1d_2\dots d_{33})$	$\pm 10^{e-6176} \times (d_0d_1d_2\dots d_{33})$
Explanation: $d_0.d_1d_2\dots d_{p-1}$ Significand in left-units view. The decimal point is to the immediate right of the leftmost digit and d_i is a decimal digit, where $0 \leq i \leq (p-1)$ and p is the format precision. $d_0d_1d_2\dots d_{p-1}$ Significand in right-units view. The decimal point is to the right of the rightmost digit and d_i is a decimal digit, where $0 \leq i \leq (p-1)$ and p is the format precision. e Biased exponent.		

Figure 20-2. Values of Finite Numbers

Programming Note: The RUV exponent, Q , is related to the LUV exponent, E , as follows: $Q = E - p + 1$, where p is the format precision.

Significand

Hereafter, the term significand is used to mean the following:

1. For finite numbers, the significand contains all trailing significand digits padded on the left with the leftmost digit of significand derived from the combination field.
2. For infinities and NaNs, the significand contains all trailing significand digits padded on the left with a zero digit.

DFP Significant Digits

For a finite number, the DFP significant digits begin with the leftmost nonzero significand digit and end with the rightmost significand digit.

For a finite number, the number of DFP significant digits is the difference of subtracting the number of leading zeros from the format precision. The number of leading zeros is the number of zeros in the significand to the left of the leftmost nonzero digit.

Programming Note: The number of DFP significant digits of the value zero is zero.

Property	Format		
	Short	Long	Extended
Format length (bits)	32	64	128
Combination length (bits)	11	13	17
Encoded Trailing significand length (bits)	20	50	110
Precision (p)	7	16	34
Maximum left-units-view (LUV) exponent (E _{max})	96	384	6144
Minimum left-units-view (LUV) exponent (E _{min})	-95	-383	-6143
Left-units-view (LUV) bias	95	383	6143
Maximum right-units-view (RUV) exponent (Q _{max})	90	369	6111
Minimum right-units-view (RUV) exponent (Q _{min})	-101	-398	-6176
Right-units-view (RUV) bias	101	398	6176
Maximum biased exponent	191	767	12,287
N _{max}	$(10^7 - 1) \times 10^{90}$	$(10^{16} - 1) \times 10^{369}$	$(10^{34} - 1) \times 10^{6111}$
N _{min}	1×10^{-95}	1×10^{-383}	1×10^{-6143}
D _{min}	1×10^{-101}	1×10^{-398}	1×10^{-6176}
Explanation: D _{min} Smallest (in magnitude) subnormal number. N _{max} Largest (in magnitude) normal number. N _{min} Smallest (in magnitude) normal number,			

Figure 20-3. Summary of DFP Formats

Canonical Declets

The trailing significand digits in a DFP data format are encoded by representing three decimal digits with a 10-bit declet. Of the 1024 possible declets, 1000 canonical declets are produced in resultant DFP operands, and 24 noncanonical declets are not produced as DFP results. Both canonical and noncanonical declets are accepted in source DFP operands. The encoding of a canonical declet or noncanonical declet is described in the section, “Densely Packed Decimal (DPD)” on page 20-53.

DFP Canonical Data

A finite number is canonical when all declets are canonical declets.

An infinity is canonical when the reserved field is zero and all digits in the trailing significand are zeros.

A NaN is canonical when the reserved field is zero and all declets are canonical declets.

Programming Note: To ensure compatibility with future extensions and interchangeability between software routines, only DFP canonical data should be used.

Classes of DFP Data

There are six classes of DFP data: zero, subnormal number, normal number, infinity, signaling NaN, and quiet NaN. The zero, subnormal number, and normal number data classes are collectively called finite numbers. The instruction TEST DATA CLASS may be used to determine the class of a DFP operand.

Zeros

Zeros have a zero significand and any representable value in the biased exponent. A +0 is distinct from a -0 and zeros with different biased exponents are distinct, except that comparison treats them as equal.

Subnormal Numbers

Subnormal numbers are numbers which are smaller than N_{min} and larger than zero in magnitude.

Normal Numbers

Normal numbers are numbers whose magnitude is between N_{min} and N_{max} inclusively.

Infinities

An infinity is represented by 11110 binary in bits 1-5 of the data format. Infinities can participate in most arithmetic operations and give a consistent result, usually infinity. In comparisons, $+\infty$ compares greater than any finite number, and $-\infty$ compares less than any finite number. All $+\infty$ are compared equal and all $-\infty$ are compared equal.

Signaling and Quiet NaNs

A NaN (Not-a-Number) is represented by 11111 binary in bits 1-5 of the data format. NaNs are produced in place of a numeric result after an invalid operation when there is no interruption. NaNs may also be used by the program to flag special operands, such as the contents of an uninitialized storage area.

There are two types of NaNs, signaling and quiet. A signaling NaN (SNaN) is distinguished from a quiet NaN (QNaN) by bit 6, the SNaN bit, of the data format: zero for the QNaN and one for the SNaN. A special QNaN, called the default QNaN, is supplied as the nontrap result for an IEEE-invalid-operation exception; it has a plus sign, and the SNaN bit, the reserved field, and the payload are set to zeros.

Normally, source QNaNs are canonicalized to become the resultant QNaN that has the same sign and payload as the source during operations so that they will remain visible at the end. (See the section “Canonicalization” on page 20-7 for details.) An SNaN operand causes an IEEE invalid-operation exception, unless otherwise stated. When the exception is recognized for a SNaN, the nontrap result is the corresponding QNaN, which is produced by setting the SNaN bit in the source SNaN to zero and then the converted QNaN is canonicalized, and the IEEE-invalid-operation flag (FPC 1.0) is set to one.

Canonicalization

For operations that are defined to produce a DFP result, the result placed at the target operand location is canonical, unless otherwise stated.

Programming Notes:

1. The following instructions may produce a noncanonical infinity as the result:

- INSERT BIASED EXPONENT
- LOAD LENGTHENED
- LOAD ROUNDED
- SHIFT SIGNIFICAND LEFT
- SHIFT SIGNIFICAND RIGHT

Infinities produced as a DFP result have the reserved field set to zero and have canonical decrements in the encoded trailing-significand field. However, the infinity produced by the above instructions does not necessarily have zeros in all digits in the trailing significand. Such an infinity is not canonical.

2. Only canonical operands should be used, as it is possible that some libraries or software routines reject noncanonical operands.
3. Except for INSERT BIASED EXPONENT, if only canonical source operands are used, the result is canonical. INSERT BIASED EXPONENT may produce a noncanonical result even if all source operands are canonical. Special precautions should be taken to avoid producing a noncanonical result from INSERT BIASED EXPONENT.

DFP-Format Conversion

DFP format conversion is described in the section “IEEE Same-Radix Format Conversion” of Chapter 9, “Floating-Point Overview and Support Instructions.”

DFP Rounding

DFP rounding is described in the section “IEEE DFP Rounding” of Chapter 9, “Floating-Point Overview and Support Instructions.”

DFP Comparison

DFP comparison is described in the section “IEEE Comparison” of Chapter 9, “Floating-Point Overview and Support Instructions.”

DFP Formatting Instructions

A set of instructions are provided to compose and decompose DFP data. These instructions do not recognize any IEEE exceptions. They are:

- CONVERT FROM SIGNED PACKED

- CONVERT FROM UNSIGNED PACKED
- CONVERT TO SIGNED PACKED
- CONVERT TO UNSIGNED PACKED
- EXTRACT BIASED EXPONENT
- INSERT BIASED EXPONENT
- SHIFT SIGNIFICAND LEFT
- SHIFT SIGNIFICAND RIGHT

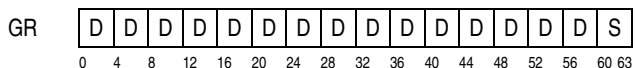
Conversion of data between character strings in an external format and the DFP format may be performed in several steps. At certain steps, for finite numbers, the significand is kept in the packed-decimal format and the biased exponent in the fixed-point format. See Chapter 8, “Decimal Instructions,” for a detailed description of the packed-decimal format.

It takes 17 digits to hold the 16 significand digits and one sign of a long DFP format; it takes 35 digits for an extended DFP format. To facilitate the conversion, the significand is divided into pieces and only one piece contains the sign. Each piece fits into a register or a register pair, and each piece is converted separately. To support this piecemeal conversion, two formats, signed-packed-decimal and unsigned-packed-decimal, are used to hold the pieces.

Signed-Packed-Decimal Format

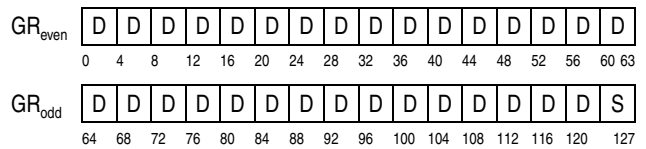
In the signed-packed-decimal format, each byte contains two 4-bit decimal digits (D), except for the rightmost byte, which contains a 4-bit sign (S) to the right of a 4-bit decimal digit.

When a signed-packed-decimal operand resides in a general register (GR), the operand is 64 bits (15 digits and sign), as shown below.



When a signed-packed-decimal operand resides in a general-register pair, the leftmost 64 bits of the oper-

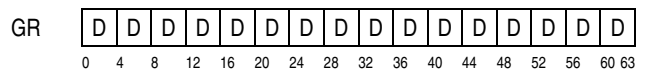
and occupy the even-numbered register of the pair and the rightmost 64 bits occupy the odd-numbered register, as shown below.



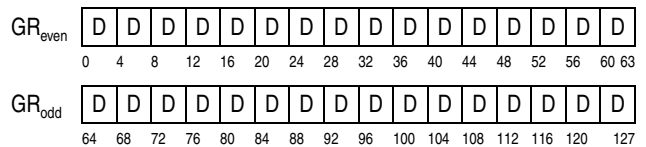
Unsigned-Packed-Decimal Format

In the unsigned-packed-decimal format, each byte contains two 4-bit decimal digits (D) and there is no sign.

When an unsigned-packed-decimal operand resides in a general register, the operand is 64 bits (16 digits), as shown below.



When an unsigned-packed-decimal operand resides in a general-register pair, the leftmost 64 bits of the operand occupy the even-numbered register of the pair and the rightmost 64 bits occupy the odd-numbered register, as shown below.



IEEE Exceptions

The IEEE exceptions for the DFP instructions are described in the section “IEEE Exceptions” on page 9-17. This also includes the quantum exception.

Summary of Preferred Quantum

Figure 20-4 summarizes all preferred quanta.

Operations	Delivered value	Preferred Quantum
ADD	Exact	The smaller quantum of the two source operands†
	Inexact	The smallest quantum of cohort members
CONVERT FROM FIXED	Exact	One†
	Inexact	The smallest quantum of cohort members
CONVERT FROM LOGICAL	Exact	One†
	Inexact	The smallest quantum of cohort members
CONVERT FROM SIGNED PACKED	--	One
CONVERT FROM UNSIGNED PACKED	--	One
DIVIDE	Exact	The quantum of the dividend divided by the quantum of the divisor†
	Inexact	The smallest quantum of cohort members
INSERT BIASED EXPONENT	--	The quantum corresponds to the requested biased exponent
LOAD AND TEST	--	The quantum of the source operand
LOAD FP INTEGER	Exact	The larger value of one and the quantum of the source operand
	Inexact	One
LOAD LENGTHENED	Exact*	The quantum of the source operand
LOAD ROUNDED	Exact	The quantum of the source operand†
	Inexact	The smallest quantum of cohort members
PERFORM FLOATING POINT OPERATION (DPQC=0)‡	Exact	The largest quantum of cohort members†
	Inexact	The smallest quantum of cohort members
PERFORM FLOATING POINT OPERATION (DPQC=1)‡	Exact	One†
	Inexact	The smallest quantum of cohort members
MULTIPLY	Exact	The product of the quanta of the two source operands†
	Inexact	The smallest quantum of cohort members
QUANTIZE	Exact	The requested quantum
	Inexact	The requested quantum
REROUND	Exact	See the instruction description
	Inexact	The quantum that corresponds to the requested significance
SHIFT SIGNIFICAND LEFT	--	The quantum of the source operand
SHIFT SIGNIFICAND RIGHT	--	The quantum of the source operand
SUBTRACT	Exact	The smaller quantum of the two source operands†
	Inexact	The smallest quantum of cohort members
Explanation: * This is true due to a condition to the left of the column. -- For these operations, the concept of exact result or inexact result does not apply. † If the delivered value cannot be represented with the preferred quantum, it is represented with the quantum closest to the preferred quantum. ‡ Applicable only when the operation-type code is 01 hex (PFPO Convert Floating-Point Radix). DPQC DFP preferred quantum control.		

Figure 20-4. Summary of Preferred Quantum

Summary of Rounding And Range Actions

Figure 20-5 on page 20-10 and Figure 20-6 on page 20-11 summarizes rounding and range actions.

Range of v	Case	Nontrap Result (r) when Effective Rounding Method Is							
		RNTE	RNTZ	RNTA	RA	RM	RFS	RP	RZ
$v < -N_{\max}, g < -N_{\max}$	Overflow	$-\infty^1$	$-\infty^1$	$-\infty^1$	$-\infty^1$	$-\infty^1$	-Nmax	-Nmax	-Nmax
$v < -N_{\max}, g = -N_{\max}$	Normal	-Nmax	-Nmax	-Nmax	—	—	-Nmax	-Nmax	-Nmax
$-N_{\max} \leq v \leq -N_{\min}$	Normal	g	g	g	g	g	g	g	g
$-N_{\min} < v \leq -D_{\min}$	Tiny	d*	d*	d*	d*	d*	d*	d	d
$-D_{\min} < v < -D_{\min}/2$	Tiny	-Dmin	-Dmin	-Dmin	-Dmin	-Dmin	-Dmin	-0	-0
$v = -D_{\min}/2$	Tiny	-0	-0	-Dmin	-Dmin	-Dmin	-Dmin	-0	-0
$-D_{\min}/2 < v < 0$	Tiny	-0	-0	-0	-Dmin	-Dmin	-Dmin	-0	-0
$v = 0$	EZD	+0	+0	+0	+0	-0	+0	+0	+0
$0 < v < +D_{\min}/2$	Tiny	+0	+0	+0	+Dmin	+0	+Dmin	+Dmin	+0
$v = +D_{\min}/2$	Tiny	+0	+0	+Dmin	+Dmin	+0	+Dmin	+Dmin	+0
$+D_{\min}/2 < v < +D_{\min}$	Tiny	+Dmin	+Dmin	+Dmin	+Dmin	+0	+Dmin	+Dmin	+0
$+D_{\min} \leq v < +N_{\min}$	Tiny	d*	d*	d*	d*	d	d*	d*	d
$+N_{\min} \leq v \leq +N_{\max}$	Normal	g	g	g	g	g	g	g	g
$+N_{\max} < v, g = +N_{\max}$	Normal	+Nmax	+Nmax	+Nmax	—	+Nmax	+Nmax	—	+Nmax
$+N_{\max} < v, +N_{\max} < g$	Overflow	$+\infty^1$	$+\infty^1$	$+\infty^1$	$+\infty^1$	+Nmax	+Nmax	$+\infty^1$	+Nmax

Explanation:

—	This situation cannot occur.
*	The rounded value, in the extreme case, may be Nmin. In this case, the exceptions are underflow, inexact and incremented.
1	The nontrap result r is considered to have been incremented.
d	The value derived when the precise intermediate value (v) is rounded to the format of the target, including both precision and bounded exponent range. Except as explained in note *, this is a subnormal number.
g	The precision-rounded value. The value derived when the precise intermediate value (v) is rounded to the precision of the target, but assuming an unbounded exponent range.
v	Precise intermediate value. This is the value, before rounding, assuming unbounded precision and an unbounded exponent range. For LOAD ROUNDED, v is the source value.
EZD	Exact zero difference. The case applies only to ADD and SUBTRACT. For all other DFP operations, a zero result is detected by inspection of the source operands without use of the R(v) function.
Dmin	Smallest (in magnitude) representable subnormal number in the target format.
Nmax	Largest (in magnitude) representable finite number in the target format.
Nmin	Smallest (in magnitude) representable normal number in the target format.
RNTA	Round to nearest with ties away from 0.
RNTE	Round to nearest with ties to even.
RNTZ	Round to nearest with ties toward 0.
RA	Round away from 0.
RFS	Round to prepare for shorter precision.
RM	Round toward $-\infty$.
RP	Round toward $+\infty$.
RZ	Round toward 0.

Figure 20-5. Action for R(v): Rounding and Range Actions

Case	Is r inexact (r≠v)	Overflow Mask (FPC 0.2)	Underflow Mask (FPC 0.3)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is QX recognized ³	Quantum Exception Control ² (XqC)	Quantum Mask ² (FPC 0.5)	Is r Incremented (r > v)	Is g inexact (g≠v)	Is g Incremented (g > v)	Results
Overflow	Yes ¹	0	—	0	0	Yes ¹	0	0	—	—	—	T(r), SFo ← 1, SFx ← 1, SFq ← 1
Overflow	Yes ¹	0	—	0	0	Yes ¹	1	—	—	—	—	T(r), SFo ← 1, SFx ← 1
Overflow	Yes ¹	0	—	1	—	Yes ¹	0	0	—	—	—	T(r), SFo ← 1, SFq ← 1
Overflow	Yes ¹	0	—	1	—	Yes ¹	1	—	—	—	—	T(r), SFo ← 1
Overflow	Yes ¹	0	—	0	0	Yes ¹	0	1	—	—	—	T(r), SFo ← 1, SFx ← 1, PIDq(04)
Overflow	Yes ¹	0	—	1	—	Yes ¹	0	1	—	—	—	T(r), SFo ← 1, PIDq(04)
Overflow	Yes ¹	0	—	0	1	No ¹	—	—	No	—	—	T(r), SFo ← 1, PIDx(08)
Overflow	Yes ¹	0	—	0	1	No ¹	—	—	Yes	—	—	T(r), SFo ← 1, PIDy(0C)
Overflow	Yes ¹	1	—	—	—	No ¹	—	—	—	No	No ¹	Tw(g ÷ Ψ), PIDo(20)
Overflow	Yes ¹	1	—	—	—	No ¹	—	—	—	Yes	No	Tw(g ÷ Ψ), PIDox(28)
Overflow	Yes ¹	1	—	—	—	No ¹	—	—	—	Yes	Yes	Tw(g ÷ Ψ), PIDoy(2C)
Normal	No	—	—	—	—	No	—	—	—	—	—	T(r)
Normal	No	—	—	—	—	Yes	0	0	—	—	—	T(r), SFq ← 1
Normal	No	—	—	—	—	Yes	1	—	—	—	—	T(r)
Normal	No	—	—	—	—	Yes	0	1	—	—	—	T(r), PIDq(04)
Normal	Yes	—	—	0	0	Yes ¹	0	0	—	—	—	T(r), SFx ← 1, SFq ← 1
Normal	Yes	—	—	0	0	Yes ¹	1	—	—	—	—	T(r), SFx ← 1

Explanation:

— The results do not depend on this condition or mask bit.

¹ This condition is true by virtue of the state of some condition to the left of this column.

² The quantum exception mask bit and the quantum exception control are defined only when the floating-point extension facility is installed.

³ This column specifies whether a QX is recognized by considering all columns to the left of this column and the existence of the quantum exception condition. The specification in this column does not include the effects of the quantum-exception mask bit and the quantum-exception control.

Ψ Scale factor. For overflow, $\Psi=2^{+\alpha}$. For underflow, $\Psi=2^{-\alpha}$. The unsigned scaling exponent (α) depends on the type of operation and operand format. For all DFP operations except LOAD ROUNDED, α depends on the target format and is 576 for long, and 9216 for extended. For LOAD ROUNDED, α depends on the source format and is 192 for long and 3072 for extended.

g The precision-rounded value. The value derived when the precise intermediate value (v) is rounded to the precision of the target, but assuming an unbounded exponent range.

r Nontrap result as defined in Part 1 of this figure.

v Precise intermediate value. This is the value, before rounding, assuming unbounded precision and unbounded exponent range.

PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. See Figure 20-8 on page 20-15.

PIDq(04) DXC 04 is defined only when the floating-point extension facility is installed.

QX Quantum exception. This exception is defined only when the floating-point extension facility is installed.

SFo IEEE overflow flag, FPC 1.2.

SFq Quantum-exception status flag, FPC 1.5. This flag is defined only when the floating-point extension facility is installed

SFu IEEE underflow flag, FPC 1.3.

SFx IEEE inexact flag, FPC 1.4.

T(x) The value x is placed at the target operand location.

Tw(x) The scaled result x is placed at the target operand location. For all DFP operations except LOAD ROUNDED, the scaled result is in the format and length of the target and rounded to the precision of the target. For LOAD ROUNDED, the scaled result is in the format and length of the source, but rounded to the precision of the target.

Figure 20-6. (Part 1 of 3) Rounding and Range Actions

Case	Is r inexact (r≠v)	Overflow Mask (FPC 0.2)	Underflow Mask (FPC 0.3)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is QX recognized ³	Quantum Exception Control ² (XqC)	Quantum Mask ² (FPC 0.5)	Is r Incremented (r > v)	Is g inexact (g≠v)	Is g Incremented (g > v)	Results
Normal	Yes	—	—	1	—	Yes ¹	0	0	—	—	—	T(r), SFq ← 1
Normal	Yes	—	—	1	—	Yes ¹	1	—	—	—	—	T(r)
Normal	Yes	—	—	0	0	Yes ¹	0	1	—	—	—	T(r), SFx ← 1, PIDq(04)
Normal	Yes	—	—	1	—	Yes ¹	0	1	—	—	—	T(r), PIDq(04)
Normal	Yes	—	—	0	1	No ¹	—	—	No	—	—	T(r), PIDx(08)
Normal	Yes	—	—	0	1	No ¹	—	—	Yes	—	—	T(r), PIDy(0C)
Tiny	No	—	0	—	—	No	—	—	—	—	—	T(r)
Tiny	No	—	0	—	—	Yes	0	0	—	—	—	T(r), SFq ← 1
Tint	No	—	0	—	—	Yes	1	—	—	—	—	T(r)
Tiny	No	—	0	—	—	Yes	0	1	—	—	—	T(r), PIDq(04)
Tiny	No	—	1	—	—	No ¹	—	—	—	No ¹	No ¹	Tw(g ÷ Ψ), PIDu(10)
Tiny	Yes	—	0	0	0	Yes ¹	0	0	—	—	—	T(r), SFu ← 1, SFx ← 1, SFq ← 1
Tiny	Yes	—	0	0	0	Yes ¹	1	—	—	—	—	T(r), SFu ← 1, SFx ← 1
Tiny	Yes	—	0	1	—	Yes ¹	0	0	—	—	—	T(r), SFu ← 1, SFq ← 1
Tiny	Yes	—	0	1	—	Yes ¹	1	—	—	—	—	T(r), SFu ← 1
Tiny	Yes	—	0	0	0	Yes ¹	0	1	—	—	—	T(r), SFu ← 1, SFx ← 1, PIDq(04)
Tiny	Yes	—	0	1	—	Yes ¹	0	1	—	—	—	T(r), SFu ← 1, PIDq(04)

Explanation:

—	The results do not depend on this condition or mask bit.
¹	This condition is true by virtue of the state of some condition to the left of this column.
²	The quantum exception mask bit and the quantum exception control are defined only when the floating-point extension facility is installed.
³	This column specifies whether a QX is recognized by considering all columns to the left of this column and the existence of the quantum exception condition. The specification in this column does not include the effects of the quantum-exception mask bit and the quantum-exception control.
Ψ	Scale factor. For overflow, $\Psi=2^{+\alpha}$. For underflow, $\Psi=2^{-\alpha}$. The unsigned scaling exponent (α) depends on the type of operation and operand format. For all DFP operations except LOAD ROUNDED, α depends on the target format and is 576 for long, and 9216 for extended. For LOAD ROUNDED, α depends on the source format and is 192 for long and 3072 for extended.
g	The precision-rounded value. The value derived when the precise intermediate value (v) is rounded to the precision of the target, but assuming an unbounded exponent range.
r	Nontrap result as defined in Part 1 of this figure.
v	Precise intermediate value. This is the value, before rounding, assuming unbounded precision and unbounded exponent range.
PIDc(h)	Program interruption for data exception, condition c, with DXC of h in hex. See Figure 20-8 on page 20-15.
PIDq(04)	DXC 04 is defined only when the floating-point extension facility is installed.
QX	Quantum exception. This exception is defined only when the floating-point extension facility is installed.
SFo	IEEE overflow flag, FPC 1.2.
SFq	Quantum-exception status flag, FPC 1.5. This flag is defined only when the floating-point extension facility is installed.
SFu	IEEE underflow flag, FPC 1.3.
SFx	IEEE inexact flag, FPC 1.4.
T(x)	The value x is placed at the target operand location.
Tw(x)	The scaled result x is placed at the target operand location. For all DFP operations except LOAD ROUNDED, the scaled result is in the format and length of the target and rounded to the precision of the target. For LOAD ROUNDED, the scaled result is in the format and length of the source, but rounded to the precision of the target.

Figure 20-6. (Part 2 of 3) Rounding and Range Actions

Case	Is r inexact (r≠v)	Overflow Mask (FPC 0.2)	Underflow Mask (FPC 0.3)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is QX recognized ³	Quantum Exception Control ² (XqC)	Quantum Mask ² (FPC 0.5)	Is r Incremented (r > v)	Is g inexact (g≠v)	Is g Incremented (g > v)	Results
Tiny	Yes	—	0	0	1	No ¹	—	—	No	—	—	T(r), SFu ← 1, PIDx(08)
Tiny	Yes	—	0	0	1	No ¹	—	—	Yes	—	—	T(r), SFu ← 1, PIDy(0C)
Tiny	Yes	—	1	—	—	No ¹	—	—	—	No	No ¹	Tw(g ÷ Ψ), PIDu(10)
Tiny	Yes	—	1	—	—	No ¹	—	—	—	Yes	No	Tw(g ÷ Ψ), PIDux(18)
Tiny	Yes	—	1	—	—	No ¹	—	—	—	Yes	Yes	Tw(g ÷ Ψ), PIDuy(1C)
Explanation: — The results do not depend on this condition or mask bit. ¹ This condition is true by virtue of the state of some condition to the left of this column. ² The quantum exception mask bit and the quantum exception control are defined only when the floating-point extension facility is installed. ³ This column specifies whether a QX is recognized by considering all columns to the left of this column and the existence of the quantum exception condition. The specification in this column does not include the effects of the quantum-exception mask bit and the quantum-exception control. Ψ Scale factor. For overflow, $\Psi=2^{+\alpha}$. For underflow, $\Psi=2^{-\alpha}$. The unsigned scaling exponent (α) depends on the type of operation and operand format. For all DFP operations except LOAD ROUNDED, α depends on the target format and is 576 for long, and 9216 for extended. For LOAD ROUNDED, α depends on the source format and is 192 for long and 3072 for extended. g The precision-rounded value. The value derived when the precise intermediate value (v) is rounded to the precision of the target, but assuming an unbounded exponent range. r Nontrap result as defined in Part 1 of this figure. v Precise intermediate value. This is the value, before rounding, assuming unbounded precision and unbounded exponent range. PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. See Figure 20-8 on page 20-15. PIDq(04) DXC 04 is defined only when the floating-point extension facility is installed. QX Quantum exception. This exception is defined only when the floating-point extension facility is installed. SFo IEEE overflow flag, FPC 1.2. SFq Quantum-exception status flag, FPC 1.5. This flag is defined only when the floating-point extension facility is installed. SFu IEEE underflow flag, FPC 1.3. SFx IEEE inexact flag, FPC 1.4. T(x) The value x is placed at the target operand location. Tw(x) The scaled result x is placed at the target operand location. For all DFP operations except LOAD ROUNDED, the scaled result is in the format and length of the target and rounded to the precision of the target. For LOAD ROUNDED, the scaled result is in the format and length of the source, but rounded to the precision of the target.												

Figure 20-6. (Part 3 of 3) Rounding and Range Actions

Result Figures

Concise descriptions of the results produced by many of the DFP instructions are made by means of figures which contain columns and rows representing all possible combinations of DFP data class for the source operands of an instruction. The information shown at the intersection of a row and a column is one or more symbols representing the result or results produced for that particular combination of source-operand data classes. Explanations of the symbols used are contained in each figure. In many cases, the explanation of a particular result is in the form of a cross-reference to another figure. In many cases, the information shown at the intersection consists of several symbols separated by commas. All such results are produced unless one of the results is a program interruption. In the case of a program interruption, the operation is suppressed or completed as shown in Figure 20-8 on page 20-15.

Data-Exception Codes (DXC) and Abbreviations

Figure 20-7 shows IEEE exceptions and flag abbreviations that are used in the result figures, and it explains the symbols “Xi:” and “Xz:” that are used in the figures. Bits 0-4 (i,z,o,u,x) of the eight-bit data-exception code (DXC) in byte 2 of the FPC register are trap flags and correspond to the same bits in bytes 0 and 1 of the register (IEEE masks and IEEE flags). The trap flag for an exception, instead of the IEEE flag, is set to one when an interruption for the exception is enabled by the corresponding IEEE mask bit. Bit 5 (y) of byte 2 is used in conjunction with bit 4, inexact (x), to indicate that the result has been incremented in magnitude.

When the floating-point extension facility is installed, bit 5 of DXC also has a second meaning. When DXC bits 0-4 are zero, DXC bit 5 is the quantum-exception trap flag and corresponds to the quantum-exception mask IMq and its status flag SFq.

Figure 20-8 on page 20-15 shows the various DXCs that can be indicated, the associated instruction endings, and abbreviations that are used for the DXCs in the result figures. (The abbreviation “PID” stands for “program interruption for a data exception.”)

Exception		FPC IEEE Mask bit	IEEE Flag	
Name	Abbr.		FPC Bit	Abbr.
IEEE invalid operation	Xi ¹	0.0	1.0	SFi
IEEE division by zero	Xz ²	0.1	1.1	SFz
IEEE overflow	Xo	0.2	1.2	SFo
IEEE underflow	Xu	0.3	1.3	SFu
IEEE inexact	Xx	0.4	1.4	SFx
Quantum-exception	Xq	0.5	1.5	SFq

Explanation:

¹ The symbol “Xi:” followed by a list of results in a figure indicates that, when FPC 0.0 is zero, then instruction execution is completed by setting SFi (FPC 1.0) to one and producing the indicated results; and when FPC 0.0 is one, then instruction execution is suppressed, the data exception code (DXC) is set to 80 hex, and a program interruption for a data exception occurs.

² The symbol “Xz:” followed by a list of results in a figure indicates that, when FPC 0.1 is zero, then instruction execution is completed by setting SFz (FPC 1.1) to one and producing the indicated results; and when FPC 0.1 is one, then instruction execution is suppressed, the data exception code (DXC) is set to 40 hex, and a program interruption for a data exception occurs.

Figure 20-7. IEEE Exception and Flag Abbreviations

Abbr.	DXC (hex)	Data-Exception-Code Name	Instruction Ending
PIDq	04	Quantum exception	Complete
PIDx	08	IEEE inexact and truncated	Complete
PIDy	0C	IEEE inexact and incremented	Complete
PIDu	10	IEEE underflow, exact	Complete, wrap exponent
PIDux	18	IEEE underflow, inexact and truncated	Complete, wrap exponent
PIDuy	1C	IEEE underflow, inexact and incremented	Complete, wrap exponent
PIDo	20	IEEE overflow, exact	Complete, wrap exponent
PIDox	28	IEEE overflow, inexact and truncated	Complete, wrap exponent
PIDoy	2C	IEEE overflow, inexact and incremented	Complete, wrap exponent
PIDz	40	IEEE division by zero	Suppress
PIDi	80	IEEE invalid operation	Suppress

Figure 20-8. IEEE Data-Exception Codes (DXC) and Abbreviations

Instructions

The Decimal-Floating-Point (DFP) instructions and their mnemonics and operation codes are listed in Figure 20-9 on page 20-16.

The figure indicates, in the column labeled “Characteristics,” the instruction format, when the condition code is set, and the exceptional conditions in operand designations, data, or results that cause a program interruption.

All DFP instructions are subject to the AFP-register-control bit, bit 45 of control register 0. For the DFP instructions to be executed successfully, the AFP-register-control bit must be one; otherwise, a DFP-instruction data exception, DXC 3, is recognized.

Mnemonics for the DFP instructions are distinguished from corresponding HFP or BFP instructions by a T in the mnemonic. Mnemonics for the DFP instructions have an R as the last letter or the letter

next to the last letter when the instruction is in the RRE or RRF format. Mnemonics for the DFP instructions have an A as the last letter when the instruction is an alternate instruction, which uses additional modifier fields not available to the original instruction. Certain letters are used for DFP instructions to represent operand-format length, as follows:

- G Sixty-four-bit fixed point
- F Thirty-two-bit fixed point
- E Short
- D Long
- X Extended

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For a register-to-register operation using COMPARE (long), for example, CDTR is the mnemonic and R₁,R₂ the operand designation.

Programming Note: The floating-point extension facility includes the following DFP features are added:

- The following new DFP instructions:
 - CONVERT FROM FIXED (CXFTR, CDFTR)
 - CONVERT FROM LOGICAL (CXLGTR, CDLGTR, CXLFTR, CDLFTR)
 - CONVERT TO FIXED (CFXTR, CFDTR)
 - CONVERT TO LOGICAL (CLGXTR, CLGDTR, CLFXTR, CLFDTR)
- All reserved values in the effective rounding method field and a quantum-exception control (XqC) are assigned for LOAD FP INTEGER, LOAD ROUNDED, QUANTIZE, and REROUND.
- All reserved values in the effective rounding method field are assigned for CONVERT TO FIXED.
- An IEEE-inexact-exception control (XxC) is added to CONVERT TO FIXED and LOAD ROUNDED.
- An effective rounding method field and a quantum-exception control (XqC) are added to ADD, DIVIDE, MULTIPLY, and SUBTRACT.

An effective rounding method field, an IEEE-inexact-exception control (XxC), and a quantum-exception control (XqC) are added to CONVERT FROM FIXED.

Name	Mne- monic	Characteristics						Op Code	Page
ADD (extended DFP)	AXTR	RRF-a C TF	SP	Dt Xi	Xo Xu Xx			B3DA	20-19
ADD (extended DFP)	AXTRA	RRF-a C F	SP	Dt Xi	Xo Xu Xx Xq			B3DA	20-19
ADD (long DFP)	ADTR	RRF-a C TF		Dt Xi	Xo Xu Xx			B3D2	20-19
ADD (long DFP)	ADTRA	RRF-a C F		Dt Xi	Xo Xu Xx Xq			B3D2	20-19
COMPARE (extended DFP)	CXTR	RRE C TF	SP	Dt Xi				B3EC	20-21
COMPARE (long DFP)	CDTR	RRE C TF		Dt Xi				B3E4	20-21
COMPARE AND SIGNAL (extended DFP)	KXTR	RRE C TF	SP	Dt Xi				B3E8	20-22
COMPARE AND SIGNAL (long DFP)	KDTR	RRE C TF		Dt Xi				B3E0	20-22
COMPARE BIASED EXPONENT (extended DFP)	CEXTR	RRE C TF	SP	Dt				B3FC	20-23
COMPARE BIASED EXPONENT (long DFP)	CEDTR	RRE C TF		Dt				B3F4	20-23
CONVERT FROM FIXED (32 to extended DFP)	CXFTR	RRE F	SP	Dt				B959	20-24
CONVERT FROM FIXED (32 to long DFP)	CDFTR	RRE F		Dt				B951	20-24
CONVERT FROM FIXED (64 to extended DFP)	CXGTR	RRE TF	SP	Dt				B3F9	20-24
CONVERT FROM FIXED (64 to extended DFP)	CXGTRA	RRF-e F	SP	Dt				B3F9	20-24
CONVERT FROM FIXED (64 to long DFP)	CDGTR	RRE TF		Dt	Xx			B3F1	20-24
CONVERT FROM FIXED (64 to long DFP)	CDGTRA	RRF-e F		Dt	Xx Xq			B3F1	20-24
CONVERT FROM LOGICAL (32 to extended DFP)	CXLFTR	RRF-e F	SP	Dt				B95B	20-25
CONVERT FROM LOGICAL (32 to long DFP)	CDLFTR	RRF-e F		Dt				B953	20-25
CONVERT FROM LOGICAL (64 to extended DFP)	CXLGTR	RRF-e F	SP	Dt				B95A	20-25
CONVERT FROM LOGICAL (64 to long DFP)	CDLGTR	RRF-e F		Dt	Xx Xq			B952	20-25
CONVERT FROM SIGNED PACKED (128 to extended DFP)	CXSTR	RRE TF	SP	Dt Dd				B3FB	20-26
CONVERT FROM SIGNED PACKED (64 to long DFP)	CDSTR	RRE TF		Dt Dd				B3F3	20-26
CONVERT FROM UNSIGNED PACKED (128 to ext. DFP)	CXUTR	RRE TF	SP	Dt Dd				B3FA	20-27
CONVERT FROM UNSIGNED PACKED (64 to long DFP)	CDUTR	RRE TF		Dt Dd				B3F2	20-27
CONVERT TO FIXED (extended DFP to 32)	CFXTR	RRF-e C F	SP	Dt Xi	Xx			B949	20-27
CONVERT TO FIXED (extended DFP to 64)	CGXTR	RRF-e C TF	SP	Dt Xi	Xx			B3E9	20-27
CONVERT TO FIXED (extended DFP to 64)	CGXTRA	RRF-e C F	SP	Dt Xi	Xx			B3E9	20-27
CONVERT TO FIXED (long DFP to 32)	CFDTR	RRF-e C F		Dt Xi	Xx			B941	20-27
CONVERT TO FIXED (long DFP to 64)	CGDTR	RRF-e C TF		Dt Xi	Xx			B3E1	20-27
CONVERT TO FIXED (long DFP to 64)	CGDTRA	RRF-e C F		Dt Xi	Xx			B3E1	20-27
CONVERT TO LOGICAL (extended DFP to 32)	CLFXTR	RRF-e C F	SP	Dt Xi	Xx			B94B	20-29
CONVERT TO LOGICAL (extended DFP to 64)	CLGXTR	RRF-e C F	SP	Dt Xi	Xx			B94A	20-29
CONVERT TO LOGICAL (long DFP to 32)	CLFDTR	RRF-e C F		Dt Xi	Xx			B943	20-29
CONVERT TO LOGICAL (long DFP to 64)	CLGDTR	RRF-e C F		Dt Xi	Xx			B942	20-29
CONVERT TO SIGNED PACKED (extended DFP to 128)	CSXTR	RRF-d TF	SP	Dt				B3EB	20-31
CONVERT TO SIGNED PACKED (long DFP to 64)	CSDTR	RRF-d TF		Dt				B3E3	20-31
CONVERT TO UNSIGNED PACKED (extended DFP to 128)	CUXTR	RRE TF	SP	Dt				B3EA	20-32
CONVERT TO UNSIGNED PACKED (long DFP to 64)	CUDTR	RRE TF		Dt				B3E2	20-32
DIVIDE (extended DFP)	DXTR	RRF-a TF	SP	Dt Xi Xz Xo Xu Xx				B3D9	20-32
DIVIDE (extended DFP)	DXTRA	RRF-a F	SP	Dt Xi Xz Xo Xu Xx Xq				B3D9	20-32
DIVIDE (long DFP)	DDTR	RRF-a TF		Dt Xi Xz Xo Xu Xx				B3D1	20-32
DIVIDE (long DFP)	DDTRA	RRF-a F		Dt Xi Xz Xo Xu Xx Xq				B3D1	20-32
EXTRACT BIASED EXPONENT (extended DFP to 64)	EEXTR	RRE TF	SP	Dt				B3ED	20-34
EXTRACT BIASED EXPONENT (long DFP to 64)	EEDTR	RRE TF		Dt				B3E5	20-34
EXTRACT SIGNIFICANCE (extended DFP)	ESXTR	RRE TF	SP	Dt				B3EF	20-35
EXTRACT SIGNIFICANCE (long DFP)	ESDTR	RRE TF		Dt				B3E7	20-35

Figure 20-9. Summary of Decimal Instructions (Part 1 of 3)

Name	Mnemonic	Characteristics						Op Code	Page
INSERT BIASED EXPONENT (64 to extended DFP)	IEXTR	RRF-b	TF	SP	Dt			B3FE	20-35
INSERT BIASED EXPONENT (64 to long DFP)	IEDTR	RRF-b	TF		Dt			B3F6	20-35
LOAD AND TEST (extended DFP)	LTXTR	RRE C	TF	SP	Dt Xi			B3DE	20-36
LOAD AND TEST (long DFP)	LTDTR	RRE C	TF		Dt Xi			B3D6	20-36
LOAD FP INTEGER (extended DFP)	FIXTR	RRF-e	TF	SP	Dt Xi	Xx Xq		B3DF	20-37
LOAD FP INTEGER (long DFP)	FIDTR	RRF-e	TF		Dt Xi	Xx Xq		B3D7	20-37
LOAD LENGTHENED (long to extended DFP)	LXDTR	RRF-d	TF	SP	Dt Xi			B3DC	20-40
LOAD LENGTHENED (short to long DFP)	LDETR	RRF-d	TF		Dt Xi			B3D4	20-40
LOAD ROUNDED (extended to long DFP)	LDXTR	RRF-e	TF	SP	Dt Xi	Xo Xu Xx Xq		B3DD	20-41
LOAD ROUNDED (long to short DFP)	LEDTR	RRF-e	TF		Dt Xi	Xo Xu Xx Xq		B3D5	20-41
MULTIPLY (extended DFP)	MXTR	RRF-a	TF	SP	Dt Xi	Xo Xu Xx		B3D8	20-43
MULTIPLY (extended DFP)	MXTRA	RRF-a	F	SP	Dt Xi	Xo Xu Xx Xq		B3D8	20-43
MULTIPLY (long DFP)	MDTR	RRF-a	TF		Dt Xi	Xo Xu Xx		B3D0	20-43
MULTIPLY (long DFP)	MDTRA	RRF-a	F		Dt Xi	Xo Xu Xx Xq		B3D0	20-43
QUANTIZE (extended DFP)	QAXTR	RRF-b	TF	SP	Dt Xi	Xx Xq		B3FD	20-45
QUANTIZE (long DFP)	QADTR	RRF-b	TF		Dt Xi	Xx Xq		B3F5	20-45
REROUND (extended DFP)	RRXTR	RRF-b	TF	SP	Dt Xi	Xx Xq		B3FF	20-47
REROUND (long DFP)	RRDTR	RRF-b	TF		Dt Xi	Xx Xq		B3F7	20-47
SHIFT SIGNIFICAND LEFT (extended DFP)	SLXT	RXF	TF	SP	Dt			ED48	20-49
SHIFT SIGNIFICAND LEFT (long DFP)	SLDT	RXF	TF		Dt			ED40	20-49
SHIFT SIGNIFICAND RIGHT (extended DFP)	SRXT	RXF	TF	SP	Dt			ED49	20-50
SHIFT SIGNIFICAND RIGHT (long DFP)	SRDT	RXF	TF		Dt			ED41	20-50
SUBTRACT (extended DFP)	SXTR	RRF-a C	TF	SP	Dt Xi	Xo Xu Xx		B3DB	20-51
SUBTRACT (extended DFP)	SXTRA	RRF-a C	F	SP	Dt Xi	Xo Xu Xx Xq		B3DB	20-51
SUBTRACT (long DFP)	SDTR	RRF-a C	TF		Dt Xi	Xo Xu Xx		B3D3	20-51
SUBTRACT (long DFP)	SDTRA	RRF-a C	F		Dt Xi	Xo Xu Xx Xq		B3D3	20-51
TEST DATA CLASS (extended DFP)	TDCXT	RXE C	TF	SP	Dt			ED58	20-51
TEST DATA CLASS (long DFP)	TDCDT	RXE C	TF		Dt			ED54	20-51
TEST DATA CLASS (short DFP)	TDCET	RXE C	TF		Dt			ED50	20-51
TEST DATA GROUP (extended DFP)	TDGXT	RXE C	TF	SP	Dt			ED59	20-52
TEST DATA GROUP (long DFP)	TDGDT	RXE C	TF		Dt			ED55	20-52
TEST DATA GROUP (short DFP)	TDGET	RXE C	TF		Dt			ED51	20-52

Figure 20-9. Summary of Decimal Instructions (Part 2 of 3)

Name		Mne- monic	Characteristics	Op Code	Page
Explanation:					
A	Access exceptions for logical addresses.				
B ₂	B ₂ field designates an access register in the access-register mode.				
C	Condition code is set.				
Dd	Decimal-operand data exception.				
Dt	DFP-instruction data exception.				
F	Floating-point extension facility.				
RRE	RRE instruction format.				
RRF	RRF instruction format.				
RXE	RXE instruction format.				
RXF	RXF instruction format.				
S	S instruction format.				
SP	Specification exception.				
TF	Decimal-Floating-Point facility.				
Xi	IEEE invalid-operation exception.				
Xo	IEEE overflow exception.				
Xq	Quantum exception; if the floating-point extension facility is installed.				
Xu	IEEE underflow exception.				
Xx	IEEE inexact exception.				
Xz	IEEE division-by-zero exception.				

Figure 20-9. Summary of Decimal Instructions (Part 3 of 3)

ADD

Mnemonic1 R_1, R_2, R_3 [RRF-a]

Op Code	R_3	////	R_1	R_2
0	16	20	24	28 31

Mnemonic1	Op Code	Operands
ADTR	'B3D2'	Long DFP
AXTR	'B3DA'	Extended DFP

Mnemonic2 R_1, R_2, R_3, M_4 [RRF-a]

Op Code	R_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic2	Op Code	Operands
ADTRA	'B3D2'	Long DFP
AXTRA	'B3DA'	Extended DFP

The third operand is added to the second operand, and the sum is placed at the first-operand location.

If both operands are finite numbers, they are added algebraically, forming an intermediate sum. The intermediate sum, if nonzero, is rounded to the operand format and the rounded value is then placed at the result location.

When the floating-point extension facility is installed, the intermediate sum is rounded by rounding as specified by the modifier in the M_4 field:

M_4 Effective Rounding Method

- | | |
|----|--|
| 0 | According to the current DFP rounding mode |
| 1 | Round to nearest with ties away from 0 |
| 2 | According to the current DFP rounding mode |
| 3 | Round to prepare for shorter precision |
| 4 | Round to nearest with ties to even |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |
| 7 | Round toward $-\infty$ |
| 8 | Round to nearest with ties to even |
| 9 | Round toward 0 |
| 10 | Round toward $+\infty$ |
| 11 | Round toward $-\infty$ |
| 12 | Round to nearest with ties away from 0 |
| 13 | Round to nearest with ties toward 0 |
| 14 | Round away from 0 |
| 15 | Round to prepare for shorter precision |

When the modifier field is zero or two, rounding is controlled by the current DFP rounding mode speci-

fied in the FPC register. When the field is 1, or 3-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception control is zero and recognition of the exception is not suppressed.

When the floating-point extension facility is not installed, no quantum exception is recognized. If bits 20-23 of the instruction contain a value of zero, the intermediate sum is rounded according to the current DFP rounding mode; if bits 20-23 of the instruction contain a nonzero value, it is unpredictable which rounding method is performed.

The sign of the sum is determined by the rules of algebra. This also applies to a result of zero, as follows:

- If the result of rounding a nonzero intermediate sum is zero, the sign of the zero result is the sign of the intermediate sum.
- If the sum of two operands with opposite signs is exactly zero, the sign of the result is plus in all rounding methods except round toward $-\infty$, in which method the sign is minus.
- The sign of the sum x plus x is the sign of x , even when x is zero.

If one operand is an infinity and the other is a finite number, the result is an infinity with the sign of the source infinity. If both operands are infinities of the same sign, the result is an infinity with the same sign. If the two operands are infinities of opposite signs, an IEEE-invalid-operation exception is recognized.

When the delivered value is exact, the preferred quantum is the smaller quantum of the two source operands; when the delivered value is inexact, the preferred quantum is the smallest quantum.

The result placed at the first-operand location is canonical.

See Figure 20-11 on page 20-21 for a detailed description of the results of this instruction. (Figure 20-10 is referred to by Figure 20-11.)

For AXTR and AXTRA, the R fields designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Result is a NaN

IEEE Exceptions:

- Invalid operation
- Overflow
- Underflow
- Inexact
- Quantum (if the floating-point extension facility is installed)

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (AXTR and AXTRA only)

Value of Result (r)	Condition code
$r = 0$	cc0
$r < 0$	cc1
$r > 0$	cc2
Explanation:	
ccn Condition code is set to n.	

Figure 20-10. Condition Code for Resultant Sum

Programming Notes:

1. Interchanging the two operands in a DFP addition does not affect the value of the sum when the result is numeric. This is not true, however, when both operands are QNaNs; or when both operands are SNaNs and the IEEE invalid-operation mask bit in the FPC register is zero, in these cases, the result is the canonical QNaN derived from the second operand.
2. For ADD or SUBTRACT, when the delivered value is inexact, the preferred quantum is the smallest quantum which is the same quantum as that closest to the smaller quantum of the two source operands.

Second Operand (b) is	Results for ADD (b+c) when Third Operand (c) is									
	$-\infty$	-Nn	-Dn	-0	+0	+Dn	+Nn	$+\infty$	QNaN	SNaN
$-\infty$	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	T($-\infty$), cc1	Xi: T(dNaN), cc3	T(c), cc3	Xi: T(c*), cc3
-Nn	T($-\infty$), cc1	R(b+c), cc1	R(b+c), cc1	T(b), cc1	T(b), cc1	R(b+c), cc1	R(b+c), ccrs	T($+\infty$), cc2	T(c), cc3	Xi: T(c*), cc3
-Dn	T($-\infty$), cc1	R(b+c), cc1	R(b+c), cc1	R(b), cc1	R(b), cc1	R(b+c), ccrs	R(b+c), cc2	T($+\infty$), cc2	T(c), cc3	Xi: T(c*), cc3
-0	T($-\infty$), cc1	T(c), cc1	R(c), cc1	T(-0), cc0	Rezd, cc0	R(c), cc2	T(c), cc2	T($+\infty$), cc2	T(c), cc3	Xi: T(c*), cc3
+0	T($-\infty$), cc1	T(c), cc1	R(c), cc1	Rezd, cc0	T(+0), cc0	R(c), cc2	T(c), cc2	T($+\infty$), cc2	T(c), cc3	Xi: T(c*), cc3
+Dn	T($-\infty$), cc1	R(b+c), cc1	R(b+c), ccrs	R(b), cc2	R(b), cc2	R(b+c), cc2	R(b+c), cc2	T($+\infty$), cc2	T(c), cc3	Xi: T(c*), cc3
+Nn	T($-\infty$), cc1	R(b+c), ccrs	R(b+c), cc2	T(b), cc2	T(b), cc2	R(b+c), cc2	R(b+c), cc2	T($+\infty$), cc2	T(c), cc3	Xi: T(c*), cc3
$+\infty$	Xi: T(dNaN), cc3	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T($+\infty$), cc2	T(c), cc3	Xi: T(c*), cc3
QNaN	T(b), cc3	T(b), cc3	T(b), cc3	T(b), cc3	T(b), cc3	T(b), cc3	T(b), cc3	T(b), cc3	T(b), cc3	Xi: T(c*), cc3
SNaN	Xi: T(b*), cc3	Xi: T(b*), cc3	Xi: T(b*), cc3	Xi: T(b*), cc3	Xi: T(b*), cc3	Xi: T(b*), cc3	Xi: T(b*), cc3	Xi: T(b*), cc3	Xi: T(b*), cc3	Xi: T(b*), cc3

Explanation:

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

ccn Condition code is set to n.

ccrs Condition code is set according to the resultant sum. See Figure 20-10 on page 20-20.

dNaN Default NaN.

Nn Normal number.

R(v) Rounding and range action is performed on the value v. See Figure 20-5 on page 20-10. The result is canonical.

Rezd Exact zero-difference result. See Figure 20-5 on page 20-10.

Dn Subnormal number.

T(x) The canonical result x is placed at the target operand location.

Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 20-11. Results: ADD

COMPARE

Mnemonic R_1, R_2 [RRE]

Op Code	Op Code	R ₁	R ₂
0	16	24	28 31

Mnemonic	Op Code	Operands
CDTR	'B3E4'	Long DFP
CXTR	'B3EC'	Extended DFP

The first operand is compared with the second operand, and the condition code is set to indicate the result.

If both operands are finite numbers, the comparison is algebraic and follows the procedure for DFP subtraction, except that the difference is discarded after setting the condition code, and both operands remain unchanged. If the difference is exactly zero with either sign, the operands are equal; this includes zero operands (so +0 equals -0). If a nonzero difference is positive or negative, the first operand is high or low, respectively.

$+\infty$ compares greater than any finite number, and all finite numbers compare greater than $-\infty$. Two infinity operands of like sign compare equal.

except that QNaN operands cause an IEEE-invalid-operation exception to be recognized. Thus QNaN operands are treated as if they were SNaNs.

See Figure 20-14 on page 20-23 for a detailed description of the results of this instruction.

For KXTR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

0 Operands equal

- 1 First operand low
- 2 First operand high
- 3 Operands unordered

IEEE Exceptions:

- Invalid operation

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (KXTR only)

First Operand (a) is	Results for COMPARE AND SIGNAL (a:b) when Second Operand (b) is						
	$-\infty$	-Fn	-0	+0	+Fn	$+\infty$	NaN
$-\infty$	cc0	cc1	cc1	cc1	cc1	cc1	Xi: cc3
-Fn	cc2	C(a:b)	cc1	cc1	cc1	cc1	Xi: cc3
-0	cc2	cc2	cc0	cc0	cc1	cc1	Xi: cc3
+0	cc2	cc2	cc0	cc0	cc1	cc1	Xi: cc3
+Fn	cc2	cc2	cc2	cc2	C(a:b)	cc1	Xi: cc3
$+\infty$	cc2	cc2	cc2	cc2	cc2	cc0	Xi: cc3
NaN	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3	Xi: cc3

Explanation:

ccn Condition code is set to n.
C(a:b) Basic compare results. See Figure 20-13.
Fn Nonzero finite number (includes both subnormal and normal).
Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 20-14. Results: COMPARE AND SIGNAL

COMPARE BIASED EXPONENT

Mnemonic	R_1, R_2	[RRE]
Op Code	Op Code	R_1 R_2
0	16	24 28 31

Mnemonic	Op Code	Operands
CEDTR	'B3F4'	Long DFP
CEXTR	'B3FC'	Extended DFP

The biased exponent of the first operand is compared with the biased exponent of the second operand, and the condition code is set to indicate the result.

This operation is performed for any second operand, including an infinity, QNaN, or SNaN, without causing an IEEE exception.

See Figure 20-15 on page 20-24 for a detailed description of the results of this instruction.

For CEXTR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Biased exponents equal
- 1 First-operand biased exponent low
- 2 First-operand biased exponent high
- 3 Unordered

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (CEXTR only)

First operand (a) is	Result for COMPARE BIASED EXPONENT when second operand (b) is			
	F	∞	QNaN	SNaN
F	$C(e_a:e_b)$	cc3	cc3	cc3
∞	cc3	cc0	cc3	cc3
QNaN	cc3	cc3	cc0	cc0
SNaN	cc3	cc3	cc0	cc0
Explanation:				
$C(e_a:e_b)$	Biased-exponent compare results. See Figure 20-16 on page 20-24.			
ccn	Condition code is set to n.			
F	All finite numbers, including zeros.			

Figure 20-15. Results: COMPARE BIASED EXPONENT

Relation of value e_a to value e_b	Condition Code for $C(e_a:e_b)$
$e_a = e_b$	cc0
$e_a < e_b$	cc1
$e_a > e_b$	cc2
Explanation:	
ccn	Condition code is set to n.
e_a	Biased exponent of the first operand
e_b	Biased exponent of the second operand

Figure 20-16. Biased-Exponent Compare Results

CONVERT FROM FIXED

Mnemonic	R_1, R_2	[RRE]
Op Code	//////	R_1 R_2
0	16	24 28 31

Mnemonic	Op Code	Operands
CDGTR	'B3F1'	64-bit binary-integer source, long DFP result
CXGTR	'B3F9'	64-bit binary-integer source, extended DFP result

Mnemonic2	R_1, M_3, R_2, M_4	[RRF-e]
Op Code	M_3 M_4 R_1 R_2	
0	16	20 24 28 31

Mnemonic2	Op Code	Operands
CDGTRA	'B3F1'	64-bit binary-integer source, long DFP result
CXGTRA	'B3F9'	64-bit binary-integer source, extended DFP result
CDFTR	'B951'	32-bit binary-integer source, long DFP result
CXFTR	'B959'	32-bit binary-integer source, extended DFP result

The fixed-point second operand is converted to the DFP format, and the result is placed at the first-operand location.

The second operand is a signed binary integer that is located in the general register designated by R_2 . A 32-bit operand is in bit positions 32-63 of the register.

The converted result is rounded and then the rounded value is placed at the first-operand location.

When the floating-point extension facility is installed, the converted result is rounded by rounding as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

- 0 According to the current DFP rounding mode
- 1 Round to nearest with ties away from 0
- 2 According to the current DFP rounding mode
- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$
- 8 Round to nearest with ties to even
- 9 Round toward 0
- 10 Round toward $+\infty$
- 11 Round toward $-\infty$
- 12 Round to nearest with ties away from 0
- 13 Round to nearest with ties toward 0
- 14 Round away from 0
- 15 Round to prepare for shorter precision

When the modifier field is zero or two, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 1, or 3-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception control is zero and recognition of the exception is not suppressed.

When the floating-point extension facility is not installed, no quantum exception is recognized. If bits 16-19 of the instruction contain a value of zero, the converted result is rounded according to the current DFP rounding mode; if bits 16-19 of the instruction contain a nonzero value, it is unpredictable which rounding method is performed.

When the delivered value is exact, the preferred quantum is one; when the delivered value is inexact, the preferred quantum is the smallest quantum.

The result placed at the first-operand location is canonical.

See Figure 20-23 on page 20-37 for a detailed description of the results of this instruction.

When the floating-point extension facility is installed, bit 1 of the M_4 field is the IEEE-inexact-exception control (XxC), and bits 0, 2, and 3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of the exception is suppressed.

When the floating-point extension facility is not installed, and if bits 20-23 of the instruction are zeros, then recognition of IEEE-inexact exception is not suppressed; if bits 20-23 of the instruction contain a nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.

For CXGTR, CXGTRA, and CXFTR, the R_1 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

An operation exception is recognized if the DFP facility is not installed. For CDFTR and CXFTR, the operation exception is also recognized if the floating-point extension facility is not installed.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Inexact (CDGTR and CDGTRA only)
- Quantum (CDGTR and CDGTRA only, if the floating-point extension facility is installed)

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (CXGTR, CXGTRA, and CXFTR only)

Programming Note: Unassigned bits in the M_4 field are reserved for future extensions and should be set to zeros; otherwise, the program may not operate compatibly in the future.

CONVERT FROM LOGICAL

Mnemonic R_1, M_3, R_2, M_4 [RRF-e]

Op Code	M_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic	Op Code	Operands
CDLGTR	'B952'	64-bit binary-integer source, long DFP result
CXLGTR	'B95A'	64-bit binary-integer source, extended DFP result
CDLFTR	'B953'	32-bit binary-integer source, long DFP result
CXLFTR	'B95B'	32-bit binary-integer source, extended DFP result

The fixed-point second operand is converted to the DFP format, and the result is placed at the first-operand location.

The second operand is an unsigned binary integer that is located in the general register designated by R_2 . A 32-bit operand is in bit positions 32-63 of the register.

The converted result is rounded to an integer value by rounding as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

0	According to the current DFP rounding mode
1	Round to nearest with ties away from 0
2	According to the current DFP rounding mode
3	Round to prepare for shorter precision
4	Round to nearest with ties to even
5	Round toward 0
6	Round toward $+\infty$
7	Round toward $-\infty$
8	Round to nearest with ties to even
9	Round toward 0
10	Round toward $+\infty$
11	Round toward $-\infty$
12	Round to nearest with ties away from 0
13	Round to nearest with ties toward 0
14	Round away from 0
15	Round to prepare for shorter precision

When the modifier field is zero or two, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 1, or 3-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception control is zero and recognition of the exception is not suppressed.

Bit 1 of the M₄ field is the IEEE-inexact-exception control (XxC). Bits 0, 2, and 3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of the exception is suppressed.

The result always has a plus sign.

When the delivered value is exact, the preferred quantum is one; when the delivered value is inexact, the preferred quantum is the smallest quantum.

The result placed at the first-operand location is canonical.

See Figure 20-23 on page 20-37 for a detailed description of the results of this instruction.

For CXLGTR and CXLFTR, the R₁ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Inexact (CDLGTR only)
- Quantum (CDLGTR only)

Program Exceptions:

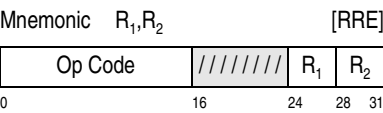
- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the floating-point extension facility is not installed)
- Specification (CXLGTR and CXLFTR only)

Programming Notes:

1. Unassigned bits in the M₄ field are reserved for future extensions and should be set to zeros; otherwise, the program may not operate compatibly in the future.
2. Using either the round toward 0 or the round toward $-\infty$ rounding method produces the same

result; using either the round away from 0 or the round toward $+\infty$ rounding method produces the same result.

CONVERT FROM SIGNED PACKED



Mnemonic	Op Code	Operands
CDSTR	'B3F3'	64-bit signed-packed-decimal source in GR, long DFP result
CXSTR	'B3FB'	128-bit signed-packed-decimal source in GRs, extended DFP result

The second operand in the signed-packed-decimal format is converted to the DFP format, and the result is placed at the first-operand location.

The second operand is located in the general register or general-register pair designated by R₂.

The preferred quantum is one, and the delivered value is always represented with the preferred quantum.

The result placed at the first-operand location is canonical.

When an invalid digit or sign code is detected in the second operand, a decimal-operand data exception is recognized.

For CXSTR, the R₁ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized. Also, the R₂ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 0, decimal operand
- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (CXSTR only)

CONVERT FROM UNSIGNED PACKED

Mnemonic	R_1, R_2	[RRE]
	Op Code	//////// R_1 R_2
	0	16 24 28 31

Mnemonic	Op Code	Operands
CDUTR	'B3F2'	64-bit unsigned-packed-decimal source in GR, long DFP result
CXUTR	'B3FA'	128-bit unsigned-packed-decimal source in GRs, extended DFP result

The second operand in the unsigned-packed-decimal format is converted to the DFP format with a positive sign, and the result is placed at the first-operand location.

The second operand is located in the general register or general-register pair designated by R_2 .

The preferred quantum is one, and the delivered value is always represented with the preferred quantum.

The result placed at the first-operand location is canonical.

When an invalid digit code is detected in the second operand, a decimal-operand data exception is recognized.

For CXUTR, the R_1 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized. Also, the R_2 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 0, decimal operand
- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (CXUTR only)

CONVERT TO FIXED

Mnemonic1	R_1, M_3, R_2	[RRF-e]
	Op Code	M_3 //// R_1 R_2
	0	16 20 24 28 31

Mnemonic1	Op Code	Operands
CGDTR	'B3E1'	Long DFP source, 64-bit binary-integer result
CGXTR	'B3E9'	Extended DFP source, 64-bit binary-integer result

Mnemonic2	R_1, M_3, R_2, M_4	[RRF-e]
	Op Code	M_3 M_4 R_1 R_2
	0	16 20 24 28 31

Mnemonic2	Op Code	Operands
CGDTRA	'B3E1'	Long DFP source, 64-bit binary-integer result
CGXTRA	'B3E9'	Extended DFP source, 64-bit binary-integer result
CFDTR	'B941'	Long DFP source, 32-bit binary-integer result
CFXTR	'B949'	Extended DFP source, 32-bit binary-integer result

The DFP second operand is rounded to an integer value and then converted to the fixed-point format. The result is placed at the first-operand location.

The result is a signed binary integer that is placed in the general register designated by R_1 . A 32-bit result replaces bits 32-63 of the register, and bits 0-31 of the register remain unchanged.

If the second operand is numeric and finite, it is rounded to an integer value by rounding as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

- | | |
|----|--|
| 0 | According to the current DFP rounding mode |
| 1 | Round to nearest with ties away from 0 |
| 2 | According to the current DFP rounding mode |
| 3 | Round to prepare for shorter precision |
| 4 | Round to nearest with ties to even |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |
| 7 | Round toward $-\infty$ |
| 8 | Round to nearest with ties to even |
| 9 | Round toward 0 |
| 10 | Round toward $+\infty$ |
| 11 | Round toward $-\infty$ |
| 12 | Round to nearest with ties away from 0 |

M₃ Effective Rounding Method

- 13 Round to nearest with ties toward 0
- 14 Round away from 0
- 15 Round to prepare for shorter precision

When the modifier field is zero, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 8-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

When the floating-point extension facility is installed and if the modifier field is 2, rounding is controlled by the current DFP rounding mode specified in the FPC register; if the field is 1, or 3-7, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

When the floating-point extension facility is not installed, and if the M₃ modifier field is 1-7, it is unpredictable which rounding method is performed.

The sign of the result is the sign of the second operand, except that a zero result has a plus sign.

This operation performs a functionally-constrained rounding. Neither overflow nor underflow condition can occur.

When the floating-point extension facility is installed, bit 1 of the M₄ field is the IEEE-inexact-exception control (XxC), and bits 0, 2, and 3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of the exception is suppressed.

When the floating-point extension facility is not installed, and if bits 20-23 of the instruction are zeros, then recognition of IEEE-inexact exception is not suppressed; if bits 20-23 of the instruction contain a

nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.

See Figure 20-17 for a detailed description of the results of this instruction.

For CGXTR, CGXTRA, and CFXTR, the R₂ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

An operation exception is recognized if the DFP facility is not installed. For CFDTR, CFDTRA, CFXTR, and CFXTRA the operation exception is also recognized if the floating-point extension facility is not installed.

Resulting Condition Code:

- 0 Source was zero
- 1 Source was less than zero
- 2 Source was greater than zero
- 3 Special case

IEEE Exceptions:

- Invalid operation
- Inexact

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation
- Specification (CGXTR, CGXTRA, and CFXTR only)

Programming Note: Unassigned bits in the M₄ field are reserved for future extensions and should be set to zeros; otherwise, the program may not operate compatibly in the future.

Operand (b)	Is n Inexact (n ≠ b)	Inv.-Op. Mask (FPC 0.0)	IEEE Inexact Exception Control ² (XxC)	Inexact Mask (FPC 0.4)	Is n Incremented (n > b)	Results
$-\infty \leq b < \text{MN}, n < \text{MN}$	—	0	—	—	—	T(MN), SFi←1, cc3 †
$-\infty \leq b < \text{MN}, n < \text{MN}$	—	1	—	—	—	PIDi(80)
$-\infty < b < \text{MN}, n = \text{MN}$	—	—	1	—	—	T(MN), cc1
$-\infty < b < \text{MN}, n = \text{MN}$	—	—	0	0	—	T(MN), SFx←1, cc1
$-\infty < b < \text{MN}, n = \text{MN}$	—	—	0	1	—	T(MN), cc1, PIDx(08)
$\text{MN} \leq b < 0$	No	—	—	—	—	T(f), cc1

Figure 20-17. Results: CONVERT TO FIXED (Part 1 of 2)

Operand (b)	Is n Inexact (n ≠ b)	Inv.-Op. Mask (FPC 0.0)	IEEE Inexact Exception Control ² (XxC)	Inexact Mask (FPC 0.4)	Is n Incremented (n > b)	Results
$MN \leq b < 0$	Yes	—	1	—	—	T(f), cc1
$MN \leq b < 0$	Yes	—	0	0	—	T(f), SFx←1, cc1
$MN \leq b < 0$	Yes	—	0	1	No	T(f), cc1, PIDx(08)
$MN \leq b < 0$	Yes	—	0	1	Yes	T(f), cc1, PIDy(0C)
-0	No ¹	—	—	—	—	T(0), cc0
+0	No ¹	—	—	—	—	T(0), cc0
$0 < b \leq MP$	No	—	—	—	—	T(f), cc2
$0 < b \leq MP$	Yes	—	1	—	—	T(f), cc2
$0 < b \leq MP$	Yes	—	0	0	—	T(f), SFx←1, cc2
$0 < b \leq MP$	Yes	—	0	1	No	T(f), cc2, PIDx(08)
$0 < b \leq MP$	Yes	—	0	1	Yes	T(f), cc2, PIDy(0C)
$MP < b < +\infty$, n = MP	—	—	1	—	—	T(MP), cc2
$MP < b < +\infty$, n = MP	—	—	0	0	—	T(MP), SFx←1, cc2
$MP < b < +\infty$, n = MP	—	—	0	1	—	T(MP), cc2, PIDx(08)
$MP < b \leq +\infty$, n > MP	—	0	—	—	—	T(MP), SFi←1, cc3 †
$MP < b \leq +\infty$, n > MP	—	1	—	—	—	PIDi(80)
NaN	—	0	—	—	—	T(MN), SFi←1, cc3 †
NaN	—	1	—	—	—	PIDi(80)

Explanation:

— The results do not depend on this condition or mask bit.

¹ This condition is true by virtue of the state of some condition to the left of this column.

² XxC is defined only when the floating-point extension facility is installed.

ccn Condition code is set to n.

† Result differs from BFP CONVERT TO FIXED. The BFP instruction recognizes inexact for this case and takes different actions depending on the inexact mask. The DFP implementation is intended to be more in keeping with the spirit of the IEEE standard.

f The value n converted to a fixed-point result.

n The value derived when the source value (b) is rounded to a floating-point integer using the effective rounding method.

MN Maximum negative number representable in the target fixed-point format.

MP Maximum positive number representable in the target fixed-point format.

PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. Figure 20-8 on page 20-15

SFi IEEE invalid-operation flag, FPC 1.0.

SFx IEEE inexact flag, FPC 1.4.

T(x) The value x is placed at the target operand location

Figure 20-17. Results: CONVERT TO FIXED (Part 2 of 2)

CONVERT TO LOGICAL

Mnemonic R_1, M_3, R_2, M_4 [RRF-e]

Op Code	M ₃	M ₄	R ₁	R ₂
0	16	20	24	28 31

Mnemonic	Op Code	Operands
CLGDTR	'B942'	Long DFP source, 64-bit binary-integer result
CLGXTR	'B94A'	Extended DFP source, 64-bit binary-integer result
CLFDTR	'B943'	Long DFP source, 32-bit binary-integer result
CLFXTR	'B94B'	Extended DFP source, 32-bit binary-integer result

The DFP second operand is rounded to an integer value and then converted to the fixed-point format. The result is placed at the first-operand location.

The result is an unsigned binary integer that is placed in the general register designated by R_1 . A 32-bit result replaces bits 32-63 of the register, and bits 0-31 of the register remain unchanged.

If the second operand is numeric and finite, it is rounded to an integer value by rounding as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

- 0 According to the current DFP rounding mode
- 1 Round to nearest with ties away from 0
- 2 According to the current DFP rounding mode
- 3 Round to prepare for shorter precision
- 4 Round to nearest with ties to even
- 5 Round toward 0
- 6 Round toward $+\infty$
- 7 Round toward $-\infty$
- 8 Round to nearest with ties to even
- 9 Round toward 0
- 10 Round toward $+\infty$
- 11 Round toward $-\infty$
- 12 Round to nearest with ties away from 0
- 13 Round to nearest with ties toward 0
- 14 Round away from 0
- 15 Round to prepare for shorter precision

When the modifier field is zero or two, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 1, or 3-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

:

This operation performs a functionally-constrained rounding. Neither overflow nor underflow condition can occur.

Bit 1 of the M_4 field is the IEEE-inexact-exception control (XxC). Bits 0, 2, and 3 are ignored. If XxC is

zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of the exception is suppressed.

See Figure 20-18 for a detailed description of the results of this instruction.

For CLGXTR and CLFXTR, the R_2 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Source was zero
- 1 Source was less than zero
- 2 Source was greater than zero
- 3 Special case

IEEE Exceptions:

- Invalid operation
- Inexact

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the floating-point extension facility is not installed.)
- Specification (CLGXTR and CLFXTR only)

Programming Note: Unassigned bits in the M_4 field are reserved for future extensions and should be set to zeros; otherwise, the program may not operate compatibly in the future.

Operand (b)	Is n Inexact ($n \neq b$)	Inv.-Op. Mask (FPC 0.0)	IEEE Inexact Exception Control ² (XxC)	Inexact Mask (FPC 0.4)	Is n Incremented ($ n > b $)	Results
$-\infty \leq b < 0, n < 0$	—	0	—	—	—	T(0), SFi←1, cc3 †
$-\infty \leq b < 0, n < 0$	—	1	—	—	—	PIDi(80)
$-\infty < b < 0, n = 0$	—	—	1	—	—	T(0), cc1
$-\infty < b < 0, n = 0$	—	—	0	0	—	T(0), SFx←1, cc1
$-\infty < b < 0, n = 0$	—	—	0	1	—	T(0), cc1, PIDx(08)
-0	No ¹	—	—	—	—	T(0), cc0
+0	No ¹	—	—	—	—	T(0), cc0
$0 < b \leq \text{MU}$	No	—	—	—	—	T(f), cc2
$0 < b \leq \text{MU}$	Yes	—	1	—	—	T(f), cc2
$0 < b \leq \text{MU}$	Yes	—	0	0	—	T(f), SFx←1, cc2
$0 < b \leq \text{MU}$	Yes	—	0	1	No	T(f), cc2, PIDx(08)

Figure 20-18. Results: CONVERT TO LOGICAL (Part 1 of 2)

Operand (b)	Is n Inexact (n ≠ b)	Inv.-Op. Mask (FPC 0.0)	IEEE Inexact Exception Control ² (XxC)	Inexact Mask (FPC 0.4)	Is n Incremented (n > b)	Results
$0 < b \leq \text{MU}$	Yes	—	0	1	Yes	T(f), cc2, PIDy(0C)
$\text{MU} < b < +\infty, n = \text{MU}$	—	—	1	—	—	T(MU), cc2
$\text{MU} < b < +\infty, n = \text{MU}$	—	—	0	0	—	T(MU), SFx←1, cc2
$\text{MU} < b < +\infty, n = \text{MU}$	—	—	0	1	—	T(MU), cc2, PIDx(08)
$\text{MU} < b \leq +\infty, n > \text{MU}$	—	0	—	—	—	T(MU), SFi←1, cc3 †
$\text{MU} < b \leq +\infty, n > \text{MU}$	—	1	—	—	—	PIDi(80)
NaN	—	0	—	—	—	T(0), SFi←1, cc3 †
NaN	—	1	—	—	—	PIDi(80)

Explanation:

—	The results do not depend on this condition or mask bit.
1	This condition is true by virtue of the state of some condition to the left of this column.
2	XxC is defined only when the floating-point extension facility is installed.
ccn	Condition code is set to n.
†	Result differs from BFP CONVERT TO LOGICAL. The BFP instruction recognizes inexact for this case and takes different actions depending on the inexact mask. The DFP implementation is intended to be more in keeping with the spirit of the IEEE standard.
f	The value n converted to a fixed-point result.
n	The value derived when the source value (b) is rounded to a floating-point integer using the effective rounding method.
MU	Maximum unsigned number representable in the target fixed-point format.
PIDc(h)	Program interruption for data exception, condition c, with DXC of h in hex. Figure 20-8 on page 20-15
SFi	IEEE invalid-operation flag, FPC 1.0.
SFx	IEEE inexact flag, FPC 1.4.
T(x)	The value x is placed at the target operand location

Figure 20-18. Results: CONVERT TO LOGICAL (Part 2 of 2)

CONVERT TO SIGNED PACKED

Mnemonic R_1, R_2, M_4 [RRF-d]

Op Code	////	M ₄	R ₁	R ₂
0	16	20	24	28 31

Mnemonic Op Code Operands

CSDTR	'B3E3'	Long DFP source, 64-bit signedpacked-decimal result in GR
CSXTR	'B3EB'	Extended DFP source, 128-bit signedpacked-decimal result in GRs

Rightmost significant digits of the DFP second operand are converted to the signed-packed-decimal format, and the result is placed at the first-operand location.

For CSDTR, the rightmost 15 digits in the trailing significant and the sign bit of the second operand are converted to a 64-bit result (15 4-bit decimal digits and a 4-bit sign); for CSXTR, the rightmost 31 digits in the trailing significant and the sign bit of the second operand are converted to a 128-bit result (31 4-bit decimal digits and a 4-bit sign). The sign of the result is the sign of the second operand.

Bit 3 of the M₄ field is the plus sign-code selection bit. When the bit is zero, the plus sign is encoded as 1100; when the bit is one, the plus sign is encoded as 1111. Bits 0-2 are ignored.

The result is placed in the general register or general-register pair designated by R₁.

This operation is performed for any second operand, including an infinity, QNaN, or SNaN, without causing an IEEE exception.

For CSXTR, the R₂ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized. Also, the R₁ field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

Program Exceptions:

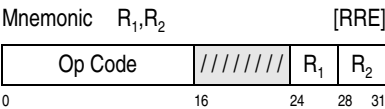
- Data with DXC 3, DFP instruction

- Operation (if the DFP facility is not installed)
- Specification (CSXTR only)

Programming Notes:

1. See also the programming note under CONVERT FROM FIXED.
2. The sign code 1100 is the System z preferred plus sign, and 1111 is the System i preferred plus sign.

CONVERT TO UNSIGNED PACKED



Mnemonic	Op Code	Operands
CUDTR	'B3E2'	Long DFP source, 64-bit unsigned-packed-decimal result in GR
CUXTR	'B3EA'	Extended DFP source, 128-bit unsigned-packed-decimal result in GRs

Rightmost significand digits of the DFP second operand are converted to the unsigned-packed-decimal format, and the result is placed at the first-operand location.

For CUDTR, 16 significand digits of the second operand are converted to a 64-bit result (16 4-bit decimal digits). If the second operand is an infinity or NaN, the 15 digits in the trailing significand are padded with a zero digit on the left to form a 16 significand digits.

For CUXTR, the rightmost 32 digits in the trailing significand of the second operand are converted to a 128-bit result (32 4-bit decimal digits).

The result is placed in the general register or general-register pair designated by R_1 .

This operation is performed for any second operand, including an infinity, QNaN, or SNaN, without causing an IEEE exception.

For CUXTR, the R_2 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized. Also, the R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

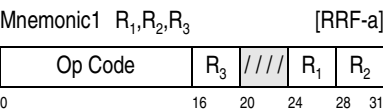
Condition Code: The code remains unchanged.

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (CUXTR only)

DIVIDE



Mnemonic1	Op Code	Operands
DDTR	'B3D1'	Long DFP
DXTR	'B3D9'	Extended DFP



Mnemonic2	Op Code	Operands
DDTRA	'B3D1'	Long DFP
DXTRA	'B3D9'	Extended DFP

The second operand (the dividend) is divided by the third operand (the divisor), and the quotient is placed at the first-operand location. No remainder is preserved.

If divisor is nonzero and both the dividend and divisor are finite numbers, the second operand is divided by the third operand to form an intermediate quotient. The intermediate quotient, if nonzero, is rounded to the target format and the rounded value is then placed at the first-operand location.

When the floating-point extension facility is installed, the intermediate quotient is rounded by rounding as specified by the modifier in the M_4 field:

M_4 Effective Rounding Method

- | | |
|---|--|
| 0 | According to the current DFP rounding mode |
| 1 | Round to nearest with ties away from 0 |
| 2 | According to the current DFP rounding mode |
| 3 | Round to prepare for shorter precision |
| 4 | Round to nearest with ties to even |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |

M₄ Effective Rounding Method

7	Round toward $-\infty$
8	Round to nearest with ties to even
9	Round toward 0
10	Round toward $+\infty$
11	Round toward $-\infty$
12	Round to nearest with ties away from 0
13	Round to nearest with ties toward 0
14	Round away from 0
15	Round to prepare for shorter precision

When the modifier field is zero or two, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 1, or 3-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception control is zero and recognition of the exception is not suppressed.

When the floating-point extension facility is not installed, no quantum exception is recognized. If bits 20-23 of the instruction contain a value of zero, the intermediate quotient is rounded according to the current DFP rounding mode; if bits 20-23 of the instruction contain a nonzero value, it is unpredictable which rounding method is performed.

When the dividend is a finite number and the divisor is infinity, then a value of zero with zero significand and zero biased exponent is produced.

The sign of the quotient, if the quotient is numeric, is the exclusive or of the operand signs. This includes the sign of a zero or infinite quotient.

When the delivered value is exact, the preferred quantum is the quantum of the dividend divided by the quantum of the divisor; when the delivered value is inexact, the preferred quantum is the smallest quantum.

The result placed at the first-operand location is canonical.

If the divisor is zero but the dividend is nonzero and finite, an IEEE-division-by-zero exception is recognized. If the dividend and divisor are both zero, or if both are infinite, regardless of sign, an IEEE-invalid-operation exception is recognized.

See Figure 20-19 on page 20-34 for a detailed description of the results of this instruction.

For DXTR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Division by zero
- Overflow
- Underflow
- Inexact
- Quantum (if the floating-point extension facility is installed)

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (DXTR and DXTRA only)

Programming Note: For DIVIDE, when the delivered value is inexact, the preferred quantum is the smallest quantum which is not always the same quantum as that closest to the quantum of the dividend divided by the quantum of the divisor.

For example, with the short data format, when 5000001×10^{-7} is divided by 400×10^0 , the smallest quantum for the delivered value is 1×10^{-9} . However, the quantum that is closest to the quantum of the dividend divided by the quantum of the divisor is 1×10^{-7} .

Dividend (b) is	Results for DIVIDE (b ÷ c) when divisor (c) is							
	-∞	-Fn	-0	+0	+Fn	+∞	QNaN	SNaN
-∞	Xi: T(dNaN)	T(+∞)	T(+∞)	T(-∞)	T(-∞)	Xi: T(dNaN)	T(c)	Xi: T(c*)
-Fn	T(+zt)	R(b÷c)	Xz: T(+∞)	Xz: T(-∞)	R(b÷c)	T(-zt)	T(c)	Xi: T(c*)
-0	T(+zt)	T(+0)	Xi: T(dNaN)	Xi: T(dNaN)	T(-0)	T(-zt)	T(c)	Xi: T(c*)
+0	T(-zt)	T(-0)	Xi: T(dNaN)	Xi: T(dNaN)	T(+0)	T(+zt)	T(c)	Xi: T(c*)
+Fn	T(-zt)	R(b÷c)	Xz: T(-∞)	Xz: T(+∞)	R(b÷c)	T(+zt)	T(c)	Xi: T(c*)
+∞	Xi: T(dNaN)	T(-∞)	T(-∞)	T(+∞)	T(+∞)	Xi: T(dNaN)	T(c)	Xi: T(c*)
QNaN	T(b)	T(b)	T(b)	T(b)	T(b)	T(b)	T(b)	Xi: T(c*)
SNaN	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)

Explanation:

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

dNaN Default NaN.

Fn Nonzero finite number (includes both subnormal and normal numbers).

R(x) Rounding and range action is performed on the value x. See Figure 20-5 on page 20-10. The result is canonical.

T(x) The canonical result x is placed at the target operand location.

Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Xz: IEEE division-by-zero exception. The results shown are produced only when FPC 0.1 is zero.

zt A value of zero with zero significand and zero biased exponent.

Figure 20-19. Results: DIVIDE

EXTRACT BIASED EXPONENT

Mnemonic	R ₁ ,R ₂	[RRE]
Op Code	//////	R ₁ R ₂
0	16	24 28 31
Mnemonic	Op Code	Operands
EEDTR	'B3E5'	Long DFP source, 64-bit binary-integer result
EEXTR	'B3ED'	Extended DFP source, 64-bit binary-integer result

The biased exponent of the DFP second operand is placed at the first-operand location.

When the second operand is a finite number, the biased exponent of the second operand is placed at the first-operand location. When the second operand is an infinity, QNaN, or SNaN, a special value is placed at the first-operand location.

The result is a 64-bit signed binary integer that is placed in the general register designated by R₁.

This operation is performed for any second operand, including an infinity, QNaN, or SNaN, without causing an IEEE exception.

See Figure 20-20 for a detailed description of the results of this instruction.

Second-operand data	First-operand value
Finite number	e
Infinity	-1
QNaN	-2
SNaN	-3

Explanation:

e Biased exponent

Figure 20-20. Results: EXTRACT BIASED EXPONENT

For EEXTR, the R₂ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

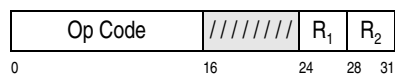
IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (EEXTR only)

EXTRACT SIGNIFICANCE

Mnemonic R_1, R_2 [RRE]



Mnemonic	Op Code	Operands
ESDTR	'B3E7'	Long DFP source, 64-bit binary-integer result
ESXTR	'B3EF'	Extended DFP source, 64-bit binary-integer result

The number of DFP significant digits of the DFP second operand is placed at the first-operand location.

When the second operand is a finite number, the result is set to the number of DFP significant digits of the second operand. When the second operand is an infinity, QNaN, or SNaN, the result is set to -1, -2, or -3, respectively.

The result is a 64-bit signed binary integer that is placed in the general register designated by R_1 .

See Figure 20-21 for a detailed description of the results of this instruction.

Second-operand data	First-operand value
Nonzero finite number	NSD
Zero	0
Infinity	-1
QNaN	-2
SNaN	-3
Explanation:	
NSD	The number of DFP significant digits of the second operand.

Figure 20-21. Results: EXTRACT SIGNIFICANCE

This operation is performed for any second operand, including an infinity, QNaN, or SNaN, without causing an IEEE exception.

For ESXTR, the R_2 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

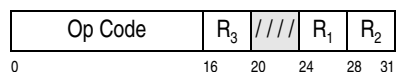
IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (ESXTR only)

INSERT BIASED EXPONENT

Mnemonic R_1, R_3, R_2 [RRF-b]



Mnemonic	Op Code	Operands
IEDTR	'B3F6'	Long DFP
IEXTR	'B3FE'	Extended DFP

A DFP operand is produced by combining the requested biased exponent with the sign bit and the significand of the DFP third operand, and the result is placed in the first-operand location.

The value of the requested biased exponent is a 64-bit signed binary integer and is located in the general register designated by R_2 .

When the value of the requested biased exponent is in the range between zero and the maximum biased exponent, inclusively, for the target format, the result is a finite number. The biased exponent of the result is set to the value of the requested biased exponent; the significand of the result is set to the significand of the third operand. If the third operand is an infinity or NaN, the significand of the third operand contains the digits in the trailing significand of the third operand padded with a zero digit on the left.

When the value of the requested biased exponent is -1, the result is an infinity. The reserved field of the result is set to zero; the trailing significand of the result is set to the trailing significand of the third operand.

When the value of the requested biased exponent is equal to -2, less than -3, or greater than the maximum biased exponent for the target format, the result is a QNaN; when the value of the requested biased exponent is -3, the result is an SNaN. When a NaN is produced as the result, the reserved field of the result is set to zero, and the trailing significand of the result is set to the trailing significand of the third operand.

The sign of the result is the same as the sign of the third operand.

The preferred quantum is the quantum that corresponds to the requested biased exponent. If the delivered value is a finite number, it is always represented with the preferred quantum.

The result placed at the first-operand location is canonical, except for infinity. When the result is an infinity, if all digits in the trailing significand of the third operand are zeros, then the result is a canonical infinity; otherwise, the result is an infinity that has the reserved field set to zero, canonical decrets in the encoded trailing-significand field, and some nonzero digits in the trailing significand.

See Figure 20-22 for a detailed description of the results of this instruction.

This operation is performed for any requested biased exponent and any third operand without causing an IEEE exception.

Value (b) in second operand	Results ¹ for INSERT BIASED EXPONENT when third operand (c) is			
	F	∞	QNaN	SNaN
$b > \text{MBE}$	T(QNaN)	T(QNaN)	T(QNaN)	T(QNaN)
$\text{MBE} \geq b \geq 0$	T(F)	T(F^2)	T(F^2)	T(F^2)
$b = -1$	N(∞)	N(∞)	N(∞)	N(∞)
$b = -2$	T(QNaN)	T(QNaN)	T(QNaN)	T(QNaN)
$b = -3$	T(SNaN)	T(SNaN)	T(SNaN)	T(SNaN)
$b \leq -4$	T(QNaN)	T(QNaN)	T(QNaN)	T(QNaN)
Explanation:				
¹	The sign of the result is the same as the sign of the third operand.			
²	The leftmost digit of the significand is zero.			
F	All finite numbers, including zeros.			
MBE	Maximum biased exponent for the target format.			
N(∞)	The result is a canonical infinity if all digits in the trailing significand of the third operand are zeros; otherwise, the resultant infinity has the reserved field set to zero, canonical decrets in the encoded trailing-significand field, and some nonzero digits in the trailing significand.			
T(x)	The canonical result x is placed at the target operand location.			

Figure 20-22. Results: INSERT BIASED EXPONENT

For IEXTR, the R₁ and R₃ fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

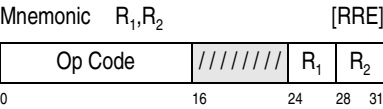
IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (IEXTR only)

Programming Note: Infinities produced as DFP results have the reserved field set to zero and have canonical decrets in the encoded trailing-significand field. An infinity is not considered canonical unless all digits in the trailing significand are zeros.

LOAD AND TEST



Mnemonic	Op Code	Operands
LTDTR	'B3D6'	Long DFP
LTXTR	'B3DE'	Extended DFP

The second operand is placed at the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code. The condition code is set the same as for a comparison of the second operand with zero.

The second operand is canonicalized before it is placed at the first-operand location. If the second operand is an SNaN, an IEEE-invalid-operation exception is recognized; if there is no interruption, the result is the corresponding QNaN.

The preferred quantum is the quantum of the second operand. If the delivered value is a finite number, it is always represented with the preferred quantum.

The result placed at the first-operand location is canonical.

See Figure 20-23 for a detailed description of the results of this instruction.

For LTXTR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero

3 Result is a NaN

Program Exceptions:

IEEE Exceptions:

- Invalid operation

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (LTXTX only)

Instruction	Results for some instructions with a single operand (b) when second operand (b) is							
	$-\infty$	-Fn	-0	+0	+Fn	$+\infty$	QNaN	SNaN
CONVERT FROM FIXED	—	Rf(b)	—	T(+0)	Rf(b)	—	—	—
CONVERT FROM LOGICAL	—	—	—	T(+0)	Rf(b)	—	—	—
LOAD AND TEST	T($-\infty$)	T(b)	T(-0)	T(+0)	T(b)	T($+\infty$)	T(b)	Xi: T(b*)
LOAD LENGTHENED (XiC = 0)	T($-\infty$)	T(b) ¹	T(-0)	T(+0)	T(b) ¹	T($+\infty$)	T(b) ¹	Xi: T(b*) ¹
LOAD LENGTHENED (XiC = 1)	N($-\infty$) ¹	T(b) ¹	T(-0)	T(+0)	T(b) ¹	N($+\infty$) ¹	T(b) ¹	T(b) ¹
LOAD ROUNDED (XiC = 0)	T($-\infty$)	R(b)	T(-0)	T(+0)	R(b)	T($+\infty$)	T(b) ²	Xi: T(b*) ²
LOAD ROUNDED (XiC = 1)	N($-\infty$) ²	R(b)	T(-0)	T(+0)	R(b)	N($+\infty$) ²	T(b) ²	T(b) ²

Explanation:

— The results do not depend on this condition.

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

¹ The operand is extended to the longer format by appending zeros on the left before it is placed at the target operand location.

² The operand is shortened to the target format by truncating the leftmost digits.

Fn Nonzero finite number (includes both subnormal and normal).

N($\pm\infty$) The resultant infinity has the reserved field set to zero and has canonical declets in the encoded trailing-significand field. The result is not considered canonical unless all digits in the trailing significand are zeros.

R(v) Rounding and range action is performed on the value v. See Figure 20-5 on page 20-10. The result is canonical.

Rf(b) The value b is converted to the exact floating-point number v, and then action R(v) is performed. The result is canonical.

T(x) The canonical result x is placed at the target operand location.

Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

XiC IEEE-invalid-operation-exception control, bit 0 of the M₄ field.

Figure 20-23. Results: Some Single-Operand Instructions

LOAD FP INTEGER

Mnemonic R₁,M₃,R₂,M₄ [RRF-e]

Op Code	M ₃	M ₄	R ₁	R ₂
0	16	20	24	28 31

Mnemonic	Op Code	Operands
FIDTR	'B3D7'	Long DFP
FIXTR	'B3DF'	Extended DFP

The second operand is rounded to an integer value in the same floating-point format, and the result is placed at the first-operand location.

The second operand, if a finite number, is rounded to an integer value as specified by the modifier in the M₃ field:

M₃ Effective Rounding Method

- | | |
|----|--|
| 0 | According to the current DFP rounding mode |
| 1 | Round to nearest with ties away from 0 |
| 2 | According to the current DFP rounding mode |
| 3 | Round to prepare for shorter precision |
| 4 | Round to nearest with ties to even |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |
| 7 | Round toward $-\infty$ |
| 8 | Round to nearest with ties to even |
| 9 | Round toward 0 |
| 10 | Round toward $+\infty$ |
| 11 | Round toward $-\infty$ |
| 12 | Round to nearest with ties away from 0 |
| 13 | Round to nearest with ties toward 0 |

M₃ Effective Rounding Method

- 14 Round away from 0
- 15 Round to prepare for shorter precision

When the modifier field is zero, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 8-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

When the floating-point extension facility is installed and if the modifier field is 2, rounding is controlled by the current DFP rounding mode specified in the FPC register; if the field is 1, or 3-7, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode. If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception control is zero and recognition of the exception is not suppressed.

When the floating-point extension facility is not installed, no quantum exception is recognized. If the M₃ modifier field is 1-7, it is unpredictable which rounding method is performed.

In the absence of an interruption, if the second operand is an infinity, the result is the canonical infinity; if the second operand is a QNaN, the result is the canonicalized source QNaN that has the same sign and payload as the source; if the second operand is an SNaN, the result is the corresponding QNaN.

The sign of the result is the sign of the second operand, even when the result is zero.

Bit 1 of the M₄ field is the IEEE-inexact-exception control (XxC). Bits 0, 2, and 3 are ignored.

When the result differs in value from the second operand, if XxC is zero, an IEEE-inexact exception is

recognized. If XxC one, then no IEEE-inexact exception is recognized.

This operation performs a functionally-constrained rounding. Neither overflow nor underflow condition can occur.

When the delivered value is exact, the preferred quantum is the larger value of one and the quantum of the second operand; when the delivered value is inexact, the preferred quantum is one. If the delivered value is a finite number, it is always represented with the preferred quantum.

The result placed at the first-operand location is canonical.

See Figure 20-24 on page 20-38 for a detailed description of the results of FIDTR and FIXTR.

When the floating-point extension facility is installed, if the quantum of the delivered finite number is not equal to the quantum of the second operand and if XqC is zero, then a quantum exception is recognized.

For FIXTR, the R fields must designate valid floating-point-register fields; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Inexact
- Quantum (if the floating-point extension facility is installed)

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (FIXTR only)

Second Operand (b)	Is n Inexact (n ≠ b)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is n Incremented (n > b)	Is Q < 1	Quantum-Exception Control (XqC) ²	Quantum Exception Mask (FPC 0.5)	Inv.-Op. Mask (FPC 0.0)	Results
-∞	No ¹	—	—	—	—	—	—	—	T(-∞)
-Fn	No	—	—	—	No	—	—	—	T(n)
-Fn	No	—	—	—	Yes	1	—	—	T(n)

Figure 20-24. Results: LOAD FP INTEGER

Second Operand (b)	Is n Inexact (n ≠ b)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is n Incremented (n > b)	Is Q < 1	Quantum-Exception Control (XqC) ²	Quantum Exception Mask (FPC 0.5)	Inv.-Op. Mask (FPC 0.0)	Results
-Fn	No	—	—	—	Yes	0	0	—	T(n), SFq←1
-Fn	No	—	—	—	Yes	0	1	—	T(n), PIDq(04)
-Fn	Yes	0	0	—	Yes	1	—	—	T(n), SFx←1
-Fn	Yes	0	0	—	Yes	0	0	—	T(n), SFx←1, SFq←1
-Fn	Yes	0	0	—	Yes	0	1	—	T(n), SFx←1, PIDq(04)
-Fn	Yes	0	1	No	—	—	—	—	T(n), PIDx(08)
-Fn	Yes	0	1	Yes	—	—	—	—	T(n), PIDy(0C)
-Fn	Yes	1	—	—	Yes	1	—	—	T(n)
-Fn	Yes	1	—	—	Yes	0	0	—	T(n), SFq←1
-Fn	Yes	1	—	—	Yes	0	1	—	T(n), PIDq(04)
-0	No ¹	—	—	—	No	—	—	—	T(-0)
-0	No ¹	—	—	—	Yes	1	—	—	T(-0)
-0	No ¹	—	—	—	Yes	0	0	—	T(-0), SFq←1
-0	No ¹	—	—	—	Yes	0	1	—	T(-0), PIDq(04)
+0	No ¹	—	—	—	No	—	—	—	T(+0)
+0	No ¹	—	—	—	Yes	1	—	—	T(+0)
+0	No ¹	—	—	—	Yes	0	0	—	T(+0), SFq←1
+0	No ¹	—	—	—	Yes	0	1	—	T(+0), PIDq(04)
+Fn	No	—	—	—	No	—	—	—	T(n)
+Fn	No	—	—	—	Yes	1	—	—	T(n)
+Fn	No	—	—	—	Yes	0	0	—	T(n), SFq←1
+Fn	No	—	—	—	Yes	0	1	—	T(n), PIDq(04)
+Fn	Yes	0	0	—	Yes	1	—	—	T(n), SFx←1
+Fn	Yes	0	0	—	Yes	0	0	—	T(n), SFx←1, SFq←1
+Fn	Yes	0	0	—	Yes	0	1	—	T(n), SFx←1, PIDq(04)
+Fn	Yes	0	1	No	—	—	—	—	T(n), PIDx(08)
+Fn	Yes	0	1	Yes	—	—	—	—	T(n), PIDy(0C)
+Fn	Yes	1	—	—	Yes	1	—	—	T(n)
+Fn	Yes	1	—	—	Yes	0	0	—	T(n), SFq←1
+Fn	Yes	1	—	—	Yes	0	1	—	T(n), PIDq(04)
+∞	No ¹	—	—	—	—	—	—	—	T(+∞)
QNaN	No ¹	—	—	—	—	—	—	—	T(b)
SNaN	No ¹	—	—	—	—	—	—	0	T(b*), SFi←1

Figure 20-24. Results: LOAD FP INTEGER

Second Operand (b)	Is n Inexact (n ≠ b)	IEEE Inexact Exception Control (XxC)	Inexact Mask (FPC 0.4)	Is n Incremented (nl > bl)	Is Q < 1	Quantum-Exception Control (XqC) ²	Quantum Exception Mask (FPC 0.5)	Inv.-Op. Mask (FPC 0.0)	Results
SNaN	No ¹	—	—	—	—	—	—	1	PIDi(80)
Explanation: — The results do not depend on this condition or mask bit. * The SNaN is converted to the corresponding QNaN before it is placed at the target operand location. ¹ This condition is true by virtue of the state of some condition to the left of this column. ² XqC is defined only if the floating-point extension facility is installed. n The value derived when the source value, b, is rounded to an integer using the effective rounding method. Fn Finite nonzero number (includes both subnormal and normal). PIDc(h) Program interruption for data exception, condition c, with DXC of h in hex. See Figure 20-8 on page 20-15. PIDq(04) Program interruption for data exception, quantum-exception condition q, with DXC of 04 in hex. It is defined only if the floating-point extension facility is installed. Q Quantum of the second operand. SFi IEEE invalid-operation flag, FPC 1.0. SFq Quantum exception flag, FPC 1.5. The flag is defined only if the floating-point extension facility is installed. SFx IEEE inexact flag, FPC 1.4. T(x) The canonical result x is placed at the target operand location. The result, when it is a finite number, is always represented with the preferred quantum.									

Figure 20-24. Results: LOAD FP INTEGER

Programming Notes:

1. Unassigned bits in the M₄ field are reserved for future extensions and should be set to zeros; otherwise, the program may not operate compatibly in the future.
2. If the DFP source operand is numeric with a quantum of at least one so that it is already an integer, the result value and quantum remain the same. Other numeric DFP source operands that are integers with a quantum less than one are converted to the same value with a quantum of one.

exception control (XiC). Bits 1-3 are ignored. When XiC is zero, recognition of IEEE-invalid-operation exception is not suppressed; when XiC is one, recognition of the exception is suppressed.

When the second operand is a finite number, the value of the second operand is placed in the target format.

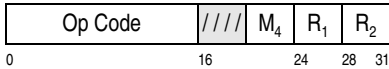
When the second operand is an infinity, if XiC is zero, the result is the canonical infinity for the target format; if XiC is one, the result is the source infinity with the reserved field of the target format being set to zero, the trailing significand being extended by appending zeros on the left, and all declets in the encoded trailing-significand field being canonicalized.

When the second operand is a QNaN, the result is the canonicalized source QNaN with the payload extended by appending zeros on the left.

When the second operand is an SNaN, if XiC is zero, an invalid-operation exception is recognized and the nontrap result is the corresponding QNaN with the payload extended by appending zeros on the left; if XiC is one, no invalid-operation exception is recognized, and the result is the canonicalized source SNaN with the payload extended by appending zeros on the left.

LOAD LENGTHENED

Mnemonic R₁,R₂,M₄ [RRF-d]



Mnemonic	Op Code	Operands
LDETR	'B3D4'	Short DFP source, long DFP result
LXDTR	'B3DC'	Long DFP source, extended DFP result

The second operand is converted to a longer format, and the result is placed at the first-operand location.

Bit 0 of the M₄ field controls the handling of SNaN and infinity, and is called the IEEE-invalid-operation-

The sign of the result is the same as the sign of the second operand.

The delivered value is always exact and the preferred quantum is the quantum of the second operand.

When XiC is zero, the result placed at the first-operand location is canonical. When XiC is one, the result is canonical, except for infinity. See the detailed description above for producing an infinity when XiC is one.

See Figure 20-23 on page 20-37 for a detailed description of the results of this instruction.

For LXDTR, the R_1 field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (LXDTR only)

Programming Notes:

1. Unassigned bits in the M_4 field are reserved for future extensions and should be set to zeros; otherwise, the program may not operate compatibly in the future.
2. The equivalent operations of LOAD LENGTHENED (short to long) in the Power architecture do not recognize the invalid-operation exception for both BFP and DFP. The equivalent operations of LOAD LENGTHENED in z/Architecture do recognize the exception for BFP. The IEEE-invalid-operation-exception control (XiC) is provided for compatibility.
3. Infinities produced as DFP results have the reserved field set to zero and have canonical declets in the encoded trailing-significand field. An infinity is not considered canonical unless all digits in the trailing significand are zeros.

LOAD ROUNDED

Mnemonic R_1, M_3, R_2, M_4 [RRF-e]

Op Code	M_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic	Op Code	Operands
LEDTR	'B3D5'	Long DFP source, short DFP target, short or long DFP result
LDXTR	'B3DD'	Extended DFP source, long DFP target, long or extended DFP result

The second operand, in the format of the source, is rounded to the precision of the target, and the result is placed at the first-operand location.

The second operand, if it is a finite number, is rounded as specified by the modifier in the M_3 field:

M_3 Effective Rounding Method

0	According to the current DFP rounding mode
1	Round to nearest with ties away from 0
2	According to the current DFP rounding mode
3	Round to prepare for shorter precision
4	Round to nearest with ties to even
5	Round toward 0
6	Round toward $+\infty$
7	Round toward $-\infty$
8	Round to nearest with ties to even
9	Round toward 0
10	Round toward $+\infty$
11	Round toward $-\infty$
12	Round to nearest with ties away from 0
13	Round to nearest with ties toward 0
14	Round away from 0
15	Round to prepare for shorter precision

When the modifier field is zero, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 8-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

When the floating-point extension facility is installed and if the modifier field is 2, rounding is controlled by the current DFP rounding mode specified in the FPC register; if the field is 1, or 3-7, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode. If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception con-

control is zero and recognition of the exception is not suppressed.

When the floating-point extension facility is not installed, no quantum exception is recognized. If the M_3 modifier field is 1-7, it is unpredictable which rounding method is performed.

Bit 0 of the M_4 field controls the handling of SNaN and infinity, and is called the IEEE-invalid-operation-exception control (XiC). When XiC is zero, recognition of IEEE-invalid-operation exception is not suppressed; when XiC is one, recognition of IEEE-invalid-operation exception is suppressed.

When the floating-point extension facility is installed, bit 1 of the M_4 field is the IEEE-inexact-exception control (XxC), and bits 2-3 are ignored. If XxC is zero, recognition of IEEE-inexact exception is not suppressed; if XxC is one, recognition of the exception is suppressed.

When the floating-point extension facility is not installed, and if bits 21-23 of the instruction are zeros, then recognition of IEEE-inexact exception is not suppressed; if bits 21-23 of the instruction contain a nonzero value, it is unpredictable whether recognition of IEEE-inexact exception is suppressed.

In the absence of a trap overflow or trap underflow, the result is in the format and length of the target. However, when a trap overflow or trap underflow occurs, the operation is completed by producing a scaled result in the same format and length as the source but rounded to the precision of the target. In this case, the result is selected from a subset of the cohort for the delivered value in the source format. This subset of the cohort consists of cohort members that have the number of DFP significand digits equal to or less than the precision of the target format. For LEDTR, the result has at most seven DFP significant digits; for LDXTTR, the result has at most 16 DFP significant digits.

When the second operand is an infinity, if XiC is zero, the result is the canonical infinity for the target format; if XiC is one, the result is the source infinity with the reserved field of the target format being set to zero, the trailing significand being shortened by removing leftmost digits, and declets corresponding to the shortened trailing significand being canonicalized.

When the second operand is a QNaN, the result is the canonicalized source QNaN with the payload shortened by removing leftmost digits.

When the second operand is an SNaN, if XiC is zero, an invalid-operation exception is recognized and the nontrap result is the corresponding QNaN with the payload shortened by removing leftmost digits; if XiC is one, no invalid-operation exception is recognized, and the result is the canonicalized source SNaN with the payload shortened by removing leftmost digits.

The sign of the result is the same as the sign of the second operand.

When the delivered value is exact, the preferred quantum is the quantum of the second operand; when the delivered value is inexact, the preferred quantum is the smallest quantum.

When XiC is zero, the result placed at the first-operand location is canonical. When XiC is one, the result is canonical, except for infinity. See the detailed description above for producing an infinity when XiC is one.

See Figure 20-23 on page 20-37 for a detailed description of the results of this instruction.

For LDXTTR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Overflow
- Underflow
- Inexact
- Quantum (if the floating-point extension facility is installed)

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (LDXTTR only)

Programming Notes:

1. LOAD ROUNDED and the round-to-prepare-for-shorter-precision rounding method can be used to simulate a DFP operation with the precision of a shorter format and with the effect of single rounding. The round-to-prepare-for-shorter-precision rounding method is used by the operation, and the desirable rounding method is used by the subsequent LOAD ROUNDED instruction.
2. Unassigned bits in the M_4 field are reserved for future extensions and should be set to zeros; otherwise, the program may not operate compatibly in the future.
3. The equivalent operation of LOAD ROUNDED (long to short) in the Power architecture does not recognize the invalid-operation exception. IEEE-invalid-operation-exception control (XiC) is provided for compatibility.
4. Infinities produced as DFP results have the reserved field set to zero and have canonical declets in the encoded trailing-significand field. An infinity is not considered canonical unless all digits in the trailing significand are zeros.
5. In case of a trap overflow or a trap underflow, the number of DFP significand digits of the delivered result is limited to the precision of the target format. For example, when the source operand is 75958100×10^{369} and is represented with the quantum of 10^{369} , execution of LEDTR using the round-away-from-0 rounding method produces a value of 7595810×10^{178} with the delivered quantum of 10^{178} as the trap overflow result.

MULTIPLYMnemonic1 R_1, R_2, R_3 [RRF-a]

Op Code	R_3	////	R_1	R_2
0	16	20	24	28 31

Mnemonic2 Op Code Operands
 MDTR 'B3D0' Long DFP
 MXTR 'B3D8' Extended DFP

Mnemonic2 R_1, R_2, R_3, M_4 [RRF-a]

Op Code	R_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic2 Op Code Operands
 MDTRA 'B3D0' Long DFP
 MXTRA 'B3D8' Extended DFP

The product of the second operand (the multiplicand) and the third operand (the multiplier) is placed at the first-operand location.

If both source operands are finite numbers, they are multiplied to form an intermediate product. The intermediate product is rounded to the target format and the rounded value is then placed at the first-operand location.

When the floating-point extension facility is installed, the intermediate product is rounded by rounding as specified by the modifier in the M_4 field:

 M_4 Effective Rounding Method

- | | |
|----|--|
| 0 | According to the current DFP rounding mode |
| 1 | Round to nearest with ties away from 0 |
| 2 | According to the current DFP rounding mode |
| 3 | Round to prepare for shorter precision |
| 4 | Round to nearest with ties to even |
| 5 | Round toward 0 |
| 6 | Round toward $+\infty$ |
| 7 | Round toward $-\infty$ |
| 8 | Round to nearest with ties to even |
| 9 | Round toward 0 |
| 10 | Round toward $+\infty$ |
| 11 | Round toward $-\infty$ |
| 12 | Round to nearest with ties away from 0 |
| 13 | Round to nearest with ties toward 0 |
| 14 | Round away from 0 |
| 15 | Round to prepare for shorter precision |

When the modifier field is zero or two, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 1, or 3-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception control is zero and recognition of the exception is not suppressed.

When the floating-point extension facility is not installed, no quantum exception is recognized. If bits 20-23 of the instruction contain a value of zero, the intermediate product is rounded according to the current DFP rounding mode; if bits 20-23 of the instruc-

tion contain a nonzero value, it is unpredictable which rounding method is performed.

The sign of the product, if the product is numeric, is the exclusive or of the operand signs. This includes the sign of a zero or infinite product.

When the delivered value is exact, the preferred quantum is the product of the quanta of the two source operands; when the delivered value is inexact, the preferred quantum is the smallest quantum.

The result placed at the first-operand location is canonical.

See Figure 20-25 on page 20-44 for a detailed description of the results of this instruction.

If one source operand is a zero and the other an infinity, an IEEE-invalid-operation exception is recognized

For MXTR and MXTRA, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Overflow
- Underflow
- Inexact
- Quantum (if the floating-point extension facility is installed)

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (MXTR and MXTRA only)

Multiplicand (b) is	Results for MULTIPLY (b • c) when multiplier (c) is							
	-∞	-Fn	-0	+0	+Fn	+∞	QNaN	SNaN
-∞	T(+∞)	T(+∞)	Xi: T(dNaN)	Xi: T(dNaN)	T(-∞)	T(-∞)	T(c)	Xi: T(c*)
-Fn	T(+∞)	R(b • c)	T(+0)	T(-0)	R(b • c)	T(-∞)	T(c)	Xi: T(c*)
-0	Xi: T(dNaN)	T(+0)	T(+0)	T(-0)	T(-0)	Xi: T(dNaN)	T(c)	Xi: T(c*)
+0	Xi: T(dNaN)	T(-0)	T(-0)	T(+0)	T(+0)	Xi: T(dNaN)	T(c)	Xi: T(c*)
+Fn	T(-∞)	R(b • c)	T(-0)	T(+0)	R(b • c)	T(+∞)	T(c)	Xi: T(c*)
+∞	T(-∞)	T(-∞)	Xi: T(dNaN)	Xi: T(dNaN)	T(+∞)	T(+∞)	T(c)	Xi: T(c*)
QNaN	T(b)	T(b)	T(b)	T(b)	T(b)	T(b)	T(b)	Xi: T(c*)
SNaN	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)

Explanation:

* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.

dNaN Default NaN.

Fn Nonzero finite number (includes both subnormal and normal).

R(v) Rounding and range action is performed on the value v. See Figure 20-5 on page 20-10. The result is canonical.

T(x) The canonical result x is placed at the target operand location.

Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 20-25. Results: MULTIPLY

Programming Notes:

1. Interchanging the two operands in a DFP multiplication does not affect the value of the product when the result is numeric. This is not true, however, when both operands are QNaNs. in which case the result is the canonical QNaN derived from the second operand; or when both operands are SNaNs and the IEEE invalid-operation

mask bit in the FPC register is zero, in which case the result is the canonical QNaN derived from the second operand.

2. For MULTIPLY, when the delivered value is inexact, the preferred quantum is the smallest quantum which is the same quantum as that closest to the product of the quanta of the two source operands.

QUANTIZE

Mnemonic R₁,R₃,R₂,M₄ [RRF-b]

Op Code	R ₃	M ₄	R ₁	R ₂
0	16	20	24	28 31

Mnemonic	Op Code	Operands
QADTR	'B3F5'	Long DFP
QAXTR	'B3FD'	Extended DFP

The third operand is rounded to a requested quantum, and the result is placed at the first-operand location.

The requested quantum is the quantum of the second operand.

When both the second and third operands are finite numbers, if the quantum of the third operand is equal to or larger than the requested quantum, the value of the third operand with the requested quantum is placed at the first-operand location. If the quantum of the third operand is smaller than the requested quantum, the third operand is rounded to the requested quantum by rounding as specified by the modifier in the M₄ field, and the result with the requested quantum is placed at the first-operand location.

M₄ Effective Rounding Method

0	According to the current DFP rounding mode
1	Round to nearest with ties away from 0
2	According to the current DFP rounding mode
3	Round to prepare for shorter precision
4	Round to nearest with ties to even
5	Round toward 0
6	Round toward $+\infty$
7	Round toward $-\infty$
8	Round to nearest with ties to even
9	Round toward 0
10	Round toward $+\infty$
11	Round toward $-\infty$
12	Round to nearest with ties away from 0
13	Round to nearest with ties toward 0
14	Round away from 0
15	Round to prepare for shorter precision

When the modifier field is zero, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 8-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

When the floating-point extension facility is installed and if the modifier field is 2, rounding is controlled by the current DFP rounding mode specified in the FPC register; if the field is 1, or 3-7, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode. If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception control is zero and recognition of the exception is not suppressed.

When the floating-point extension facility is not installed, no quantum exception is recognized. If the M₄ modifier field is 1-7, it is unpredictable which rounding method is performed.

The sign of the result, if numeric, is the same as the sign of the third operand.

This operation performs a functionally-constrained rounding, and does not recognize an underflow exception. No overflow condition can occur.

The preferred quantum is the requested quantum. If the delivered value is a finite number, it is always represented with the preferred quantum.

The result placed at the first-operand location is canonical.

See Figure 20-26 on page 20-46 for a detailed description of the results of this instruction.

An invalid-operation exception is recognized when the resultant value cannot be represented in the target format with the requested quantum. When the invalid-operation exception is recognized, and if the exception is disabled, the result is the default QNaN.

An IEEE inexact exception is recognized when the result differs in value from the third operand,

When the floating-point extension facility is installed, if the quantum of the delivered finite number is not equal to the quantum of the third operand and if XqC is zero, then a quantum exception is recognized.

For QAXTR, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Inexact
- Quantum (if the floating-point extension facility is installed)
- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (QAXTR only)

Program Exceptions:

When second operand (b) is	Result for QUANTIZE when third operand (c) is							
	$-\infty$	-Fn	-0	+0	+Fn	$+\infty$	QNaN	SNaN
$-\infty$	T($-\infty$)	Xi: T(dNaN)	Xi: T(dNaN)	Xi: T(dNaN)	Xi: T(dNaN)	T($+\infty$)	T(c)	Xi: T(c*)
-Fn	Xi: T(dNaN)	Q(b:c)	E(-0)	E(+0)	Q(b:c)	Xi: T(dNaN)	T(c)	Xi: T(c*)
-0	Xi: T(dNaN)	Q(b:c)	E(-0)	E(+0)	Q(b:c)	Xi: T(dNaN)	T(c)	Xi: T(c*)
+0	Xi: T(dNaN)	Q(b:c)	E(-0)	E(+0)	Q(b:c)	Xi: T(dNaN)	T(c)	Xi: T(c*)
+Fn	Xi: T(dNaN)	Q(b:c)	E(-0)	E(+0)	Q(b:c)	Xi: T(dNaN)	T(c)	Xi: T(c*)
$+\infty$	T($-\infty$)	Xi: T(dNaN)	Xi: T(dNaN)	Xi: T(dNaN)	Xi: T(dNaN)	T($+\infty$)	T(c)	Xi: T(c*)
QNaN	T(b)	T(b)	T(b)	T(b)	T(b)	T(b)	T(b)	Xi: T(c*)
SNaN	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)	Xi: T(b*)
Explanation:								
* The SNaN is converted to the corresponding QNaN before it is placed at the target operand location.								
dNaN Default NaN.								
E(0) The value zero with the quantum of operand b is placed at the target operand location.								
Fn Finite nonzero number (includes both subnormal and normal).								
Q(b:c) See Figure 20-27.								
T(x) The canonical result x is placed at the target operand location.								
Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.								

Figure 20-26. Results: QUANTIZE

Relation ¹	Is $ Vcl > (10^p - 1) \times 10^{Qb}$?	Result for Q(b:c)
$10^{Qb} < 10^{Qc}$	Yes	Xi: T(dNaN)
	No	L(c)
$10^{Qb} = 10^{Qc}$		T(c)
$10^{Qb} > 10^{Qc}$		Rq(c)
Explanation: ¹ The quantum of the second operand and the third operand is 10^{Qb} and 10^{Qc} , respectively. dNaN Default NaN. p Format precision. Qb The right-units-view exponent of the second operand (b). Qc The right-units-view exponent of the third operand (c). IVcl The absolute value of the third operand (c). L(c) The third operand (c) is represented with the requested quantum. The result is represented with the preferred quantum and is canonical. Rq(c) The third operand (c) is rounded to the requested quantum. The result is represented with the preferred quantum and is canonical. T(x) The canonical result x is placed at the target operand location. The result, when it is a finite number, is represented with the preferred quantum and is canonical. Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.		

Figure 20-27. Results: Q(b:c)

Programming Notes:

- When the third operand is larger than $(10^p - 1) \times 10^{Qb}$ in magnitude, an invalid-operation exception is recognized, where p is the format precision and 10^{Qb} is the requested quantum.

The expression, $(10^p - 1) \times 10^{Qb}$, specifies the largest magnitude that can be represented with the requested quantum in the format. A value larger than $(10^p - 1) \times 10^{Qb}$ in magnitude will have more than p digits when the value is represented with the requested quantum.

- When both source operands are finite numbers, an invalid-operation exception can occur only if the requested quantum is smaller than the quantum of the third operand; and an inexact exception can occur only if the requested quantum is larger than the quantum of the third operand.

REROUNDMnemonic R_1, R_3, R_2, M_4 [RRF-b]

Op Code	R_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic	Op Code	Operands
RRDTR	'B3F7'	Long DFP
RRXTR	'B3FF'	Extended DFP

The third operand is rounded to the requested significance, and the result is placed at the first-operand location.

The requested significance, which specifies a number of DFP significant digits, is an unsigned binary integer and is in bit positions 58-63 of general register R_2 . The contents of bit positions 0-57 of general register R_2 are ignored.

When the third operand is a finite number, if the requested significance is zero, then the value of the third operand with the original quantum is placed at the first-operand location.

When the third operand is a finite number, if the requested significance is nonzero, and if the number of DFP significant digits of the third operand is equal to or less than the requested significance, then the value of the third operand with the original quantum is placed at the first-operand location.

When the third operand is a finite number, if the requested significance is nonzero, and if the number of DFP significant digits of the third operand is greater than the requested significance, then the third operand is rounded to the requested significance, and the result with the requested significance is placed at the first-operand location.

The third operand is rounded to the requested significance by rounding as specified by the modifier in the M_4 field:

 M_4 Effective Rounding Method

0	According to the current DFP rounding mode
1	Round to nearest with ties away from 0
2	According to the current DFP rounding mode
3	Round to prepare for shorter precision
4	Round to nearest with ties to even
5	Round toward 0
6	Round toward $+\infty$
7	Round toward $-\infty$

M₄ Effective Rounding Method

- | | |
|----|--|
| 8 | Round to nearest with ties to even |
| 9 | Round toward 0 |
| 10 | Round toward $+\infty$ |
| 11 | Round toward $-\infty$ |
| 12 | Round to nearest with ties away from 0 |
| 13 | Round to nearest with ties toward 0 |
| 14 | Round away from 0 |
| 15 | Round to prepare for shorter precision |

When the modifier field is zero, rounding is controlled by the current DFP rounding mode specified in the FPC register. When the field is 8-15, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode.

When the floating-point extension facility is installed and if the modifier field is 2, rounding is controlled by the current DFP rounding mode specified in the FPC register; if the field is 1, or 3-7, rounding is performed as specified by the modifier, regardless of the current DFP rounding mode. If the modifier field is 0 or 8-15, the quantum-exception control (XqC) is one and recognition of the quantum exception is suppressed; if the modifier field is 1-7, the quantum-exception control is zero and recognition of the exception is not suppressed.

When the floating-point extension facility is not installed, no quantum exception is recognized. If the M₄ modifier field is 1-7, it is unpredictable which rounding method is performed.

The sign of the result, if numeric, is the same as the sign of the third operand.

This operation performs a functionally-constrained rounding, and does not recognize an underflow exception. No overflow condition can occur.

When the delivered value is exact, if the requested significance is zero, or if the requested significance is nonzero and the number of DFP significant digits of the third operand is equal to or less than the requested significance, then the preferred quantum is the quantum of the third operand. When the delivered value is exact, if the requested significance is nonzero and the number of DFP significant digits of the third operand is greater than the requested significance, then the preferred quantum is the quantum that corresponds to the requested significance.

When the delivered value is inexact, the preferred quantum is the quantum that corresponds to the requested significance.

If the delivered value is a finite number, it is always represented with the preferred quantum.

The result placed at the first-operand location is canonical.

See Figure 20-28 for a detailed description of the results of this instruction.

An invalid-operation exception is recognized when the resultant value cannot be represented in the target format with the requested significance. When the invalid-operation exception is recognized, and if the exception is disabled, the result is the default QNaN.

When an invalid-operation exception is recognized, the IEEE inexact exception is not recognized.

In the absence of an invalid-operation exception, if the result differs in value from the third operand, an IEEE inexact exception is recognized.

When the floating-point extension facility is installed, if the quantum of the delivered finite number is not equal to the quantum of the third operand and if XqC is zero, then a quantum exception is recognized.

For RRXTR, the R₁ and R₃ fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions:

- Invalid operation
- Inexact
- Quantum (if the floating-point extension facility is installed)

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (RRXTR only)

Significance Comparison	Is $ Rp(c) > (10^k - 1) \times 10^{Q_{max}}$?	Results for REROUND when the Third Operand (c) is						
		$-\infty$	$-Fn$	-0	$+0$	$+Fn$	$+\infty$	QNaN
$k = 0$	--	$T(-\infty)$	$T(c)$	$T(c)$	$T(c)$	$T(c)$	$T(+\infty)$	$T(c)$
$k \neq 0; k > m$	--	$T(-\infty)$	$T(c)$	$T(c)$	$T(c)$	$T(c)$	$T(+\infty)$	$T(c)$
$k \neq 0; k = m$	--	$T(-\infty)$	$T(c)$	--	--	$T(c)$	$T(+\infty)$	$T(c)$
$k \neq 0; k < m$	No	$T(-\infty)$	$Rp(c)$	--	--	$Rp(c)$	$T(+\infty)$	$T(c)$
	Yes	$T(-\infty)$	$Xi:T(dNaN)$	--	--	$Xi:T(dNaN)$	$T(+\infty)$	$T(c)$

Explanation:

-- Not applicable.
 * The SNaN is converted to the corresponding QNaN and placed at the target operand location.
 dNaN Default NaN.
 Fn Nonzero finite numbers (includes both subnormal and normal).
 k The requested significance.
 m Number of DFP significant digits in the third operand (c).
 Qmax Maximum right-units-view exponent.
 Rp(c) The third operand (c) is rounded to the requested significance. The result is canonical and is represented with the preferred quantum.
 |Rp(c)| The absolute value of Rp(c).
 T(x) The canonical result x is placed at the target operand location. The result is canonical and, when it is a finite number, is represented with the preferred quantum.
 Xi: IEEE invalid-operation exception. The results shown are produced only when FPC 0.0 is zero.

Figure 20-28. Results: REROUND

Programming Notes:

1. REROUND and the round-to-prepare-for-shorter-precision rounding method can be used to simulate a DFP operation with a precision other than any supported format precision and with the effect of single rounding. The round-to-prepare-for-shorter-precision rounding method is used by the operation, and the desirable rounding method is used by the subsequent REROUND instruction.
2. When the rounded intermediate value is larger than $(10^k - 1) \times 10^{Q_{max}}$ in magnitude, an invalid-operation exception is recognized, where k is the requested significance and $10^{Q_{max}}$ is the largest quantum for the format.

The expression, $(10^k - 1) \times 10^{Q_{max}}$, specifies the largest magnitude that can be represented with the requested significance in the format. A value larger than $(10^p - 1) \times 10^{Q_b}$ in magnitude will have a quantum larger than $10^{Q_{max}}$ when the value is represented with the requested significance.

SHIFT SIGNIFICAND LEFTMnemonic $R_1, R_3, D_2(X_2, B_2)$ [RXF]

OpCode	R_3	X_2	B_2	D_2	R_1	////	OpCode
0	8	12	16	20	32	36	40
							47

Mnemonic	Op Code	Operands
SLDT	'ED40'	Long DFP
SLXT	'ED48'	Extended DFP

The significand of the third operand is shifted left the number of digits specified by the second-operand address, and the result is placed at the first-operand location.

Digits shifted out of the leftmost digit are lost. Zeros are supplied to the vacated positions on the right. The sign of the result is the same as the sign of the third operand.

For a finite number, all digits in the significand participate in the shift and the result is a finite number with the same biased exponent as the third operand and the shifted significand. For an infinity, all digits in the trailing significand participate in the shift, and the result is an infinity with the shifted trailing significand and a zero in the reserved field of the format. For a QNaN or SNaN, all digits in the payload participate in the shift and the result is a QNaN or SNAN, respec-

tively, with the shifted payload and a zero in the reserved field of the format.

The second-operand address is not used to address data; its rightmost six bits indicate the number of digit positions to be shifted. The remainder of the address is ignored.

The preferred quantum is the quantum of the third operand. If the delivered value is a finite number, it is always represented with the preferred quantum.

The result placed at the first-operand location is canonical, except for infinity. When the result is an infinity, if all digits in the trailing significand of the result are zeros, then the result is canonical; otherwise, the result is an infinity that has the reserved field set to zero, canonical declets in the encoded trailing-significand field, and some nonzero digits in the trailing significand.

This operation is performed for any second operand, including an infinity, QNaN, or SNaN, without causing an IEEE exception.

For SLXT, the R_1 and R_3 fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

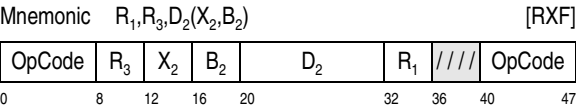
IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (SLXT only)

Programming Note: Infinities produced as DFP results have the reserved field set to zero and have canonical declets in the encoded trailing-significand field. An infinity is not considered canonical unless all digits in the trailing significand are zeros.

SHIFT SIGNIFICAND RIGHT



Mnemonic	Op Code	Operands
SRDT	'ED41'	Long DFP
SRXT	'ED49'	Extended DFP

The significand of the third operand is shifted right the number of digits specified by the second-operand address, and the result is placed at the first-operand location.

Digits shifted out of the rightmost digit are lost. Zeros are supplied to the vacated positions on the left. The sign of the result is the same as the sign of the third operand.

For a finite number, all digits in the significand participate in the shift and the result is a finite number with the same biased exponent as the third operand and the shifted significand. For an infinity, all digits in the trailing significand participate in the shift, and the result is an infinity with the shifted trailing significand and a zero in the reserved field of the format. For a QNaN or SNaN, all digits in the payload participate in the shift and the result is a QNaN or SNaN, respectively, with the shifted payload and a zero in the reserved field of the format.

The second-operand address is not used to address data; its rightmost six bits indicate the number of digit positions to be shifted. The remainder of the address is ignored.

The preferred quantum is the quantum of the third operand. If the delivered value is a finite number, it is always represented with the preferred quantum.

The result placed at the first-operand location is canonical, except for infinity. When the result is an infinity, if all digits in the trailing significand of the result are zeros, then the result is canonical; otherwise, the result is an infinity that has the reserved field set to zero, canonical declets in the encoded trailing-significand field, and some nonzero digits in the trailing significand.

This operation is performed for any second operand, including an infinity, QNaN, or SNaN, without causing an IEEE exception.

For SRXT, the R_1 and R_3 fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (SRXT only)

Programming Note: Infinities produced as DFP results have the reserved field set to zero and have canonical declets in the encoded trailing-significand field. An infinity is not considered canonical unless all digits in the trailing significand are zeros.

SUBTRACT

Mnemonic1 R_1, R_2, R_3 [RRF-a]

Op Code	R_3	////	R_1	R_2
0	16	20	24	28 31

Mnemonic1	Op Code	Operands
SDTR	'B3D3'	Long DFP
SXTR	'B3DB'	Extended DFP

Mnemonic2 R_1, R_2, R_3, M_4 [RRF-a]

Op Code	R_3	M_4	R_1	R_2
0	16	20	24	28 31

Mnemonic2	Op Code	Operands
SDTRA	'B3D3'	Long DFP
SXTRA	'B3DB'	Extended DFP

The third operand is subtracted from the second operand, and the difference is placed at the first-operand location.

The execution of SUBTRACT is identical to that of ADD, except that the third operand, if numeric, participates in the operation with its sign bit inverted. When the third operand is a NaN, it participates in the operation with its sign bit unchanged. See Figure 20-11 on page 20-21 for the detailed results of ADD.

When the delivered value is exact, the preferred quantum is the smaller quantum of the two source

operands; when the delivered value is inexact, the preferred quantum is the smallest quantum.

The result placed at the first-operand location is canonical.

For SXTR and SXTRA, the R fields must designate valid floating-point-register pairs; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Result is a NaN

IEEE Exceptions:

- Invalid operation
- Overflow
- Underflow
- Inexact
- Quantum (if the floating-point extension facility is installed)

Program Exceptions:

- Data with DXC 3, DFP instruction
- Data with DXC for IEEE exception
- Operation (if the DFP facility is not installed)
- Specification (SXTR and SXTRA only)

Programming Note: See the programming notes for ADD.

TEST DATA CLASS

Mnemonic $R_1, D_2(X_2, B_2)$ [RXE]

OpCode	R_1	X_2	B_2	D_2	////////	OpCode
0	8	12	16	20	32	40 47

Mnemonic	Op Code	Operands
TDCET	'ED50'	Short DFP
TDCDT	'ED54'	Long DFP
TDCXT	'ED58'	Extended DFP

The class and sign of the first operand are examined to select one bit from the second-operand address. Condition code 0 or 1 is set according to whether the selected bit is zero or one, respectively.

The second-operand address is not used to address data; instead, the rightmost 12 bits of the address, bits 52-63, are used to specify 12 combinations of data class and sign. Bits 0-51 of the second-operand address are ignored.

As shown in Figure 20-29, DFP operands are divided into six classes: zero, subnormal, normal, infinity, quiet NaN, and signaling NaN:

DFP data class	Bit used when sign is	
	+	-
Zero	52	53
Subnormal	54	55
Normal	56	57
Infinity	58	59
Quiet NaN	60	61
Signaling NaN	62	63

Figure 20-29. Second-Operand-Address Bits for TEST DATA CLASS

One or more of the second-operand-address bits may be set to one. If the second-operand-address bit corresponding to the class and sign of the first operand is one, condition code 1 is set; otherwise, condition code 0 is set.

Operands, including SNaNs and QNaNs, are examined without causing an IEEE exception.

For TDCXT, the R₁ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Selected bit is 0 (no match)
- 1 Selected bit is 1 (match)
- 2 --
- 3 --

IEEE Exceptions: None.

Program Exceptions:

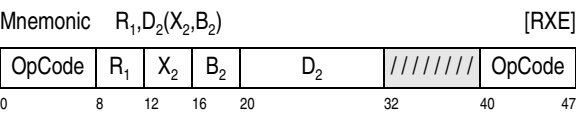
- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (TDCXT only)

Programming Notes:

1. TEST DATA CLASS provides a way to test an operand without risk of an exception or setting the IEEE flags.

2. The bits used to specify the combinations of data class and sign are not all the same for BFP and DFP. Specifically, BFP TEST DATA CLASS uses bits 54 and 55 for normal number, and bits 56 and 57 for subnormal number.

TEST DATA GROUP



Mnemonic	Op Code	Operands
TDGET	'ED51'	Short DFP
TDGDT	'ED55'	Long DFP
TDGXT	'ED59'	Extended DFP

The group and sign of the first operand are examined to select one bit from the second-operand address. Condition code 0 or 1 is set according to whether the selected bit is zero or one, respectively.

The second-operand address is not used to address data; instead, the rightmost 12 bits of the address, bits 52-63, are used to specify 12 combinations of data group and sign. Bits 0-51 of the second-operand address are ignored.

TEST DATA GROUP is used to determine whether a finite number is safe. A finite number is safe if the exponent is neither maximum nor minimum, and the leftmost significand digit is zero.

Figure 20-30 on page 20-53 shows the data groups and the bit assignment. There are six data groups: safe zero, zero with extreme exponent, nonzero with extreme exponent, safe nonzero, nonzero leftmost significand digit with nonextreme exponent, and special. The special group is defined for infinity and NaN. Depending on the model, subnormal with nonextreme exponent may be placed in the nonzero-with-extreme-exponent group or the safe-nonzero group.

One or more of the second-operand-address bits may be set to one. If the second-operand-address bit corresponding to the group and sign of the first operand is one, condition code 1 is set; otherwise, condition code 0 is set.

Operands, including SNaNs and QNaNs, are examined without causing an IEEE exception.

For TDGXT, the R₁ field must designate a valid floating-point-register pair; otherwise, a specification exception is recognized.

Resulting Condition Code:

- 0 Selected bit is 0 (no match)
- 1 Selected bit is 1 (match)
- 2 --
- 3 --

IEEE Exceptions: None.

Program Exceptions:

- Data with DXC 3, DFP instruction
- Operation (if the DFP facility is not installed)
- Specification (TDGXT only)

Programming Notes:

1. TEST DATA GROUP provides a way to test an operand without risk of an exception or setting the IEEE flags.
2. TEST DATA GROUP can be issued after an operation that produces a DFP result to quickly determine if the result is safe. For DFP results that are finite numbers, the result is safe if using a wider data format by the operation would have produced the same value and quantum. A safe result has two important characteristics: (1) the exponent is neither the maximum exponent nor the minimum exponent, and (2) the leftmost significand digit is zero.
3. TEST DATA GROUP may be used to test whether a nonzero finite number is safe by setting bits 58 and 59 of the second-operand address to ones.
4. TEST DATA GROUP may be used to test whether a nonzero finite number has reached the limit of the format precision but not the limit of the format range by setting bits 60 and 61 of the second-operand address to ones.
5. Subnormal with nonextreme exponent may be grouped with either the nonzero-with-extreme-exponent group or the safe-nonzero group. The

program shouldn't depend on which group subnormal with nonextreme exponent is in.

DFP Operand	Exponent	LMD	Data Group	Bit used when sign is	
				+	-
Zero	Nonextreme	z ¹	Safe zero	52	53
	Extreme	z ¹	Zero with extreme exponent	54	55
Nonzero finite	Extreme	--	Nonzero with extreme exponent	56	57
	Nonextreme	z	Safe nonzero	58	59
	Nonextreme	nz	Nonzero leftmost significand digit with nonextreme exponent	60	61
Infinity or NaN	na	na	Special	62	63

Explanation:

-- The result does not depend on this condition.

1 This condition is true by virtue of the condition to the left of this column.

Extreme Maximum right-units-view (RUV) exponent, Q_{max}, or minimum right-units-view (RUV) exponent, Q_{min}.

Nonextreme Q_{max} < right-units-view (RUV) exponent < Q_{min}.

LMD Leftmost significand digit.

na Not applicable.

nz Nonzero.

z Zero.

Figure 20-30. Second-Operand-Address Bits for TEST DATA GROUP

Densely Packed Decimal (DPD)

Decimal-to-DPD Mapping

The mapping of a 3-digit decimal number (000 - 999) to a 10-bit value, called a declet is shown in Figure 20-31 on page 20-55. The DPD entries are shown in hexadecimal. The first two digits of the decimal number are shown in the leftmost column and the third digit along the top row. The table is split into two halves, with the right half being a continuation of the left half.

DPD-to-Decimal Mapping

The mapping of the 10-bit declet to a 3-digit decimal number is shown in Figure 20-32 on page 20-56. The 10-bit declet value is split into a 6-bit index shown in the left column and a 4-bit index shown along the top row, both represented in hexadecimal. The values

marked with an asterisk are results mapped from noncanonical declets and are explained further in the following section.

	0	1	2	3	4	5	6	7	8	9
00_	000	001	002	003	004	005	006	007	008	009
01_	010	011	012	013	014	015	016	017	018	019
02_	020	021	022	023	024	025	026	027	028	029
03_	030	031	032	033	034	035	036	037	038	039
04_	040	041	042	043	044	045	046	047	048	049
05_	050	051	052	053	054	055	056	057	058	059
06_	060	061	062	063	064	065	066	067	068	069
07_	070	071	072	073	074	075	076	077	078	079
08_	00A	00B	02A	02B	04A	04B	06A	06B	04E	04F
09_	01A	01B	03A	03B	05A	05B	07A	07B	05E	05F
10_	080	081	082	083	084	085	086	087	088	089
11_	090	091	092	093	094	095	096	097	098	099
12_	0A0	0A1	0A2	0A3	0A4	0A5	0A6	0A7	0A8	0A9
13_	0B0	0B1	0B2	0B3	0B4	0B5	0B6	0B7	0B8	0B9
14_	0C0	0C1	0C2	0C3	0C4	0C5	0C6	0C7	0C8	0C9
15_	0D0	0D1	0D2	0D3	0D4	0D5	0D6	0D7	0D8	0D9
16_	0E0	0E1	0E2	0E3	0E4	0E5	0E6	0E7	0E8	0E9
17_	0F0	0F1	0F2	0F3	0F4	0F5	0F6	0F7	0F8	0F9
18_	08A	08B	0AA	0AB	0CA	0CB	0EA	0EB	0CE	0CF
19_	09A	09B	0BA	0BB	0DA	0DB	0FA	0FB	0DE	0DF
20_	100	101	102	103	104	105	106	107	108	109
21_	110	111	112	113	114	115	116	117	118	119
22_	120	121	122	123	124	125	126	127	128	129
23_	130	131	132	133	134	135	136	137	138	139
24_	140	141	142	143	144	145	146	147	148	149
25_	150	151	152	153	154	155	156	157	158	159
26_	160	161	162	163	164	165	166	167	168	169
27_	170	171	172	173	174	175	176	177	178	179
28_	10A	10B	12A	12B	14A	14B	16A	16B	14E	14F
29_	11A	11B	13A	13B	15A	15B	17A	17B	15E	15F
30_	180	181	182	183	184	185	186	187	188	189
31_	190	191	192	193	194	195	196	197	198	199
32_	1A0	1A1	1A2	1A3	1A4	1A5	1A6	1A7	1A8	1A9
33_	1B0	1B1	1B2	1B3	1B4	1B5	1B6	1B7	1B8	1B9
34_	1C0	1C1	1C2	1C3	1C4	1C5	1C6	1C7	1C8	1C9
35_	1D0	1D1	1D2	1D3	1D4	1D5	1D6	1D7	1D8	1D9
36_	1E0	1E1	1E2	1E3	1E4	1E5	1E6	1E7	1E8	1E9
37_	1F0	1F1	1F2	1F3	1F4	1F5	1F6	1F7	1F8	1F9
38_	18A	18B	1AA	1AB	1CA	1CB	1EA	1EB	1CE	1CF
39_	19A	19B	1BA	1BB	1DA	1DB	1FA	1FB	1DE	1DF
40_	200	201	202	203	204	205	206	207	208	209
41_	210	211	212	213	214	215	216	217	218	219
42_	220	221	222	223	224	225	226	227	228	229
43_	230	231	232	233	234	235	236	237	238	239
44_	240	241	242	243	244	245	246	247	248	249
45_	250	251	252	253	254	255	256	257	258	259
46_	260	261	262	263	264	265	266	267	268	269
47_	270	271	272	273	274	275	276	277	278	279
48_	20A	20B	22A	22B	24A	24B	26A	26B	24E	24F
49_	21A	21B	23A	23B	25A	25B	27A	27B	25E	25F

	0	1	2	3	4	5	6	7	8	9
50_	280	281	282	283	284	285	286	287	288	289
51_	290	291	292	293	294	295	296	297	298	299
52_	2A0	2A1	2A2	2A3	2A4	2A5	2A6	2A7	2A8	2A9
53_	2B0	2B1	2B2	2B3	2B4	2B5	2B6	2B7	2B8	2B9
54_	2C0	2C1	2C2	2C3	2C4	2C5	2C6	2C7	2C8	2C9
55_	2D0	2D1	2D2	2D3	2D4	2D5	2D6	2D7	2D8	2D9
56_	2E0	2E1	2E2	2E3	2E4	2E5	2E6	2E7	2E8	2E9
57_	2F0	2F1	2F2	2F3	2F4	2F5	2F6	2F7	2F8	2F9
58_	28A	28B	2AA	2AB	2CA	2CB	2EA	2EB	2CE	2CF
59_	29A	29B	2BA	2BB	2DA	2DB	2FA	2FB	2DE	2DF
60_	300	301	302	303	304	305	306	307	308	309
61_	310	311	312	313	314	315	316	317	318	319
62_	320	321	322	323	324	325	326	327	328	329
63_	330	331	332	333	334	335	336	337	338	339
64_	340	341	342	343	344	345	346	347	348	349
65_	350	351	352	353	354	355	356	357	358	359
66_	360	361	362	363	364	365	366	367	368	369
67_	370	371	372	373	374	375	376	377	378	379
68_	30A	30B	32A	32B	34A	34B	36A	36B	34E	34F
69_	31A	31B	33A	33B	35A	35B	37A	37B	35E	35F
70_	380	381	382	383	384	385	386	387	388	389
71_	390	391	392	393	394	395	396	397	398	399
72_	3A0	3A1	3A2	3A3	3A4	3A5	3A6	3A7	3A8	3A9
73_	3B0	3B1	3B2	3B3	3B4	3B5	3B6	3B7	3B8	3B9
74_	3C0	3C1	3C2	3C3	3C4	3C5	3C6	3C7	3C8	3C9
75_	3D0	3D1	3D2	3D3	3D4	3D5	3D6	3D7	3D8	3D9
76_	3E0	3E1	3E2	3E3	3E4	3E5	3E6	3E7	3E8	3E9
77_	3F0	3F1	3F2	3F3	3F4	3F5	3F6	3F7	3F8	3F9
78_	38A	38B	3AA	3AB	3CA	3CB	3EA	3EB	3CE	3CF
79_	39A	39B	3BA	3BB	3DA	3DB	3FA	3FB	3DE	3DF
80_	00C	00D	10C	10D	20C	20D	30C	30D	02E	02F
81_	01C	01D	11C	11D	21C	21D	31C	31D	03E	03F
82_	02C	02D	12C	12D	22C	22D	32C	32D	12E	12F
83_	03C	03D	13C	13D	23C	23D	33C	33D	13E	13F
84_	04C	04D	14C	14D	24C	24D	34C	34D	22E	22F
85_	05C	05D	15C	15D	25C	25D	35C	35D	23E	23F
86_	06C	06D	16C	16D	26C	26D	36C	36D	32E	32F
87_	07C	07D	17C	17D	27C	27D	37C	37D	33E	33F
88_	00E	00F	10E	10F	20E	20F	30E	30F	06E	06F
89_	01E	01F	11E	11F	21E	21F	31E	31F	07E	07F
90_	08C	08D	18C	18D	28C	28D	38C	38D	0AE	0AF
91_	09C	09D	19C	19D	29C	29D	39C	39D	0BE	0BF
92_	0AC	0AD	1AC	1AD	2AC	2AD	3AC	3AD	1AE	1AF
93_	0BC	0BD	1BC	1BD	2BC	2BD	3BC	3BD	1BE	1BF
94_	0CC	0CD	1CC	1CD	2CC	2CD	3CC	3CD	2AE	2AF
95_	0DC	0DD	1DC	1DD	2DC	2DD	3DC	3DD	2BE	2BF
96_	0EC	0ED	1EC	1ED	2EC	2ED	3EC	3ED	3AE	3AF
97_	0FC	0FD	1FC	1FD	2FC	2FD	3FC	3FD	3BE	3BF
98_	08E	08F	18E	18F	28E	28F	38E	38F	0EE	0EF
99_	09E	09F	19E	19F	29E	29F	39E	39F	0FE	0FF

Figure 20-31. Decimal-to-DPD Mapping

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00_	000	001	002	003	004	005	006	007	008	009	080	081	800	801	880	881
01_	010	011	012	013	014	015	016	017	018	019	090	091	810	811	890	891
02_	020	021	022	023	024	025	026	027	028	029	082	083	820	821	808	809
03_	030	031	032	033	034	035	036	037	038	039	092	093	830	831	818	819
04_	040	041	042	043	044	045	046	047	048	049	084	085	840	841	088	089
05_	050	051	052	053	054	055	056	057	058	059	094	095	850	851	098	099
06_	060	061	062	063	064	065	066	067	068	069	086	087	860	861	888	889
07_	070	071	072	073	074	075	076	077	078	079	096	097	870	871	898	899
08_	100	101	102	103	104	105	106	107	108	109	180	181	900	901	980	981
09_	110	111	112	113	114	115	116	117	118	119	190	191	910	911	990	991
0A_	120	121	122	123	124	125	126	127	128	129	182	183	920	921	908	909
0B_	130	131	132	133	134	135	136	137	138	139	192	193	930	931	918	919
0C_	140	141	142	143	144	145	146	147	148	149	184	185	940	941	188	189
0D_	150	151	152	153	154	155	156	157	158	159	194	195	950	951	198	199
0E_	160	161	162	163	164	165	166	167	168	169	186	187	960	961	988	989
0F_	170	171	172	173	174	175	176	177	178	179	196	197	970	971	998	999
10_	200	201	202	203	204	205	206	207	208	209	280	281	802	803	882	883
11_	210	211	212	213	214	215	216	217	218	219	290	291	812	813	892	893
12_	220	221	222	223	224	225	226	227	228	229	282	283	822	823	828	829
13_	230	231	232	233	234	235	236	237	238	239	292	293	832	833	838	839
14_	240	241	242	243	244	245	246	247	248	249	284	285	842	843	288	289
15_	250	251	252	253	254	255	256	257	258	259	294	295	852	853	298	299
16_	260	261	262	263	264	265	266	267	268	269	286	287	862	863	888*	889*
17_	270	271	272	273	274	275	276	277	278	279	296	297	872	873	898*	899*
18_	300	301	302	303	304	305	306	307	308	309	380	381	902	903	982	983
19_	310	311	312	313	314	315	316	317	318	319	390	391	912	913	992	993
1A_	320	321	322	323	324	325	326	327	328	329	382	383	922	923	928	929
1B_	330	331	332	333	334	335	336	337	338	339	392	393	932	933	938	939
1C_	340	341	342	343	344	345	346	347	348	349	384	385	942	943	388	389
1D_	350	351	352	353	354	355	356	357	358	359	394	395	952	953	398	399
1E_	360	361	362	363	364	365	366	367	368	369	386	387	962	963	988*	989*
1F_	370	371	372	373	374	375	376	377	378	379	396	397	972	973	998*	999*
20_	400	401	402	403	404	405	406	407	408	409	480	481	804	805	884	885
21_	410	411	412	413	414	415	416	417	418	419	490	491	814	815	894	895
22_	420	421	422	423	424	425	426	427	428	429	482	483	824	825	848	849
23_	430	431	432	433	434	435	436	437	438	439	492	493	834	835	858	859
24_	440	441	442	443	444	445	446	447	448	449	484	485	844	845	488	489
25_	450	451	452	453	454	455	456	457	458	459	494	495	854	855	498	499
26_	460	461	462	463	464	465	466	467	468	469	486	487	864	865	888*	889*
27_	470	471	472	473	474	475	476	477	478	479	496	497	874	875	898*	899*
28_	500	501	502	503	504	505	506	507	508	509	580	581	904	905	984	985
29_	510	511	512	513	514	515	516	517	518	519	590	591	914	915	994	995
2A_	520	521	522	523	524	525	526	527	528	529	582	583	924	925	948	949
2B_	530	531	532	533	534	535	536	537	538	539	592	593	934	935	958	959
2C_	540	541	542	543	544	545	546	547	548	549	584	585	944	945	588	589
2D_	550	551	552	553	554	555	556	557	558	559	594	595	954	955	598	599
2E_	560	561	562	563	564	565	566	567	568	569	586	587	964	965	988*	989*
2F_	570	571	572	573	574	575	576	577	578	579	596	597	974	975	998*	999*
30_	600	601	602	603	604	605	606	607	608	609	680	681	806	807	886	887
31_	610	611	612	613	614	615	616	617	618	619	690	691	816	817	896	897
32_	620	621	622	623	624	625	626	627	628	629	682	683	826	827	868	869
33_	630	631	632	633	634	635	636	637	638	639	692	693	836	837	878	879
34_	640	641	642	643	644	645	646	647	648	649	684	685	846	847	688	689
35_	650	651	652	653	654	655	656	657	658	659	694	695	856	857	698	699
36_	660	661	662	663	664	665	666	667	668	669	686	687	866	867	888*	889*
37_	670	671	672	673	674	675	676	677	678	679	696	697	876	877	898*	899*
38_	700	701	702	703	704	705	706	707	708	709	780	781	906	907	986	987
39_	710	711	712	713	714	715	716	717	718	719	790	791	916	917	996	997
3A_	720	721	722	723	724	725	726	727	728	729	782	783	926	927	968	969
3B_	730	731	732	733	734	735	736	737	738	739	792	793	936	937	978	979
3C_	740	741	742	743	744	745	746	747	748	749	784	785	946	947	788	789
3D_	750	751	752	753	754	755	756	757	758	759	794	795	956	957	798	799
3E_	760	761	762	763	764	765	766	767	768	769	786	787	966	967	988*	989*
3F_	770	771	772	773	774	775	776	777	778	779	796	797	976	977	998*	999*

* Result mapped from a noncanonical declct.

Figure 20-32. DPD-to-Decimal Mapping

Appendix A. Number Representation and Instruction-Use Examples

Number Representation	A-2	FIND LEFTMOST ONE (FLOGR)	A-22
Binary Integers	A-2	INSERT CHARACTERS UNDER MASK	
Signed Binary Integers	A-2	(ICM)	A-23
Unsigned Binary Integers	A-3	LOAD (L, LR)	A-23
Decimal Integers	A-4	LOAD ADDRESS (LA)	A-23
Hexadecimal-Floating-Point Numbers	A-5	LOAD HALFWORD (LH)	A-24
Conversion Example	A-6	MOVE (MVC, MVI)	A-24
Instruction-Use Examples	A-7	MVC Example	A-24
Machine Format	A-7	MVI Example	A-25
Assembler-Language Format	A-7	MOVE INVERSE (MVCIN)	A-25
Addressing Mode in Examples	A-7	MOVE LONG (MVCL)	A-26
General Instructions	A-7	MOVE NUMERICS (MVN)	A-26
ADD HALFWORD (AH)	A-8	MOVE STRING (MVST)	A-27
AND (N, NC, NI, NR)	A-8	MOVE WITH OFFSET (MVO)	A-27
NI Example	A-8	MOVE ZONES (MVZ)	A-28
Linkage Instructions (BAL, BALR, BAS, BASR,		MULTIPLY (M, MR)	A-28
BASSM, BSM)	A-8	MULTIPLY HALFWORD (MH)	A-29
Other BALR and BASR Examples	A-10	OR (O, OC, OI, OR)	A-29
BRANCH AND STACK (BAKR)	A-10	OI Example	A-29
BAKR Example 1	A-10	PACK (PACK)	A-29
BAKR Example 2	A-11	ROTATE THEN EXCLUSIVE OR SELECTED	
BAKR Example 3	A-11	BITS	A-30
BRANCH ON CONDITION (BC, BCR)	A-12	ROTATE THEN INSERT SELECTED BITS	A-30
BRANCH ON COUNT (BCT, BCTR)	A-12	ROTATE THEN OR SELECTED BITS	A-30
BRANCH ON INDEX HIGH (BXH)	A-12	SEARCH STRING (SRST)	A-31
BXH Example 1	A-12	SRST Example 1	A-31
BXH Example 2	A-13	SRST Example 2	A-31
BRANCH ON INDEX LOW OR EQUAL		SHIFT LEFT DOUBLE (SLDA)	A-31
(BXLE)	A-13	SHIFT LEFT SINGLE (SLA)	A-32
BXLE Example 1	A-14	STORE CHARACTERS UNDER MASK	
BXLE Example 2	A-14	(STCM)	A-32
COMPARE AND FORM CODEWORD (CFC)	A-14	STORE MULTIPLE (STM)	A-32
COMPARE HALFWORD (CH)	A-14	TEST UNDER MASK (TM)	A-33
COMPARE LOGICAL (CL, CLC, CLI, CLR)	A-15	TRANSLATE (TR)	A-33
CLC Example	A-15	TRANSLATE AND TEST (TRT)	A-34
CLI Example	A-15	UNPACK (UNPK)	A-35
CLR Example	A-15	UPDATE TREE (UPT)	A-36
COMPARE LOGICAL CHARACTERS UNDER		Decimal Instructions	A-36
MASK (CLM)	A-16	ADD DECIMAL (AP)	A-36
COMPARE LOGICAL LONG (CLCL)	A-16	COMPARE DECIMAL (CP)	A-36
COMPARE LOGICAL STRING (CLST)	A-18	DIVIDE DECIMAL (DP)	A-37
CONVERT TO BINARY (CVB)	A-18	EDIT (ED)	A-37
CONVERT TO DECIMAL (CVD)	A-19	EDIT AND MARK (EDMK)	A-38
DIVIDE (D, DR)	A-19	MULTIPLY DECIMAL (MP)	A-39
EXCLUSIVE OR (X, XC, XI, XR)	A-20	SHIFT AND ROUND DECIMAL (SRP)	A-39
XC Example	A-20	Decimal Left Shift	A-39
XI Example	A-21	Decimal Right Shift	A-39
EXECUTE (EX)	A-21	Decimal Right Shift and Round	A-40

Multiplying by a Variable Power of 10	A-40
ZERO AND ADD (ZAP)	A-41
Hexadecimal-Floating-Point Instructions	A-41
ADD NORMALIZED (AD, ADR, AE, AER, AXR).	A-41
ADD UNNORMALIZED (AU, AUR, AW, AWR)	A-42
COMPARE (CD, CDR, CE, CER)	A-42
DIVIDE (DD, DDR, DE, DER)	A-43
HALVE (HDR, HER).	A-43
MULTIPLY (MD, MDR, MDE, MDER, MXD, MXDR, MXR)	A-43
Hexadecimal-Floating-Point-Number Conversion.	A-44
Fixed Point to Hexadecimal Floating Point. A-44	
Hexadecimal Floating Point to Fixed Point. A-44	
Multiprogramming and Multiprocessing Examples	A-45

Example of a Program Failure Using OR Immediate	A-45
Conditional Swapping Instructions (CS, CDS) A-46	
Setting a Single Bit	A-46
Updating Counters.	A-47
Bypassing Post and Wait	A-47
Lock/Unlock.	A-47
Lock/Unlock with LIFO Queuing for Contentions	A-48
Lock/Unlock with FIFO Queuing for Contentions	A-49
Free-Pool Manipulation	A-50
PERFORM LOCKED OPERATION (PLO) . .	A-51
Sorting Instructions	A-53
Tree Format.	A-53
Example of Use of Sort Instructions	A-54

This appendix (except for the examples for FIND LEFTMOST ONE, ROTATE THEN EXCLUSIVE OR SELECTED BITS, ROTATE THEN INSERT SELECTED BITS, ROTATE THEN OR SELECTED BITS, and changes in the terminology for signed-packed decimal and unsigned-packed decimal) is the same as in *Enterprise Systems Architecture/390 Principles of Operation*, SA22-7201; it has not been revised to show the enlargement of register sizes or the 64-bit addressing mode.

Number Representation

Binary Integers

Signed Binary Integers

Signed binary integers are most commonly represented as halfwords (16 bits) or words (32 bits). In both lengths, the leftmost bit (bit 0) is the sign of the number. The remaining bits (bits 1-15 for halfwords and 1-31 for words) are used to specify the magnitude of the number. Binary integers are also referred to as fixed-point numbers, because the radix point (binary point) is considered to be fixed at the right, and any scaling is done by the programmer.

Positive binary integers are in true binary notation with a zero sign bit. Negative binary integers are in two's-complement notation with a one bit in the sign

position. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit (that is, all zeros for positive numbers, all ones for negative numbers).

Negative binary integers are formed in two's-complement notation by inverting each bit of the positive binary integer and adding one. As an example using the halfword format, the binary number with the decimal value +26 is made negative (-26) in the following manner:

```
+26      0 000 0000 0001 1010
Invert 1 111 1111 1110 0101
Add 1                      1
-----
-26      1 111 1111 1110 0110 (Two's complement form)

(S is the sign bit.)
```

This is equivalent to subtracting the number:

```
      00000000 00011010
from
1 00000000 00000000
```

Negative binary integers are changed to positive in the same manner.

The following addition examples illustrate two's-complement arithmetic and overflow conditions. Only eight bit positions are used.

1. $+57 = 0011\ 1001$
 $+35 = 0010\ 0011$

 $+92 = 0101\ 1100$
2. $+57 = 0011\ 1001$
 $-35 = 1101\ 1101$

 $+22 = 0001\ 0110$ No overflow – carry into leftmost position and carry out
3. $+35 = 0010\ 0011$
 $-57 = 1100\ 0111$

 $-22 = 1110\ 1010$ Sign change only – no carry into leftmost position and no carry out
4. $-57 = 1100\ 0111$
 $-35 = 1101\ 1101$

 $-92 = 1010\ 0100$ No overflow – carry into leftmost position and carry out
5. $+57 = 0011\ 1001$
 $+92 = 0101\ 1100$

 $+149 = *1001\ 0101$ *Overflow – carry into leftmost position, no carry out
6. $-57 = 1100\ 0111$
 $-92 = 1010\ 0100$

 $-149 = *0110\ 1011$ *Overflow – no carry into leftmost position but carry out

The presence or absence of an overflow condition may be recognized from the carries:

- There is no overflow:
 1. If there is no carry into the leftmost bit position and no carry out (examples 1 and 3).
 2. If there is a carry into the leftmost position and also a carry out (examples 2 and 4).
- There is an overflow:
 1. If there is a carry into the leftmost position but no carry out (example 5).
 2. If there is no carry into the leftmost position but there is a carry out (example 6).

The following are 16-bit signed binary integers. The first is the maximum positive 16-bit binary integer. The last is the maximum negative 16-bit binary integer (the negative 16-bit binary integer with the greatest absolute value).

$$\begin{aligned}
 2^{15}-1 &= 32,767 = 0\ 111\ 1111\ 1111\ 1111 \\
 2^0 &= 1 = 0\ 000\ 0000\ 0000\ 0001 \\
 0 &= 0 = 0\ 000\ 0000\ 0000\ 0000 \\
 -2^0 &= -1 = 1\ 111\ 1111\ 1111\ 1111 \\
 -2^{15} &= -32,768 = 1\ 000\ 0000\ 0000\ 0000
 \end{aligned}$$

Figure A-1 illustrates several 32-bit signed binary integers arranged in descending order. The first is the maximum positive binary integer that can be represented by 32 bits, and the last is the maximum negative binary integer that can be represented by 32 bits.

$2^{31}-1$	= 2 147 483 647	= 0 111 1111 1111 1111 1111 1111 1111
2^{16}	= 65 536	= 0 000 0000 0000 0001 0000 0000 0000
2^0	= 1	= 0 000 0000 0000 0000 0000 0000 0001
0	= 0	= 0 000 0000 0000 0000 0000 0000 0000
-2^0	= -1	= 1 111 1111 1111 1111 1111 1111 1111
-2^1	= -2	= 1 111 1111 1111 1111 1111 1111 1110
-2^{16}	= -65 536	= 1 111 1111 1111 1111 0000 0000 0000
$-2^{31}+1$	= -2 147 483 647	= 1 000 0000 0000 0000 0000 0000 0001
-2^{31}	= -2 147 483 648	= 1 000 0000 0000 0000 0000 0000 0000

Figure A-1. 32-Bit Signed Binary Integers

Unsigned Binary Integers

Certain instructions, such as ADD LOGICAL, treat

binary integers as unsigned rather than signed. Unsigned binary integers have the same format as

signed binary integers, except that the leftmost bit is interpreted as another numeric bit rather than a sign bit. There is no complement notation because all unsigned binary integers are considered positive.

The following examples illustrate the addition of unsigned binary integers. Only eight bit positions are used. The examples are numbered the same as the corresponding examples for signed binary integers.

$$\begin{array}{r} 1. \quad 57 = 0011 \ 1001 \\ \quad 35 = 0010 \ 0011 \\ \hline \quad 92 = 0101 \ 1100 \end{array}$$

$$\begin{array}{r} 2. \quad 57 = 0011 \ 1001 \\ \quad 221 = 1101 \ 1101 \\ \hline \quad 278 = *0001 \ 0110 \text{ *Carry out of leftmost position} \end{array}$$

$$\begin{array}{r} 3. \quad 35 = 0010 \ 0011 \\ \quad 199 = 1100 \ 0111 \\ \hline \quad 234 = 1110 \ 1010 \end{array}$$

$2^{32}-1$	= 4 294 967 295	= 1111 1111 1111 1111 1111 1111 1111 1111
2^{31}	= 2 147 483 648	= 1000 0000 0000 0000 0000 0000 0000 0000
$2^{31}-1$	= 2 147 483 647	= 0111 1111 1111 1111 1111 1111 1111 1111
2^{16}	= 65 536	= 0000 0000 0000 0001 0000 0000 0000 0000
2^0	= 1	= 0000 0000 0000 0000 0000 0000 0000 0001
0	= 0	= 0000 0000 0000 0000 0000 0000 0000 0000

Figure A-2. 32-Bit Unsigned Binary Integers

Decimal Integers

Decimal integers consist of one or more decimal digits and a sign. Each digit and the sign are represented by a 4-bit code. The decimal digits are in binary-coded decimal (BCD) form, with the values 0-9 encoded as 0000-1001. The sign is usually represented as 1100 (C hex) for plus and 1101 (D hex) for minus. These are the preferred sign codes, which are generated by the machine for the results of decimal-arithmetic operations. There are also several alternate sign codes (1010, 1110, and 1111 for plus; 1011 for minus). The alternate sign codes are accepted by the machine as valid in source operands but are not generated for results.

Decimal integers may have different lengths, from one to 16 bytes. Of the many decimal formats, only

$$\begin{array}{r} 4. \quad 199 = 1100 \ 0111 \\ \quad 221 = 1101 \ 1101 \\ \hline \quad 420 = *1010 \ 0100 \text{ *Carry out of leftmost position} \\ \\ 5. \quad 57 = 0011 \ 1001 \\ \quad 92 = 0101 \ 1100 \\ \hline \quad 149 = 1001 \ 0101 \\ \\ 6. \quad 199 = 1100 \ 0111 \\ \quad 164 = 1010 \ 0100 \\ \hline \quad 363 = *0110 \ 1011 \text{ *Carry out of leftmost position} \end{array}$$

A carry out of the leftmost bit position may or may not imply an overflow, depending on the application.

Figure A-2 illustrates several 32-bit unsigned binary integers arranged in descending order.

two will be considered here: signed-packed-decimal and zoned. In the signed-packed-decimal format, each byte contains two decimal digits, except for the rightmost byte, which contains the sign code in the right half. For decimal arithmetic, the number of decimal digits in the signed-packed-decimal format can vary from one to 31. Because decimal integers must consist of whole bytes and there must be a sign code on the right, the number of decimal digits in this format is always odd. If an even number of significant digits is desired, a leading zero must be inserted on the left.

In the zoned format, each byte consists of a decimal digit on the right and the zone code 1111 (F hex) on the left, except for the rightmost byte where the sign code replaces the zone code. Thus, a decimal integer in the zoned format can have from one to 16 digits. The zoned format may be used directly for input

and output in the extended binary-coded-decimal interchange code (EBCDIC), except that the sign must be separated from the rightmost digit and handled as a separate character. For positive (unsigned) numbers, however, the sign can simply be represented by the zone code of the rightmost digit because the zone code is one of the acceptable alternate codes for plus.

In either format, negative decimal integers are represented in true notation with a separate sign. As for binary integers, the radix point (decimal point) of decimal integers is considered to be fixed at the right, and any scaling is done by the programmer.

The following are some examples of decimal integers shown in hexadecimal notation:

Decimal Value	Signed Packed Format	Zoned Format
+123	12 3C or 12 3F	F1 F2 C3 or F1 F2 F3
-4321	04 32 1D	F4 F3 F2 D1
+000050	00 00 05 0C or 00 00 05 0F	F0 F0 F0 F0 F5 C0 or F0 F0 F0 F0 F5 F0
-7	7D	D7
00000	00 00 0C or 00 00 0F	F0 F0 F0 F0 C0 or F0 F0 F0 F0 F0

Under some circumstances, a zero with a minus sign (negative zero) is produced. For example, the multiplicand:

00 12 3D (-123)

times the multiplier:

0C (+0)

generates the product:

00 00 0D (-0)

because the product sign follows the algebraic rule of signs even when the value is zero. A negative zero,

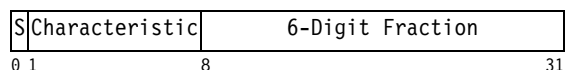
however, is equivalent to a positive zero in that they compare equal in a decimal comparison.

Hexadecimal-Floating-Point Numbers

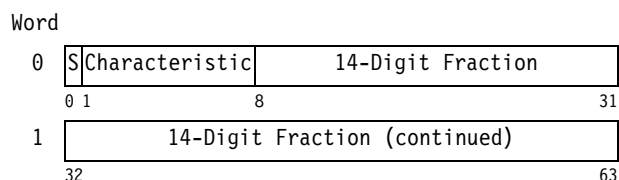
A hexadecimal-floating-point (HFP) number is expressed as a hexadecimal fraction multiplied by a separate power of 16. The term floating point indicates that the placement, of the radix (hexadecimal) point, or scaling, is automatically maintained by the machine.

The part of an HFP number which represents the significant digits of the number is called the fraction. A second part specifies the power (exponent) to which 16 is raised and indicates the location of the radix point of the number. The fraction and exponent may be represented by 32 bits (short format), 64 bits (long format), or 128 bits (extended format).

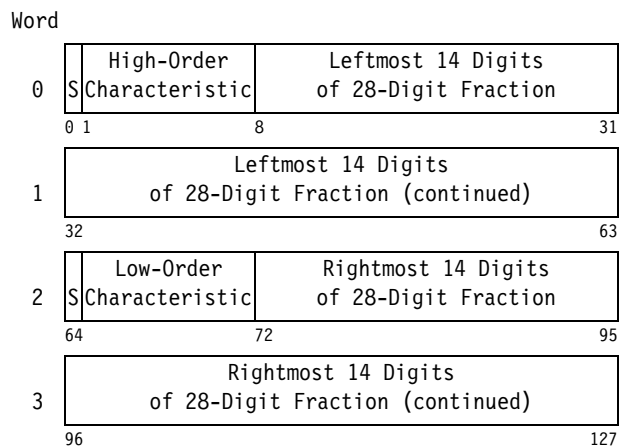
Short HFP Number



Long HFP Number



Extended HFP Number



An HFP number has two signs: one for the fraction and one for the exponent. The fraction sign, which is also the sign of the entire number, is the leftmost bit of each format (0 for plus, 1 for minus). The numeric part of the fraction is in true notation regardless of the sign. The numeric part is contained in bits 8-31 for the short format, in bits 8-63 for the long format, and in bits 8-63 followed by bits 72-127 for the extended format.

The exponent sign is obtained by expressing the exponent in excess-64 notation; that is, the exponent is added as a signed number to 64. The resulting number is called the characteristic. It is located in bits 1-7 for all formats. The characteristic can vary from 0 to 127, permitting the exponent to vary from -64 through 0 to +63. This provides a scale multiplier in the range of 16^{-64} to 16^{+63} . A nonzero fraction, if normalized, has a value less than one and greater than or equal to 1/16, so that the range covered by the magnitude M of a normalized floating-point number is:

$$16^{-65} \leq M < 16^{63}$$

In decimal terms:

$$16^{-65} \text{ is approximately } 5.4 \times 10^{-79}$$

$$16^{63} \text{ is approximately } 7.2 \times 10^{75}$$

1.0	= +1/16x16 ¹	= 0 100 0001 0001 0000 0000 0000 0000 0000 ₂
0.5	= +8/16x16 ⁰	= 0 100 0000 1000 0000 0000 0000 0000 0000 ₂
1/64	= +4/16x16 ⁻¹	= 0 011 1111 0100 0000 0000 0000 0000 0000 ₂
0.0	= +0 x16 ⁻⁶⁴	= 0 000 0000 0000 0000 0000 0000 0000 0000 ₂
-15.0	= -15/16x16 ¹	= 1 100 0001 1111 0000 0000 0000 0000 0000 ₂
5.4x10 ⁻⁷⁹	≈ +1/16x16 ⁻⁶⁴	= 0 000 0000 0001 0000 0000 0000 0000 0000 ₂
7.2x10 ⁷⁵	≈ (1-16 ⁻⁶)x16 ⁶³	= 0 111 1111 1111 1111 1111 1111 1111 1111 ₂

Figure A-3. Normalized Short Hexadecimal-Floating-Point Numbers

Conversion Example

Convert the decimal number 59.25 to a short HFP number. (In another appendix are tables for the conversion of hexadecimal and decimal integers and fractions.)

1. The number is separated into a decimal integer and a decimal fraction.

$$59.25 = 59 \text{ plus } 0.25$$

More precisely,

In the short format:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \leq M \leq (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \leq M \leq (1 - 16^{-28}) \times 16^{63}$$

Within a given fraction length (6, 14, or 28 digits), an HFP operation will provide the greatest precision if the fraction is normalized. A fraction is normalized when the leftmost digit (bit positions 8, 9, 10, and 11) is nonzero. It is unnormalized if the leftmost digit contains all zeros.

If normalization of the operand is desired, the HFP instructions that provide automatic normalization are used. This automatic normalization is accomplished by left-shifting the fraction (four bits per shift) until a nonzero digit occupies the leftmost digit position. The characteristic is reduced by one for each digit shifted.

Figure A-3 illustrates sample normalized short HFP numbers. The last two numbers represent the smallest and the largest positive normalized numbers.

2. The decimal integer is converted to its hexadecimal representation.

$$59_{10} = 3B_{16}$$

3. The decimal fraction is converted to its hexadecimal representation.

$$0.25_{10} = 0.4_{16}$$

4. The integral and fractional parts are combined and expressed as a fraction times a power of 16 (exponent).

$$3B.4_{16} = 0.3B4_{16} \times 16^2$$

- The characteristic is developed from the exponent and converted to binary.

$$\begin{array}{rcl} \text{base} + \text{exponent} & = & \text{characteristic} \\ 64 + 2 & = & 66 = 1000010 \end{array}$$

- The fraction is converted to binary and grouped hexadecimally.

$$.3B4_{16} = .0011\ 1011\ 0100$$

- The characteristic and the fraction are stored in the short format. The sign position contains the sign of the fraction.

S	Char	Fraction
0	1000010	0011 1011 0100 0000 0000 0000

Examples of instruction sequences that may be used to convert between signed binary integers and HFP numbers are shown in “Hexadecimal-Floating-Point-Number Conversion” on page A-44.

Instruction-Use Examples

The following examples illustrate the use of many of the unprivileged instructions. Before studying one of these examples, the reader should consult the instruction description.

The instruction-use examples are written principally for assembler-language programmers, to be used in conjunction with the appropriate assembler-language publications.

Most examples present one particular instruction, both as it is written in an assembler-language statement and as it appears when assembled in storage (machine format).

Machine Format

All machine-format values are given in hexadecimal notation unless otherwise specified. Storage addresses are also given in hexadecimal. Hexadecimal operands are shown converted into binary, decimal, or both if such conversion helps to clarify the example for the reader.

Assembler-Language Format

In assembler-language statements, registers and lengths are presented in decimal. Displacements, immediate operands, and masks may be shown in decimal, hexadecimal, or binary notation; for example, 12, X'C', and B'1100' represent the same value. Whenever the value in a register or storage location is referred to as “not significant,” this value is replaced during the execution of the instruction.

When SS-format instructions are written in the assembler language, lengths are given as the total number of bytes in the field. This differs from the machine definition, in which the length field specifies the number of bytes to be added to the field address to obtain the address of the last byte of the field. Thus, the machine length is one less than the assembler-language length. The assembler program automatically subtracts one from the length specified when the instruction is assembled.

In some of the examples, symbolic addresses are used in order to simplify the examples. In assembler-language statements, a symbolic address is represented as a mnemonic term written in all capitals, such as FLAGS, which may denote the address of a storage location containing data or program-control information. When symbolic addresses are used, the assembler supplies actual base and displacement values according to the programmer's specifications. Therefore, the actual values for base and displacement are not shown in the assembler-language format or in the machine-language format. For assembler-language formats, in the labels that designate instruction fields, the letter “S” is used to indicate the combination of base and displacement fields for an operand address. (For example, S2 represents the combination of B2 and D2.) In the machine-language format, the base and displacement address components are shown as asterisks (****).

Addressing Mode in Examples

Except where otherwise specified, the examples assume the 24-bit addressing mode.

General Instructions

(See Chapter 7, “General Instructions” for a complete description of the general instructions.)

ADD HALFWORD (AH)

The ADD HALFWORD instruction algebraically adds the contents of a two-byte field in storage to the contents of a register. The storage operand is expanded to 32 bits after it is fetched and before it is used in the add operation. The expansion consists in propagating the leftmost (sign) bit 16 positions to the left. For example, assume that the contents of storage locations 2000-2001 are to be added to register 5. Initially:

Register 5 contains 00 00 00 19 = 25_{10} .

Storage locations 2000-2001 contain FF FE = -2_{10} .

Register 12 contains 00 00 18 00.

Register 13 contains 00 00 01 50.

The format of the required instruction is:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
4A	5	D	C	6B0

Assembler Format

Op Code R₁, D₂(X₂, B₂)

AH 5,X'6B0'(13,12)

After the instruction is executed, register 5 contains 00 00 00 17 = 23_{10} . Condition code 2 is set to indicate a result greater than zero.

AND (N, NC, NI, NR)

When the Boolean operator AND is applied to two bits, the result is one when both bits are one; otherwise, the result is zero. When two bytes are ANDed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of ANDing two bytes:

First-operand byte: 0011 0101₂

Second-operand byte: 0101 1100₂

Result byte: 0001 0100₂

NI Example

A frequent use of the AND instruction is to set a particular bit to zero. For example, assume that storage location 4891 contains 0100 0011₂. To set the rightmost bit of this byte to zero without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

Op Code	I ₂	B ₁	D ₁
94	FE	8	001

Assembler Format

Op Code D₁(B₁), I₂

NI 1(8),X'FE'

When this instruction is executed, the byte in storage is ANDed with the immediate byte (the I₂ field of the instruction):

Location 4891: 0100 0011₂

Immediate byte: 1111 1110₂

Result: 0100 0010₂

The resulting byte, with bit 7 set to zero, is stored back in location 4891. Condition code 1 is set.

Linkage Instructions (BAL, BALR, BAS, BASR, BASSM, BSM)

Four unprivileged instructions (BRANCH AND LINK, BRANCH AND SAVE, BRANCH AND SAVE AND SET MODE, and BRANCH AND SET MODE) are available, together with the unconditional branch (BRANCH ON CONDITION with a mask of 15), to provide linkage between subroutines. BRANCH AND LINK (BAL or BALR) is provided primarily for compatibility with programs written for System/370; BRANCH AND SAVE (BAS or BASR) is recommended instead for programs which are to be executed using ESA/370. The instructions BRANCH AND SAVE AND SET MODE (BASSM) and BRANCH AND SET MODE (BSM) provide subroutine linkage together with switching between the 24-bit and the 31-bit addressing modes. The use of these instructions is discussed in a programming note at the end of "Subroutine Linkage without the Linkage Stack." (See also the semiprivileged instruction BRANCH AND STACK.)

The following example compares the operation of these instructions and of the unconditional-branch instruction BRANCH ON CONDITION (BC or BCR with a mask of 15). Assume that each instruction in turn is located at the current instruction address, ready to be executed next. For the first set of examples, the addressing-mode bit, PSW bit 32, is initially zero (24-bit addressing in effect). For the second set, PSW bit 32 is initially one (31-bit addressing). Assume also that general register 5 is to receive the linkage information, and that general register 6 contains the branch address.

The format of the BALR instruction is:

Machine Format

Op Code	R ₁	R ₂
05	5	6

Assembler Format

Op Code	R ₁ , R ₂
BALR	5,6

The other linkage instructions in the RR format have the same format but different op codes:

BASR 0D
BASSM 0C
BSM 0B

For comparison with the RR-format instructions, the results of two RX-format instructions are also shown.

The format of the BAL instruction is:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
45	5	0	6	000

Assembler Format

Op Code	R ₁ , D ₂ (X ₂ , B ₂)
BAL	5,0(0,6)

The BAS instruction has the same format, but the op code is 4D.

The BCR instruction specifies only one register:

Machine Format

Op Code	M ₁	R ₂
07	F	6

Assembler Format

Op Code	M ₁ , R ₂
BCR	15,6

Assume that:

Register 5 contains BB BB BB BB.

Register 6 contains 82 46 8A CE.

PSW bits 32-63 contain

00 00 10 D6 (for 24-bit addressing).
80 00 10 D6 (for 31-bit addressing).

Condition code is 01₂.

Program mask is 1100₂.

The effect of executing each instruction in turn is as follows:

24-Bit Mode Initially

Instruction	Register 5	PSW (32-63)
Before	BB BB BB BB	00 00 10 D6
BCR 15,6	BB BB BB BB	00 46 8A CE
BAL 5,0(0,6)	9C 00 10 DA	00 46 8A CE
BAS 5,0(0,6)	00 00 10 DA	00 46 8A CE
BALR 5,6	5C 00 10 D8	00 46 8A CE
BASR 5,6	00 00 10 D8	00 46 8A CE
BASSM 5,6	00 00 10 D8	82 46 8A CE
BSM 5,6	3B BB BB BB	82 46 8A CE

31-Bit Mode Initially

Instruction	Register 5	PSW (32-63)
Before	BB BB BB BB	80 00 10 D6
BCR 15,6	BB BB BB BB	82 46 8A CE
BAL 5,0(0,6)	80 00 10 DA	82 46 8A CE
BAS 5,0(0,6)	80 00 10 DA	82 46 8A CE
BALR 5,6	80 00 10 D8	82 46 8A CE
BASR 5,6	80 00 10 D8	82 46 8A CE
BASSM 5,6	80 00 10 D8	82 46 8A CE
BSM 5,6	BB BB BB BB	82 46 8A CE

Note that a value of zero in the R₂ field of any of the RR-format instructions indicates that the branching function is not to be performed; it does not refer to register 0. Likewise, a value of zero in the R₁ field of the BSM instruction indicates that the old value of

PSW bit 32 is not to be saved and that register 0 is to be left unchanged. Register 0 can be designated by the R_1 field of instructions BAL, BALR, BAS, BASR, and BASSM, however. In the RX-format branch instructions, branching occurs independent of whether there is a value of zero in the B_2 field or X_2 field of the instruction. However, when the field is zero, instead of using the contents of general register 0, a value of zero is used for that component of address generation.

Programming Note: It should be noted that execution of BAL in the 24-bit addressing mode results in bit 0 of register 5 being set to one. This is because the ILC for an RX-format instruction is 10. This is the only case in which bit zero of the return register does not correctly reflect the addressing mode of the caller. Thus, BSM may be used to return for BALR, BAS, BASR, and BASSM in both the 24-bit and the 31-bit addressing modes, but it cannot be used to return if the program was called by using BAL in the 24-bit addressing mode.

Other BALR and BASR Examples

The BALR or BASR instruction with the R_2 field set to zero may be used to load a register for use as a base register. For example, in the assembler language, the two statements:

```
BALR    15,0
USING   *,15
```

or

```
BASR    15,0
USING   *,15
```

indicate that the address of the next sequential instruction following the BALR or BASR instruction will be placed in register 15, and that the assembler may use register 15 as a base register until otherwise instructed. (The USING statement is an “assembler instruction” and is thus not a part of the object program.)

BRANCH AND STACK (BAKR)

The semiprivileged BRANCH AND STACK instruction facilitates linkage between subroutines by saving status in a linkage-stack state entry (sometimes called a branch state entry to distinguish it from a program-call state entry). When BRANCH AND STACK has been used, the return from the called

program is made by means of the PROGRAM RETURN instruction. PROGRAM RETURN restores access registers 2-14, general registers 2-14, and the PSW with values saved in the state entry, except that it leaves the PER mask unchanged and sets the condition code to an unpredictable value. The use of BRANCH AND STACK is discussed in “Branching Using the Linkage Stack” on page 5-69.

BRANCH AND STACK can be used to perform a calling linkage, or it can be used at or near the entry point of the called program, depending on whether the R_1 field of the instruction is zero or nonzero, respectively. If the R_1 field is zero, bits 32-63 of the PSW saved in the state entry indicate the current addressing mode (24-bit or 31-bit) and the address of the next sequential instruction after the BRANCH AND STACK instruction or an EXECUTE instruction. If the R_1 field is nonzero, bits 32-63 of the PSW saved in the state entry are set with a value generated from the contents of general register R_1 : bit 32 of the PSW is set equal to bit 0 of the register, and bits 1-31 of the PSW are set with an address generated from bits 1-31 of the register under the control of bit 0 of the register. Bits 32-63 of the PSW saved in the state entry are referred to in the following examples as the return value.

The branch address for the instruction is generated from the contents of general register R_2 under the control of the current addressing mode. Bit 0 of general register R_2 does not affect the operation. If the R_2 field of the instruction is zero, the operation is performed without branching.

In addition to saving a complete PSW (except with an unpredictable PER mask) in the state entry, BRANCH AND STACK saves the new value of bits 32-63 of the current PSW in the state entry. Bits 32-63 are referred to in the following examples as the branch value.

The following examples contain cases in which bit 32 of the current PSW is either zero or one (24-bit or 31-bit addressing) before BRANCH AND STACK is executed and in which bit 0 of the general register designated by a nonzero R_1 or R_2 field is either zero or one.

BAKR Example 1

This example shows BAKR used in a calling program. BAKR performs a branch, and the return is to be to the next sequential instruction.

The format of the BAKR instruction is:

Machine Format

Op Code	R ₁	R ₂
B240	0	6

Assembler Format

Op Code R₁,R₂

BAKR 0,6

Assume four cases of initial values, as follows:

	PSW (32-63)	Register 6
1.	00 00 10 D6	02 46 8A CE
2.	00 00 10 D6	82 46 8A CE
3.	80 00 10 D6	02 46 8A CE
4.	80 00 10 D6	82 46 8A CE

The results in the four cases are as follows:

	Return Value	Branch Value and PSW (32-63)
1.	00 00 10 DA	00 46 8A CE
2.	00 00 10 DA	00 46 8A CE
3.	80 00 10 DA	82 46 8A CE
4.	80 00 10 DA	82 46 8A CE

BAKR Example 2

This example shows BAKR used in a called program. BAKR does not perform a branch, and the return is to be as specified in general register R₁.

The format of the BAKR instruction is:

Machine Format

Op Code	R ₁	R ₂
B240	5	0

Assembler Format

Op Code R₁,R₂

BAKR 5,0

Assume four cases of initial values, as follows:

	Register 5	PSW (32-63)
1.	04 00 10 D6	00 46 8A CE
2.	04 00 10 D6	82 46 8A CE

3. 84 00 10 D6 00 46 8A CE
4. 84 00 10 D6 82 46 8A CE

The results in the four cases are as follows:

	Return Value	Branch Value and PSW (32-63)
1.	00 00 10 D6	00 46 8A D2
2.	00 00 10 D6	82 46 8A D2
3.	84 00 10 D6	00 46 8A D2
4.	84 00 10 D6	82 46 8A D2

BAKR Example 3

This example shows BAKR used in a called program. BAKR performs a branch, and the return is to be as specified in general register R₁.

The format of the BAKR instruction is:

Machine Format

Op Code	R ₁	R ₂
B240	5	6

Assembler Format

Op Code R₁,R₂

BAKR 5,6

Assume eight cases of initial values, as follows:

	Register 5	Register 6	PSW (32-63)
1.	04 00 10 D6	06 99 99 00	00 46 8A CE
2.	04 00 10 D6	06 99 99 00	82 46 8A CE
3.	04 00 10 D6	86 99 99 00	00 46 8A CE
4.	04 00 10 D6	86 99 99 00	82 46 8A CE
5.	84 00 10 D6	06 99 99 00	00 46 8A CE
6.	84 00 10 D6	06 99 99 00	82 46 8A CE
7.	84 00 10 D6	86 99 99 00	00 46 8A CE
8.	84 00 10 D6	86 99 99 00	82 46 8A CE

The results in the eight cases are as follows:

	Return Value	Branch Value and PSW (32-63)
1.	00 00 10 D6	00 99 99 00
2.	00 00 10 D6	86 99 99 00
3.	00 00 10 D6	00 99 99 00
4.	00 00 10 D6	86 99 99 00
5.	84 00 10 D6	00 99 99 00
6.	84 00 10 D6	86 99 99 00
7.	84 00 10 D6	00 99 99 00
8.	84 00 10 D6	86 99 99 00

BRANCH ON CONDITION (BC, BCR)

The BRANCH ON CONDITION instruction tests the condition code to see whether a branch should or should not occur. The branch occurs only if the current condition code corresponds to a one bit in a mask specified by the instruction.

Condition Code	Instruction (Mask) Bit	Mask Value
0	8	8
1	9	4
2	10	2
3	11	1

For example, assume that an ADD (A or AR) operation has been performed and that a branch to address 6050 is desired if the sum is zero or less (condition code is 0 or 1). Also assume:

Register 10 contains 00 00 50 00.

Register 11 contains 00 00 10 00.

The RX form of the instruction performs the required test (and branch if necessary) when written as:

Machine Format

Op Code	M ₁	X ₂	B ₂	D ₂
47	C	B	A	050

Assembler Format

Op Code M₁, D₂ (X₂, B₂)

BC 12, X'50' (11, 10)

A mask of 12₁₀ means that there are ones in instruction bits 8 and 9 and zeros in bits 10 and 11, so that branching takes place when the condition code is either 0 or 1.

A mask of 15 would indicate a branch on any condition (an unconditional branch). A mask of zero would indicate that no branch is to occur (a no-operation).

(See also "Linkage Instructions (BAL, BALR, BAS, BASR, BASSM, BSM)" on page A-8 for an example of the BCR instruction.)

BRANCH ON COUNT (BCT, BCTR)

The BRANCH ON COUNT instruction is often used to execute a program loop for a specified number of times. For example, assume that the following represents some lines of coding in an assembler-language program:

```

:
LUPE  AR    8,1
:
BACK  BCT   6,LUPE
:

```

where register 6 contains 00 00 00 03 and the address of LUPE is 6826. Assume that, in order to address this location, register 10 is used as a base register and contains 00 00 68 00.

The format of the BCT instruction is:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
46	6	0	A	026

Assembler Format

Op Code R₁, D₂ (X₂, B₂)

BCT 6, X'26' (0, 10)

The effect of the coding is to execute three times the loop defined by the instructions labeled LUPE through BACK, while register 6 is decremented from three to zero.

BRANCH ON INDEX HIGH (BXH)

BXH Example 1

The BRANCH ON INDEX HIGH instruction is an index-incrementing and loop-controlling instruction that causes a branch whenever the sum of an index value and an increment value is greater than some compare value. For example, assume that:

Register 4 contains 00 00 00 8A = 138₁₀ = the index.

Register 6 contains 00 00 00 02 = 2₁₀ = the increment.

Register 7 contains 00 00 00 AA = 170₁₀ = the compare value.

Register 10 contains 00 00 71 30 = the branch address.

The format of the BXH instruction is:

Machine Format

Op Code	R ₁	R ₃	B ₂	D ₂
86	4	6	A	000

Assembler Format

Op Code R₁,R₃,D₂(B₂)

BXH 4,6,0(10)

When the instruction is executed, first the contents of register 6 are added to register 4, second the sum is compared with the contents of register 7, and third the decision whether to branch is made. After execution:

Register 4 contains 00 00 00 8C = 140₁₀.

Registers 6 and 7 are unchanged.

Since the new value in register 4 is not yet greater than the value in register 7, the branch to address 7130 is not taken. Repeated use of the instruction will eventually cause the branch to be taken when the value in register 4 reaches 172₁₀.

BXH Example 2

When the register used to contain the increment is odd, that register also becomes the compare-value register. The following assembler-language subroutine illustrates how this may be used to search a table.

Table	
2 Bytes	2 Bytes
ARG1	FUNCT1
ARG2	FUNCT2
ARG3	FUNCT3
ARG4	FUNCT4
ARG5	FUNCT5
ARG6	FUNCT6

Assume that:

Register 8 contains the search argument.

Register 9 contains the width of the table in bytes (00 00 00 04).

Register 10 contains the length of the table in bytes (00 00 00 18).

Register 11 contains the starting address of the table.

Register 14 contains the return address to the main program.

As the following subroutine is executed, the argument in register 8 is successively compared with the arguments in the table, starting with argument 6 and working backward to argument 1. If an equality is found, the corresponding function replaces the argument in register 8. If an equality is not found, zero replaces the argument in register 8.

```

SEARCH    LNR  9,9
NOTEQUAL  BXH  10,9,LOOP
NOTFOUND  SR   8,8
          BCR  15,14
LOOP      CH   8,0(10,11)
          BC   7,NOTEQUAL
          LH   8,2(10,11)
          BCR  15,14

```

The first instruction (LNR) causes the value in register 9 to be made negative. After execution of this instruction, register 9 contains FF FF FF FC = -4₁₀. Considering the case when no equality is found, the BXH instruction will be executed seven times. Each time BXH is executed, a value of -4 is added to register 10, thus reducing the value in register 10 by 4. The new value in register 10 is compared with the -4 value in register 9. The branch is taken each time until the value in register 10 is -4. Then the branch is not taken, and the SR instruction sets register 8 to zero.

BRANCH ON INDEX LOW OR EQUAL (BXLE)

The BRANCH ON INDEX LOW OR EQUAL instruction performs the same operation as BRANCH ON INDEX HIGH, except that branching occurs when the sum is lower than or equal to (instead of higher than) the compare value. As the instruction which increments and tests an index value in a program loop, BXLE is useful at the end of the loop and BXH at the beginning. The following assembler-language routines illustrate loops with BXLE.

BXLE Example 1

Assume that a group of ten 32-bit signed binary integers are stored at consecutive locations, starting at location GROUP. The integers are to be added together, and the sum is to be stored at location SUM.

SR	5,5	Set sum to zero
LA	6,GROUP	Load first address
SR	7,7	Set index to zero
LA	8,4	Load increment 4
LA	9,39	Load compare value
LOOP A	5,0(7,6)	Add integer to sum
BXLE	7,8,LOOP	Test end of loop
ST	5,SUM	Store sum

The two-instruction loop contains an ADD (A) instruction which adds each integer to the contents of general register 5. The ADD instruction uses the contents of general register 7 as an index value to modify the starting address obtained from register 6. Next, BXLE increments the index value by 4, the increment previously loaded into register 8, and compares it with the compare value in register 9, the odd register of this even-odd pair. The compare value was previously set to 39, which is one less than the number of bytes in the data area; this is also the address, relative to the starting address, of the rightmost byte of the last integer to be added. When the last integer has been added, BXLE increments the index value to the next relative address (40), which is found to be greater than the compare value (39) so that no branching takes place.

BXLE Example 2

The technique illustrated in Example 1 is restricted to loops containing instructions in the RX instruction format. That format allows both a base register and an index register to be specified (double indexing).

For instructions in other formats, where an index register cannot be specified, the previous technique may be modified by having the address itself serve as the index value in a BXLE instruction and by using as the compare value the address of the last byte rather than its relative address. The base register then provides the address directly at each iteration of the loop, and it is not necessary to specify a second register to hold the index value (single indexing).

In the following example, an AND (NI) instruction in the SI instruction format sets to zero the rightmost bit of each of the same group of integers as in Example

1, thus making all of them even. The I_2 field of the NI instruction contains the byte X'FE', which consists of seven ones and a zero. That byte is ANDed into byte 3, the rightmost byte, of each of the integers in turn.

LA	6,GROUP	Load first address
LA	8,4	Load increment 4
LA	9,GROUP+39	Load compare value
LOOP NI	3(6),X'FE'	AND immediate
BXLE	6,8,LOOP	Test end of loop

The technique shown in Example 2 does not work, however, on an ESA/370 system when it is in the 31-bit addressing mode and the data is located at the rightmost end of a 31-bit address space. In this case, the compare value would be set to $2^{31}-1$, which is the largest possible 32-bit signed binary value. The reason the technique does not work is that the BXLE and BXH instructions treat their operands as 32-bit signed binary integers. When the address in general register 6 reaches the value $2^{31}-4$, BXLE increments it to a value that is interpreted as -2^{31} , rather than 2^{31} , and the comparison remains low, which causes looping to continue indefinitely.

This situation can be avoided by not allowing data areas to extend to the rightmost location in a 31-bit address space or by using other techniques; these may include double indexing when possible, as in Example 1, or starting at the end and stepping downward through the data area with a negative increment.

COMPARE AND FORM CODEWORD (CFC)

See "Sorting Instructions" on page A-53.

COMPARE HALFWORD (CH)

The COMPARE HALFWORD instruction compares a 16-bit signed binary integer in storage with the contents of a register. For example, assume that:

Register 4 contains FF FF 80 00 = $-32,768_{10}$.

Register 13 contains 00 01 60 50.

Storage locations 16080-16081 contain 8000 = $-32,768_{10}$.

When the instruction:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
49	4	0	D	030

Assembler Format

Op Code R₁,D₂(X₂,B₂)

CH 4,X'30'(0,13)

is executed, the contents of locations 16080-16081 are fetched, expanded to 32 bits (the sign bit is propagated to the left), and compared with the contents of register 4. Because the two numbers are equal, condition code 0 is set.

COMPARE LOGICAL (CL, CLC, CLI, CLR)

The COMPARE LOGICAL instruction differs from the signed-binary comparison instructions (C, CH, CR) in that all quantities are handled as unsigned binary integers or as unstructured data.

CLC Example

The COMPARE LOGICAL (CLC) instruction can be used to perform the byte-by-byte comparison of storage fields up to 256 bytes in length. For example, assume that the following two fields of data are in storage:

Field 1

1886																1891
D1	D6	C8	D5	E2	D6	D5	6B	C1	4B	C2	4B					

Field 2

1900																190B
D1	D6	C8	D5	E2	D6	D5	6B	C1	4B	C3	4B					

Also assume:

Register 9 contains 00 00 18 80.

Register 7 contains 00 00 19 00.

Execution of the instruction:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
D5	0B	9	006	7	000

Assembler Format

Op Code D₁(L,B₁),D₂(B₂)

CLC 6(12,9),0(7)

sets condition code 1, indicating that the contents of field 1 are lower in value than the contents of field 2.

Because the collating sequence of the EBCDIC code is determined simply by a logical comparison of the bits in the code, the CLC instruction can be used to collate EBCDIC-coded fields. For example, in EBCDIC, the above two data fields are:

Field 1: JOHNSON,A.B.

Field 2: JOHNSON,A.C.

Condition code 1 indicates that JOHNSON,A.B. should precede JOHNSON,A.C. for the fields to be in alphabetic sequence.

CLI Example

The COMPARE LOGICAL (CLI) instruction compares a byte from the instruction stream with a byte from storage. For example, assume that:

Register 10 contains 00 00 17 00.

Storage location 1703 contains 7E.

Execution of the instruction:

Machine Format

Op Code	I ₂	B ₁	D ₁
95	AF	A	003

Assembler Format

Op Code D₁(B₁),I₂

CLI 3(10),X'AF'

sets condition code 1, indicating that the first operand (the quantity in main storage) is lower than the second (immediate) operand.

CLR Example

Assume that:

Register 4 contains 00 00 00 01 = 1.

Register 7 contains FF FF FF FF = $2^{32} - 1$.

Execution of the instruction:

Machine Format

Op Code	R ₁	R ₂
15	4	7

Assembler Format

Op Code	R ₁ , R ₂
CLR	4, 7

sets condition code 1. Condition code 1 indicates that the first operand is lower than the second.

If, instead, the signed-binary comparison instruction COMPARE (CR) had been executed, the contents of register 4 would have been interpreted as +1 and the contents of register 7 as -1. Thus, the first operand would have been higher, so that condition code 2 would have been set.

COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)

The COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) instruction provides a means of comparing bytes selected from a general register to a contiguous field of bytes in storage. The M₃ field of the CLM instruction is a four-bit mask that selects zero to four bytes from a general register, each mask bit corresponding, left to right, to a register byte. In the comparison, the register bytes corresponding to ones in the mask are treated as a contiguous field. The operation proceeds left to right. For example, assume that:

Storage locations 10200-10202 contain F0 BC 7B.

Register 12 contains 00 01 00 00.

Register 6 contains F0 BC 5C 7B.

Execution of the instruction:

Machine Format

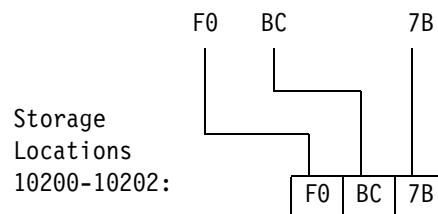
Op Code	R ₁	M ₃	B ₂	D ₂
BD	6	D	C	200

Assembler Format

Op Code	R ₁ , M ₃ , D ₂ (B ₂)
CLM	6, B'1101', X'200' (12)

causes the following comparison:

Register 6:	F0	BC	5C	7B
Mask M ₃ :	1	1	0	1
	--	--		--

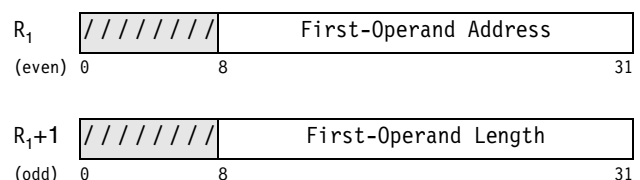


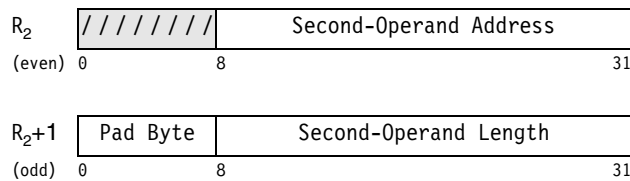
Because the selected bytes are equal, condition code 0 is set.

COMPARE LOGICAL LONG (CLCL)

The COMPARE LOGICAL LONG instruction is used to compare two operands in storage, byte by byte. Each operand can be of any length. Two even-odd pairs of general registers (four registers in all) are used to locate the operands and to control the execution of the CLCL instruction, as illustrated in the following diagram. The first register of each pair must be an even register, and it contains the storage address of an operand. The odd register of each pair contains the length of the operand it covers, and the leftmost byte of the second-operand odd register contains a padding byte which is used to extend the shorter operand, if any, to the same length as the longer operand.

The following illustrates the assignment of registers in the 24-bit addressing mode:





In the 31-bit addressing mode, the operand addresses would be in bit positions 1-31 of the even registers shown above.

Since the CLCL instruction may be interrupted during execution, the interrupting program must preserve the contents of the four registers for use when the instruction is resumed.

The following instructions set up two register pairs to control a text-string comparison. For example, assume:

Operand 1
 Address: 20800₁₆
 Length: 100₁₀

Operand 2
 Address: 20A00₁₆
 Length: 132₁₀

Padding Byte
 Address: 20003₁₆
 Length: 1
 Value: 40₁₆

Register 12 contains 00 02 00 00.

The setup instructions are:

LA	4,X'800'(12)	Set register 4 to start of first operand
LA	5,100	Set register 5 to length of first operand
LA	8,X'A00'(12)	Set register 8 to start of second operand
LA	9,132	Set register 9 to length of second operand
ICM	9,B'1000',3(12)	Insert padding byte in leftmost byte position of register 9

Register pair 4,5 defines the first operand. Bits 8-31 of register 4 contain the storage address of the start of an EBCDIC text string, and bits 8-31 of register 5 contain the length of the string, in this case 100 bytes.

Register pair 8,9 defines the second operand, with bits 8-31 of register 8 containing the starting location of the second operand and bits 8-31 of register 9 containing the length of the second operand, in this case 132 bytes. Bits 0-7 of register 9 contain an EBCDIC blank character (X'40') to pad the shorter operand. In this example, the padding byte is used in the first operand, after the 100th byte, to compare with the remaining bytes in the second operand.

With the register pairs thus set up, the format of the CLCL instruction is:

Machine Format

Op Code	R_1	R_2
0F	4	8

Assembler Format

Op Code	R_1, R_2
CLCL	4,8

When this instruction is executed, the comparison starts at the left end of each operand and proceeds to the right. The operation ends as soon as an inequality is detected or the end of the longer operand is reached.

If this CLCL instruction is interrupted after 60 bytes have compared equal, the operand lengths in registers 5 and 9 will have been decremented to 40 and 72, respectively. The operand addresses in registers 4 and 8 will have been incremented to X'2083C' and X'20A3C'; the leftmost byte of registers 4 and 8 will have been set to zero. The padding byte X'40' remains in register 9. When the CLCL instruction is reexecuted with these register contents, the comparison resumes at the point of interruption.

Now, assume that the instruction is interrupted after 110 bytes. That is, the first 100 bytes of the second operand have compared equal to the first operand, and the next 10 bytes of the second operand have compared equal to the padding byte (blank). The residual operand lengths in registers 5 and 9 are 0 and 22, respectively, and the operand addresses in registers 4 and 8 are X'20864' (the value when the first operand was exhausted) and X'20A6E' (the current value for the second operand).

When the comparison ends, the condition code is set to 0, 1, or 2, depending on whether the first operand

is equal to, less than, or greater than the second operand, respectively.

When the operands are unequal, the addresses in registers 4 and 8 indicate the bytes that caused the mismatch.

COMPARE LOGICAL STRING (CLST)

The COMPARE LOGICAL STRING instruction is used to compare a first operand designated by general register R_1 and a second operand designated by general register R_2 . The comparison is made left to right, byte by byte, until unequal bytes are compared, an ending character specified in general register 0 is encountered in either operand, or a CPU-determined number of bytes have been compared. The condition code is set to 0 if the two operands are equal, to 1 if the first operand is low, to 2 if the second operand is low, or to 3 if a CPU-determined number of bytes have been compared. If the ending character is found in both operands simultaneously, the operands are equal. If it is found in only one operand, that operand is low.

When condition code 1 or 2 is set, the addresses of the last bytes processed in the first and second operands are placed in general registers R_1 and R_2 , respectively. These are the addresses of unequal bytes in the two operands, or they are the address of an ending character in one operand and of the byte in the corresponding byte position in the other operand. When condition code 3 is set, the addresses of the next bytes to be processed are placed in the registers. When condition code 0 is set, the contents of the registers remain unchanged.

Following are examples of first and second operands beginning at decimal locations 1000 and 2000, respectively. The addresses in general registers R_1 and R_2 are 1000 and 2000, respectively. The ending character in general register 0 is 00 hex (as in the C programming language). The values of the operand bytes are shown in hex, and the resulting condition code and final contents of general registers R_1 and R_2 are shown.

Example 1

```
1000      2000
C1 C2 C3 00  C1 C2 C3 00

CC: 0; ( $R_1$ ): 1000; ( $R_2$ ): 2000
```

Example 2

```
1000      2000
40 40 40 C1  40 40 40 C2

CC: 1; ( $R_1$ ): 1003; ( $R_2$ ): 2003
```

Example 3

```
1000      2000
40 40 40 C2  40 40 40 C1

CC: 2; ( $R_1$ ): 1003; ( $R_2$ ): 2003
```

Example 4

```
1000      2000
C1 C2 C3 00  C1 C2 C3 C4

CC: 1; ( $R_1$ ): 1003; ( $R_2$ ): 2003
```

Example 5

```
1000      2000
C1 C2 C3 C4  C1 C2 C3 00

CC: 2; ( $R_1$ ): 1003; ( $R_2$ ): 2003
```

Example 6

Assuming that the CPU-determined number of bytes compared is 256:

```
1000      1256 2000      2256
40 .. 40 00  40 .. 40 00

CC: 3; ( $R_1$ ): 1256; ( $R_2$ ): 2256
```

Example 7

```
1000      2000
00 40 40 40  40 40 40 40

CC: 1; ( $R_1$ ): 1000; ( $R_2$ ): 2000
```

Example 8

```
1000      2000
40 40 40 40  00 40 40 40

CC: 2; ( $R_1$ ): 1000; ( $R_2$ ): 2000
```

Example 9

```
1000      2000
00 40 40 40  00 40 40 40

CC: 0; ( $R_1$ ): 1000; ( $R_2$ ): 2000
```

CONVERT TO BINARY (CVB)

The CONVERT TO BINARY instruction converts an eight-byte, signed-packed-decimal number into a signed binary integer and loads the result into a gen-

eral register. After the conversion operation is completed, the number is in the proper form for use as an operand in signed binary arithmetic. For example, assume:

Storage locations 7608-760F contain a decimal number in the signed-packed-decimal format: 00 00 00 00 25 59 4C (+25,594).

The contents of register 7 are not significant.

Register 13 contains 00 00 76 00.

The format of the conversion instruction is:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
4F	7	0	D	008

Assembler Format

Op Code	R ₁ , D ₂ (X ₂ , B ₂)
CVB	7, 8(0, 13)

After the instruction is executed, register 7 contains 00 00 63 FA.

CONVERT TO DECIMAL (CVD)

The CONVERT TO DECIMAL instruction is the opposite of the CONVERT TO BINARY instruction. CVD converts a signed binary integer in a register to signed-packed-decimal number and stores the eight-byte result. For example, assume:

Register 1 contains the signed binary integer: 00 00 0F 0F.

Register 13 contains 00 00 76 00.

The format of the instruction is:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
4E	1	0	D	008

Assembler Format

Op Code	R ₁ , D ₂ (X ₂ , B ₂)
CVD	1, 8(0, 13)

After the instruction is executed, storage locations 7608-760F contain 00 00 00 00 03 85 5C (+3855).

The plus sign generated is the preferred plus sign, 1100₂.

DIVIDE (D, DR)

The DIVIDE instruction divides the dividend in an even-odd register pair by the divisor in a register or in storage. Since the instruction assumes the dividend to be 64 bits long, it is important first to extend a 32-bit dividend on the left with bits equal to the sign bit. For example, assume that:

Storage locations 3550-3553 contain 00 00 08 DE = 2270₁₀ (the dividend).

Storage locations 3554-3557 contain 00 00 00 32 = 50₁₀ (the divisor).

The initial contents of registers 6 and 7 are not significant.

Register 8 contains 00 00 35 50.

The following assembler-language statements load the registers properly and perform the divide operation:

Statement	Comments
L 6, 0(0, 8)	Places 00 00 08 DE into register 6.
SRDA 6, 32(0)	Shifts 00 00 08 DE into register 7. Register 6 is filled with zeros (sign bits).
D 6, 4(0, 8)	Performs the division.

The machine format of the above DIVIDE instruction is:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
5D	6	0	8	004

After the instructions listed above are executed:

Register 6 contains 00 00 00 14 = 20₁₀ = the remainder.

Register 7 contains 00 00 00 2D = 45₁₀ = the quotient.

Note that if the dividend had not been first placed in register 6 and shifted into register 7, register 6 might not have been filled with the proper dividend-sign bits (zeros in this example), and the DIVIDE instruction might not have given the expected results.

EXCLUSIVE OR (X, XC, XI, XR)

When the Boolean operator EXCLUSIVE OR is applied to two bits, the result is one when either, but not both, of the two bits is one; otherwise, the result is zero. When two bytes are EXCLUSIVE ORed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of the EXCLUSIVE OR of two bytes:

First-operand byte: 0011 0101₂

Second-operand byte: 0101 1100₂

Result byte: 0110 1001₂

XC Example

The EXCLUSIVE OR (XC) instruction can be used to exchange the contents of two areas in storage without the use of an intermediate storage area. For example, assume two three-byte fields in storage:

Field 1 359 358

00	17	90
----	----	----

Field 2 360 362

00	14	01
----	----	----

Execution of the instruction (assume that register 7 contains 00 00 03 58):

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
D7	02	7	001	7	008

Assembler Format

Op Code D₁(L, B₁), D₂(B₂)

XC 1(3, 7), 8(7)

Field 1 is EXCLUSIVE ORed with field 2 as follows:

Field 1: 00000000 00010111 10010000₂ = 00 17 90₁₆

Field 2: 00000000 00010100 00000001₂ = 00 14 01₁₆

Result: 00000000 00000011 10010001₂ = 00 03 91₁₆

The result replaces the former contents of field 1. Condition code 1 is set to indicate a nonzero result.

Now, execution of the instruction:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
D7	02	7	008	7	001

Assembler Format

Op Code D₁(L, B₁), D₂(B₂)

XC 8(3, 7), 1(7)

produces the following result:

Field 1: 00000000 00000011 10010001₂ = 00 03 91₁₆

Field 2: 00000000 00010100 00000001₂ = 00 14 01₁₆

Result: 00000000 00010111 10010000₂ = 00 17 90₁₆

The result of this operation replaces the former contents of field 2. Field 2 now contains the original value of field 1. Condition code 1 is set to indicate a nonzero result.

Lastly, execution of the instruction:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
D7	02	7	001	7	008

Assembler Format

Op Code D₁(L, B₁), D₂(B₂)

XC 1(3, 7), 8(7)

produces the following result:

Field 1: 00000000 00000011 10010001₂ = 00 03 91₁₆

Field 2: 00000000 00010111 10010000₂ = 00 17 90₁₆

Result: 00000000 00010100 00000001₂ = 00 14 01₁₆

The result of this operation replaces the former contents of field 1. Field 1 now contains the original value of field 2. Condition code 1 is set to indicate a nonzero result.

XI Example

A frequent use of the EXCLUSIVE OR (XI) instruction is to invert a bit (change a zero bit to a one or a one bit to a zero). For example, assume that storage location 8082 contains $0110\ 1001_2$. To invert the leftmost and rightmost bits without affecting any of the other bits, the following instruction can be used (assume that register 9 contains $00\ 00\ 80\ 80$):

Machine Format

Op Code	I ₂	B ₁	D ₁
97	81	9	002

Assembler Format

Op Code D₁(B₁), I₂

XI 2(9), X'81'

When the instruction is executed, the byte in storage is EXCLUSIVE ORed with the immediate byte (the I₂ field of the instruction):

Location 8082:	$0110\ 1001_2$
Immediate byte:	$1000\ 0001_2$
<hr/>	
Result:	$1110\ 1000_2$

The resulting byte is stored back in location 8082. Condition code 1 is set to indicate a nonzero result.

Notes:

1. With the XC instruction, fields up to 256 bytes in length can be exchanged.
2. With the XR instruction, the contents of two registers can be exchanged.
3. Because the X instruction operates storage to register only, an exchange cannot be made solely by the use of X.
4. A field EXCLUSIVE ORed with itself is cleared to zeros.
5. For additional examples of the use of EXCLUSIVE OR, see "Hexadecimal-Floating-Point-Number Conversion" on page A-44.

EXECUTE (EX)

The EXECUTE instruction causes one *target instruction* in main storage to be executed out of sequence

without actually branching to the target instruction. Unless the R₁ field of the EXECUTE instruction is zero, bits 8-15 of the target instruction are ORed with bits 56-63 of the R₁ register before the target instruction is executed. Thus, EXECUTE may be used to supply the length field for an SS instruction without modifying the SS instruction in storage. For example, assume that a MOVE (MVC) instruction is the target that is located at address 3820, with a format as follows:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
D2	00	C	003	D	000

Assembler Format

Op Code D₁(L, B₁), D₂(B₂)

MVC 3(1, 12), 0(13)

where register 12 contains $00\ 00\ 89\ 13$ and register 13 contains $00\ 00\ 90\ A0$.

Further assume that at storage address 5000, the following EXECUTE instruction is located:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
44	1	0	A	000

Assembler Format

Op Code R₁, D₂(X₂, B₂)

EX 1, 0(0, 10)

where register 10 contains $00\ 00\ 38\ 20$ and register 1 contains $00\ 0F\ F0\ 03$.

When the instruction at 5000 is executed, the rightmost byte of register 1 is ORed with the second byte of the target instruction:

Instruction byte:	$0000\ 0000_2 = 00$
Register byte:	$0000\ 0011_2 = 03$
<hr/>	
Result:	$0000\ 0011_2 = 03$

causing the instruction at 3820 to be executed as if it originally were:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
D2	03	C	003	D	000

Assembler Format

Op Code D₁(L,B₁),D₂(B₂)

MVC 3(4,12),0(13)

However, after execution:

Register 1 is unchanged.

The instruction at 3820 is unchanged.

The contents of the four bytes starting at location 90A0 have been moved to the four bytes starting at location 8916.

The CPU next executes the instruction at address 5004 (PSW bits 104-127 contain 00 50 04).

FIND LEFTMOST ONE (FLOGR)

FIND LEFTMOST ONE may be used to locate free objects, the availability of which is tracked in a hierarchical bit map.

In the following example, the hierarchical bit map consists of multiple levels of bits, each level having 64 times the number of bits as the next lower-numbered level. This example shows four levels, mapping a total of 16M objects, however additional levels may be added to manage larger numbers of objects.

The highest-numbered bit map level (L3 in this illustration) represents the availability of individual objects, for example, records or pages. A one bit indicates that the object is available, and a zero bit means that the object is not available.

In lower-numbered bit map levels (L0, L1, and L2 in this illustration), a bit position corresponds to a doubleword in the next higher-numbered level. When a bit in a lower-numbered bit map level is one, at least one bit of the corresponding doubleword in the higher-level is one. When a bit in a lower-number bit

map level is zero, no bits in the corresponding higher-level doubleword are one.

An example of using FIND LEFTMOST ONES to quickly locate the next available lowest-numbered object, as mapped by such a hierarchical bit map, is shown below. Upon completion, at the label "FOUND", general register 15 contains the number of the located object. If no objects are available, the code branches to the label "NOTFND".

```

LA 15,0          Zero accumulator
LA 9,OFFSET      Point to table
LA 10,4          Load increment
LA 11,ENDOFF-1   Load compare value
FIND LGR 7,15     Copy accum. to R7
      ALGF 7,0(,9) Add # DWs to bits
      SLLG 7,7,3  Get byte offset
      LG 1,BITS(7) Get DW to test
      FLOGR 0,1   Find leftmost one
      BZ NOTFND   No one bits found
      SLLG 15,15,6 Times # bits in DW
      AGR 15,0    Add bit just found
      BXLEG 9,10,FIND Do for each level
FOUND STG 1,BITS(7) Update changed bit
      ...
NOTFND [Any instruction]
```

```

OFFSET DC A((L0-BITS)/8) Offset to level 0
        DC A((L1-BITS)/8) Offset to level 1
        DC A((L2-BITS)/8) Offset to level 2
        DC A((L3-BITS)/8) Offset to level 3
ENDOFF EQU *
```

```

BITS DS 0D          Start of bit map
L0 DS FD            -L0 (64 objects)
L1 DS 64FD          -L1 (4K objects)
L2 DS (64*64)FD     -L2 (256K objects)
L3 DS (64*64*64)FD -L3 (16M objects)
```

This code sequence shows the zeroing of the found bit with the STORE (STG) instruction at the label "FOUND". Proper maintenance of such a hierarchical bit map requires that when all bits of a doubleword become zeros, the corresponding bit in the next lower-numbered level is set to zeros. Similarly, when adding a free object to the highest level, the corresponding bits in lower-numbered levels must be set to one. These maintenance functions are not shown.

Note, in a multiprocessing environment, access to the bit map is assumed to be serialized for the duration of the update process.

INSERT CHARACTERS UNDER MASK (ICM)

The INSERT CHARACTERS UNDER MASK (ICM) instruction may be used to replace all or selected bytes in a general register with bytes from storage and to set the condition code to indicate the value of the inserted field.

For example, if it is desired to insert a three-byte address from FIELD A into register 5 and leave the leftmost byte of the register unchanged, assume:

Machine Format

Op Code	R ₁	M ₃	S ₂
BF	5	7	* * * *

Assembler Format

Op Code R₁,M₃,S₂

ICM 5,B'0111',FIELD A

FIELD A: FE DC BA
 Register 5 (before): 12 34 56 78
 Register 5 (after): 12 FE DC BA
 Condition code (after): 1 (leftmost bit of inserted field is one)

As another example:

Machine Format

Op Code	R ₁	M ₃	S ₂
BF	6	9	* * * *

Assembler Format

Op Code R₁,M₃,S₂

ICM 6,B'1001',FIELD B

FIELD B: 12 34
 Register 6 (before): 00 00 00 00
 Register 6 (after): 12 00 00 34
 Condition code (after): 2 (inserted field is nonzero with leftmost zero bit)

When the mask field contains 1111, the ICM instruction produces the same result as LOAD (L) (provided that the indexing capability of the RX format is not

needed), except that ICM also sets the condition code. The condition-code setting is useful when an all-zero field (condition code 0) or a leftmost one bit (condition code 1) is used as a flag.

LOAD (L, LR)

The LOAD instruction takes four bytes from storage or from a general register and place them unchanged into a general register. For example, assume that the four bytes starting with location 21003 are to be loaded into register 10. Initially:

Register 5 contains 00 02 00 00.

Register 6 contains 00 00 10 03.

The contents of register 10 are not significant.

Storage locations 21003-21006 contain 00 00 AB CD.

To load register 10, the RX form of the instruction can be used:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
58	A	5	6	000

Assembler Format

Op Code R₁,D₂(X₂,B₂)

L 10,0(5,6)

After the instruction is executed, register 10 contains 00 00 AB CD.

LOAD ADDRESS (LA)

The LOAD ADDRESS instruction provides a convenient way to place a nonnegative binary integer up to 4095₁₀ in a register without first defining a constant and then using it as an operand. For example, the following instruction places the number 2048₁₀ in register 1:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
41	1	0	0	800

Assembler Format

Op Code $R_1, D_2(X_2, B_2)$

LA 1,2048(0,0)

The LOAD ADDRESS instruction can also be used to increment a register by an amount up to 4095_{10} specified in the D_2 field. Depending on the addressing mode, only the rightmost 24 or 31 bits of the sum are retained, however. The leftmost bits of the 32-bit result are set to zeros. For example, assume that register 5 contains 00 12 34 56.

The instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
41	5	0	5	00A

Assembler Format

Op Code $R_1, D_2(X_2, B_2)$

LA 5,10(0,5)

adds 10 (decimal) to the contents of register 5 as follows:

Register 5 (old): 00 12 34 56
 D_2 field: 00 00 00 0A

Register 5 (new): 00 12 34 60

The register may be specified as either B_2 or X_2 . Thus, the instruction LA 5,10(5,0) produces the same result.

As the most general example, the instruction LA 6,10(5,4) forms the sum of three values: the contents of register 4, the contents of register 5, and a displacement of 10 and places the 24-bit or 31-bit sum with zeros appended on the left in register 6.

LOAD HALFWORD (LH)

The LOAD HALFWORD instruction places unchanged a halfword from storage into the right half of a register. The left half of the register is loaded with zeros or ones according to the sign (leftmost bit) of the halfword.

For example, assume that the two bytes in storage locations 1803-1804 are to be loaded into register 6. Also assume:

The contents of register 6 are not significant.

Register 14 contains 00 00 18 03.

Locations 1803-1804 contain 00 20.

The instruction required to load the register is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
48	6	0	E	000

Assembler Format

Op Code $R_1, D_2(X_2, B_2)$

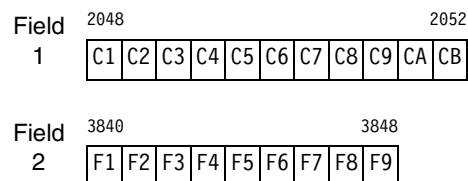
LH 6,0(0,14)

After the instruction is executed, register 6 contains 00 00 00 20. If locations 1803-1804 had contained a negative number, for example, A7 B6, a minus sign would have been propagated to the left, giving FF FF A7 B6 as the final result in register 6.

MOVE (MVC, MVI)

MVC Example

The MOVE (MVC) instruction can be used to move data from one storage location to another. For example, assume that the following two fields are in storage:



Also assume:

Register 1 contains 00 00 20 48.

Register 2 contains 00 00 38 40.

With the following instruction, the first eight bytes of field 2 replace the first eight bytes of field 1:

Op Code	L	B ₁	D ₁	B ₂	D ₂
D2	07	1	000	2	000

MVC $0(8,1), 0(2)$

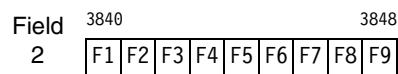
Field	2048											2052
1	F1	F2	F3	F4	F5	F6	F7	F8	C9	CA	CB	

00	F1	F2	F3	F4	F5	F6	F7	F8
----	----	----	----	----	----	----	----	----

MVC $1(8,11), 0(11)$

Op Code	I ₂	B ₁	D ₁
92	5B	1	000

Field	2048									2052	
1	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB



A-25

Machine Format

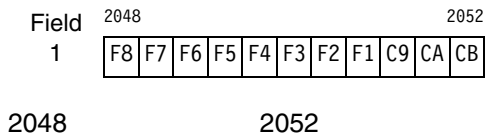
Op Code	L	B ₁	D ₁	B ₂	D ₂
E8	07	1	000	2	007

Assembler Format

Op Code D₁(L,B₁),D₂(B₂)

MVCIN 0(8,1),7(2)

After the instruction is executed, field 1 becomes:



Field 2 is unchanged.

Note: This example uses the same general registers, storage locations, and original values as the first example for MVC. For MVCIN, the second-operand address must designate the rightmost byte of the field to be moved, in this case location 3847. This is accomplished by means of the 7 in the D₂ field of the instruction.

MOVE LONG (MVCL)

The MOVE LONG (MVCL) instruction can be used for moving data in storage as in the first example of the MVC instruction, provided that the two operands do not overlap. MVCL differs from MVC in that the address and length of each operand are specified in an even-odd pair of general registers. Consequently, MVCL can be used to move more than 256 bytes of data with one instruction. As an example, assume:

Register 2 contains 00 0A 00 00.

Register 3 contains 00 00 08 00.

Register 8 contains 00 06 00 00.

Register 9 contains 00 00 08 00.

Execution of the instruction:

Machine Format

Op Code	R ₁	R ₂
0E	8	2

Assembler Format

Op Code R₁,R₂

MVCL 8,2

moves 2,048₁₀ bytes from locations A0000-A07FF to locations 60000-607FF. Assuming that the CPU is in the 24-bit addressing mode, bits 8-31 of registers 2 and 8 are incremented by 800₁₆, and bits 0-7 of registers 2 and 8 are set to zeros. Bits 8-31 of registers 3 and 9 are decremented to zero. Condition code 0 is set to indicate that the operand lengths are equal.

If register 3 had contained F0 00 04 00, only the 1,024₁₀ bytes from locations A0000-A03FF would have been moved to locations 60000-603FF. The remaining locations 60400-607FF of the first operand would have been filled with 1,024 copies of the padding byte X'F0', as specified by the leftmost byte of register 3. Bits 8-31 of register 2 would have been incremented by 400₁₆, bits 8-31 of register 8 would have been incremented by 800₁₆, and bits 0-7 of registers 2 and 8 would have been set to zeros. Bits 8-31 of registers 3 and 9 would still have been decremented to zero. Condition code 2 would have been set to indicate that the first operand was longer than the second.

The technique for setting a field to zeros that is illustrated in the second example of MVC cannot be used with MVCL. If the registers were set up to attempt such an operation with MVCL, no data movement would take place and condition code 3 would indicate destructive overlap.

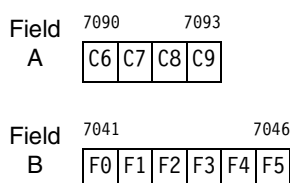
Instead, MVCL may be used to clear a storage area to zeros as follows. Assume register 8 and 9 are set up as before. Register 3 contains only zeros, specifying zero length for the second operand and a zero padding byte. Register 2 is not used to access storage, and its contents are not significant. Executing the instruction MVCL 8,2 causes locations 60000-607FF to be filled with zeros. Bits 8-31 of register 8 are incremented by 800₁₆, and bits 0-7 of registers 2 and 8 are set to zeros. Bits 8-31 of register 9 are decremented to zero, and condition code 2 is set to indicate that the first operand is longer than the second.

MOVE NUMERICS (MVN)

Two related instructions, MOVE NUMERICS and MOVE ZONES, may be used with decimal data in the zoned format to operate separately on the rightmost

four bits (the numeric bits) and the leftmost four bits (the zone bits) of each byte. Both are similar to MOVE (MVC), except that MOVE NUMERICS moves only the numeric bits and MOVE ZONES moves only the zone bits.

To illustrate the operation of the MOVE NUMERICS instruction, assume that the following two fields are in storage:



Also assume:

Register 14 contains 00 00 70 90.

Register 15 contains 00 00 70 40.

After the instruction:

Machine Format

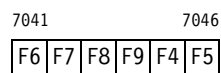
Op Code	L	B ₁	D ₁	B ₂	D ₂
D1	03	F	001	E	000

Assembler Format

Op Code D₁(L, B₁), D₂(B₂)

MVN 1(4, 15), 0(14)

is executed, field B becomes:



The numeric bits of the bytes at locations 7090-7093 have been stored in the numeric bits of the bytes at locations 7041-7044. The contents of locations 7090-7093 and 7045-7046 are unchanged.

MOVE STRING (MVST)

The MOVE STRING instruction is used to move a second operand designated by general register R₂ to a first-operand location designated by general register R₁. The movement is made left to right until an ending character specified in general register 0 has been moved or a CPU-determined number of bytes

have been moved. The condition code is set to 1 if the ending character was moved or to 3 if a CPU-determined number of bytes were moved.

When condition code 1 is set, the address of the ending character in the first operand is placed in general register R₁, and the contents of general register R₂ remain unchanged. When condition code 3 is set, the address of the next byte to be processed in the first and second operands is placed in general registers R₁ and R₂, respectively.

Following is an example program that sets string A equal to the concatenation of string B followed by string C, where the length of each of strings B and C is unknown, and the end of each of strings B and C is indicated by an ending character of 00 hex (as in the C programming language). The program is not written for execution in the access-register mode.

```

L      4, STRAADR
L      5, STRBADR
SR      0, 0
LOOP1  MVST  4, 5
BC      1, LOOP1
L      5, STRCADR
LOOP2  MVST  4, 5
BC      1, LOOP2
[Any instruction]
```

MOVE WITH OFFSET (MVO)

MOVE WITH OFFSET may be used to shift a signed-packed-decimal number an odd number of digit positions or to concatenate a sign to an unsigned-packed-decimal number.

Assume that the three-byte unsigned-packed-decimal number in storage locations 4500-4502 is to be moved to locations 5600-5603 and given the sign of the signed-packed-decimal number ending at location 5603. Also assume:

Register 12 contains 00 00 56 00.

Register 15 contains 00 00 45 00.

Storage locations 5600-5603 contain 77 88 99 0C.

Storage locations 4500-4502 contain 12 34 56.

After the instruction:

Machine Format

Op Code	L ₁	L ₂	B ₁	D ₁	B ₂	D ₂
F1	3	2	C	000	F	000

Assembler Format

Op Code D₁(L₁,B₁),D₂(L₂,B₂)

MV0 0(4,12),0(3,15)

is executed, the storage locations 5600-5603 contain 01 23 45 6C. Note that the second operand is extended on the left with one zero to fill out the first-operand field.

MOVE ZONES (MVZ)

The MOVE ZONES instruction can operate on overlapping or nonoverlapping fields, as can the instructions MOVE (MVC) and MOVE NUMERICS. When operating on nonoverlapping fields, MOVE ZONES works like the MOVE NUMERICS instruction (see its example), except that MOVE ZONES moves only the zone bits of each byte. To illustrate the use of MOVE ZONES with overlapping fields, assume that the following data field is in storage:

800						805
F1	C2	F3	C4	F5	C6	

Also assume that register 15 contains 00 00 08 00. The instruction:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
D3	04	F	001	F	000

Assembler Format

Op Code D₁(L,B₁),D₂(B₂)

MVZ 1(5,15),0(15)

propagates the zone bits from the byte at address 800 through the entire field, so that the field becomes:

800	805				
F1	F2	F3	F4	F5	F6

MULTIPLY (M, MR)

Assume that a number in register 5 is to be multiplied by the contents of a four-byte field at address 3750. Initially:

The contents of register 4 are not significant.

Register 5 contains 00 00 00 9A = 154₁₀ = the multiplicand.

Register 11 contains 00 00 06 00.

Register 12 contains 00 00 30 00.

Storage locations 3750-3753 contain 00 00 00 83 = 131₁₀ = the multiplier.

The instruction required for performing the multiplication is:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
5C	4	B	C	150

Assembler Format

Op Code R₁,D₂(X₂,B₂)

M 4,X'150'(11,12)

After the instruction is executed, the product is in the register pair 4 and 5:

Register 4 contains 00 00 00 00.

Register 5 contains 00 00 4E CE = 20,174₁₀.

Storage locations 3750-3753 are unchanged.

The RR format of the instruction can be used to square the number in a register. Assume that register 7 contains 00 01 00 05. The contents of register 6 are not significant. The instruction:

Machine Format

Op Code	R ₁	R ₂
1C	6	7

Assembler Format

Op Code R₁,R₂

MR 6,7

multiplies the number in register 7 by itself and places the result in the pair of registers 6 and 7:

Register 6 contains 00 00 00 01.

Register 7 contains 00 0A 00 19.

MULTIPLY HALFWORD (MH)

The MULTIPLY HALFWORD instruction is used to multiply the contents of a register by a two-byte field in storage. For example, assume that:

Register 11 contains 00 00 00 15 = 21_{10} = the multiplicand.

Register 14 contains 00 00 01 00.

Register 15 contains 00 00 20 00.

Storage locations 2102-2103 contain FF D9 = - 39_{10} = the multiplier.

The instruction:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
4C	B	E	F	002

Assembler Format

Op Code R₁, D₂(X₂, B₂)

MH 11, 2(14, 15)

multiplies the two numbers. The product, FF FF FC CD = - 819_{10} , replaces the original contents of register 11.

Only the rightmost 32 bits of a product are stored in a register; any significant bits on the left are lost. No program interruption occurs on overflow.

OR (O, OC, OI, OR)

When the Boolean operator OR is applied to two bits, the result is one when either bit is one; otherwise, the result is zero. When two bytes are ORed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of ORing two bytes:

First-operand byte: 0011 0101₂

Second-operand byte: 0101 1100₂

Result byte: 0111 1101₂

OI Example

A frequent use of the OR instruction is to set a particular bit to one. For example, assume that storage location 4891 contains 0100 0010₂. To set the rightmost bit of this byte to one without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

Op Code	I ₂	B ₁	D ₁
96	01	8	001

Assembler Format

Op Code D₁(B₁), I₂

OI 1(8), X'01'

When this instruction is executed, the byte in storage is ORed with the immediate byte (the I₂ field of the instruction):

Location 4891: 0100 0010₂
Immediate byte: 0000 0001₂

Result: 0100 0011₂

The resulting byte with bit 7 set to one is stored back in location 4891. Condition code 1 is set.

PACK (PACK)

Assume that storage locations 1000-1003 contain the following zoned-decimal number that is to be converted to a signed-packed-decimal number and left in the same location:

	1000		1003
Zoned number	F1	F2	F3 C4

Also assume that register 12 contains 00 00 10 00. After the instruction:

Machine Format

Op Code	L ₁	L ₂	B ₁	D ₁	B ₂	D ₂
F2	3	3	C	000	C	000

Assembler Format

Op Code $D_1(L_1, B_1), D_2(L_2, B_2)$

PACK 0(4,12),0(4,12)

is executed, the result in locations 1000-1003 is in the signed-packed-decimal format:

Signed-packed number

1000	1001	1002	1003
00	01	23	4C

Notes:

1. This example illustrates the operation of PACK when the first- and second-operand fields overlap completely.
2. During the operation, the second operand was extended on the left with zeros.

ROTATE THEN EXCLUSIVE OR SELECTED BITS

The following example illustrates the use of ROTATE THEN EXCLUSIVE OR SELECTED BITS to determine if the access-control bits of a page's storage-protection key match the key in the PSW. In this example, only the condition code is set; the result is not placed in the first-operand register.

LAY 9,BLOCK Point to storage block.
IVSK 14,9 Insert key into bits 56-63.
EPSW 4,0 Extract PSW key (bits 40-43).
RXSBG 4,14,128+40,43,16 Compare keys.

On completion, the condition code will be set to zero if the PSW key matches the access-control bits in the storage key of page addressed by general register 9. Condition code 1 will be set if the keys do not match.

ROTATE THEN INSERT SELECTED BITS

1. The following example illustrates the use of ROTATE THEN INSERT SELECTED BITS to extract the various DAT-table indices of a virtual address contained in general register 8. This example shows the use of the zero-remaining-bits control to set the remaining bits of the result registers to zero.

The virtual address in general register 8 is assumed to be 123456789ABCDEF0 hex.

RISBG 1,8,53,128+63,11 Get reg. 1st index.
RISBG 2,8,53,128+63,22 Get reg. 2nd index.
RISBG 3,8,53,128+63,33 Get reg. 3rd index.
RISBG 4,8,53,128+63,44 Get segment index.
RISBG 5,8,56,128+63,52 Get page index.
RISBG 6,8,52,128+63,0 Get byte index.

The condition code following each RISBG is 2. On completion, general registers 1-6 contain the region-first index, region-second index, region-third index, segment index, page index, and byte index, respectively, as follows:

GR1: 0000000000000091 (RFX)
GR2: 0000000000000515 (RSX)
GR3: 00000000000004F1 (RTX)
GR4: 00000000000001AB (SX)
GR5: 00000000000000CD (PX)
GR6: 0000000000000EF0 (BX)

2. The following example illustrates shifting a 64-bit value to the left by L bits:

RISBG $R_1, R_2, 0, X'80' + 63 - L, L$

3. The following example illustrates shifting a 32-bit value to the right by R bits and setting the left-most 32 bits to zero:

RISBG $R_1, R_2, 32 + R, X'80' + 63, -R$

4. The following example illustrates zeroing bits S through E of general register 3.

RISBG 3,3,mod64($E+1$),128+mod64($S-1$),0

Mod64 represents a modulo function which effectively ANDs the value in parentheses with 3F hex.

ROTATE THEN OR SELECTED BITS

The following example illustrates the use of ROTATE THEN OR SELECTED BITS to rotate and combine selected bits of general registers 6 and 8.

LG 6,=X'004C487040CF4600'
LG 8,=X'C07FAF37FC968280'
ROSBG 6,8,8,54,32

On completion, condition code 1 is set, and general register 6 is as follows:

```
GR6: 00DECAF0C0FFEE00
```

Note that only bits 8-54 of the registers are ORed; bits 0-7 and 55-63 of general register 6 are not modified.

SEARCH STRING (SRST)

The SEARCH STRING instruction is used to search a second operand designated by general register R_2 for a character specified in general register 0. The length of the second operand is known — the address of the first byte after the second operand is in general register R_1 .

When the specified character is found, condition code 1 is set, the address of the character is placed in general register R_1 , and the contents of general register R_2 remain unchanged. When the address of the next second-operand byte to be examined equals the address in general register R_1 , condition code 2 is set, and the contents of general register R_1 and R_2 remain unchanged. When a CPU-determined number of second-operand bytes have been examined, condition code 3 is set, the address of the next byte to be processed in the second operand is placed in general register R_2 , and the contents of general register R_1 remain unchanged.

SRST Example 1

Following is an example program that determines the end of string A, as indicated by an ending character equal to 00 hex (as in the C programming language), and then determines the address of the first character equal to C1 hex in the string. The program is based on the assumption that the second operand does not begin at location 0 or wrap around in storage, and, therefore, condition code 2 will not be set by the first SEARCH STRING instruction because of the address in general register 0. The program is not written for execution in the access-register mode.

```
L      5,STRAADR
SR      0,0
LOOP1 SRST 0,5
BC      1,LOOP1
L      5,STRAADR
LR      4,0
LA      0,X'C1'
```

```
LOOP2 SRST 4,5
BC      1,LOOP2
BC      2,NOTFND
FOUND [Any instruction]
...
NOTFND [Any instruction]
```

SRST Example 2

Following is an example program that determines the address of the first character equal to C1 hex in the string A whose length is known. The program is not written for execution in the access-register mode.

```
L      5,STRAADR
L      4,STRALEN
AR      4,5
LA      0,X'C1'
LOOP1 SRST 4,5
BC      1,LOOP1
BC      2,NOTFND
FOUND [Any instruction]
...
NOTFND [Any instruction]
```

In this example, the value in STRALEN may be a length that either does or does not include an ending character at the end of the string, provided that the ending character is not the character for which the search is made.

SHIFT LEFT DOUBLE (SLDA)

The SHIFT LEFT DOUBLE instruction shifts the 63 numeric bits of an even-odd register pair to the left, leaving the sign bit unchanged. Thus, the instruction performs an algebraic left shift of a 64-bit signed binary integer.

For example, if the contents of registers 2 and 3 are:

```
00 7F 0A 72   FE DC BA 98 =
00000000 01111111 00001010 01110010
11111110 11011100 10111010 100110002
```

The instruction:

Machine Format

Op Code	R_1	B_2	D_2
8F	2	////	0
			01F

Assembler Format

Op Code $R_1, D_2(B_2)$

SLDA 2,31(0)

results in registers 2 and 3 both being left-shifted 31 bit positions, so that their new contents are:

7F 6E 5D 4C 00 00 00 00 =
01111111 01101110 01011101 01001100
00000000 00000000 00000000 00000000₂

Because significant bits are shifted out of bit position 1 of register 2, overflow is indicated by setting condition code 3, and, if the fixed-point-overflow mask bit in the PSW is one, a fixed-point-overflow program interruption occurs.

SHIFT LEFT SINGLE (SLA)

The SHIFT LEFT SINGLE instruction is similar to SHIFT LEFT DOUBLE, except that it shifts only the 31 numeric bits of a single register. Therefore, this instruction performs an algebraic left shift of a 32-bit signed binary integer.

For example, if the contents of register 2 are:

00 7F 0A 72 = 00000000 01111111 00001010 01110010₂

The instruction:

Machine Format

Op Code	R_1	B_2	D_2
8B	2	///	008

Assembler Format

Op Code $R_1, D_2(B_2)$

SLA 2,8(0)

results in register 2 being shifted left eight bit positions so that its new contents are:

7F 0A 72 00 = 01111111 00001010 01110010 00000000₂

Condition code 2 is set to indicate that the result is greater than zero.

If a left shift of nine places had been specified, a significant bit would have been shifted out of bit position 1. Condition code 3 would have been set to

indicate this overflow and, if the fixed-point-overflow mask bit in the PSW were one, a fixed-point overflow interruption would have occurred.

STORE CHARACTERS UNDER MASK (STCM)

STORE CHARACTERS UNDER MASK (STCM) may be used to place selected bytes from a register into storage. For example, if it is desired to store a three-byte address from general register 8 into location FIELD3, assume:

Machine Format

Op Code	R_1	M_3	S_2
BE	8	7	* * * *

Register Format

Op Code R_1, M_3, S_2

STCM 8,B'0111',FIELD3

Register 8: 12 34 56 78
FIELD3 (before): not significant
FIELD3 (after): 34 56 78

As another example:

Machine Format

Op Code	R_1	M_3	S_2
BE	9	5	* * * *

Register Format

Op Code R_1, M_3, S_2

STCM 9,B'0101',FIELD2

Register 9: 01 23 45 67
FIELD2 (before): not significant
FIELD2 (after): 23 67

STORE MULTIPLE (STM)

Assume that the contents of general registers 14, 15, 0, and 1 are to be stored in consecutive four-byte fields starting with location 4050 and that:

Register 14 contains 00 00 25 63.

Register 15 contains 00 01 27 36.

Register 0 contains 12 43 00 62.

Register 1 contains 73 26 12 57.

Register 6 contains 00 00 40 00.

The initial contents of locations 4050-405F are not significant.

The STORE MULTIPLE instruction allows the use of just one instruction to store the contents of the four registers:

Machine Format

Op Code	R ₁	R ₃	B ₂	D ₂
90	E	1	6	050

Assembler Format

Op Code R₁, R₃, D₂(B₂)

STM 14, 1, X'50' (6)

After the instruction is executed:

Locations 4050-4053 contain 00 00 25 63.

Locations 4054-4057 contain 00 01 27 36.

Locations 4058-405B contain 12 43 00 62.

Locations 405C-405F contain 73 26 12 57.

TEST UNDER MASK (TM)

The TEST UNDER MASK instruction examines selected bits of a byte and sets the condition code accordingly. For example, assume that:

Storage location 9999 contains FB.

Register 7 contains 00 00 99 90.

Assume the instruction to be:

Machine Format

Op Code	I ₂	B ₁	D ₁
91	C3	7	009

Assembler Format

Op Code D₁(B₁), I₂

TM 9(7), B'11000011'

The instruction tests only those bits of the byte in storage for which the mask bits are ones:

FB = 1111 1011₂

Mask = 1100 0011₂

Test = 11xx xx11₂

Condition code 3 is set: all selected bits in the test result are ones. (The bits marked "x" are ignored.)

If location 9999 had contained B9, the test would have been:

B9 = 1011 1001₂

Mask = 1100 0011₂

Test = 10xx xx01₂

Condition code 1 is set: the selected bits are both zeros and ones.

If location 9999 had contained 3C, the test would have been:

3C = 0011 1100₂

Mask = 1100 0011₂

Test = 00xx xx00₂

Condition code 0 is set: all selected bits are zeros.

Note: Storage location 9999 remains unchanged.

TRANSLATE (TR)

The TRANSLATE instruction can be used to translate data from any character code to any other desired code, provided that each character code consists of eight bits or fewer. An appropriate translation table is required in storage.

In the following example, EBCDIC code is translated to ASCII code. The first step is to create a 256-byte table in storage locations 1000-10FF. This table contains the characters of the ASCII code in the sequence of the binary representation of the EBCDIC code; that is, the ASCII representation of a

character is placed in storage at the starting address of the table plus the binary value of the EBCDIC representation of the same character.

For simplicity, the example shows only the part of the table containing the decimal digits:

10F0								10F9			
30	31	32	33	34	35	36	37	38	39		

Assume that the four-byte field at storage location 2100 contains the EBCDIC code for the digits 1984:

Locations 2100-2103 contain F1 F9 F8 F4.

Register 12 contains 00 00 21 00.

Register 15 contains 00 00 10 00.

As the instruction:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
DC	03	C	000	F	000

Assembler Format

Op Code D₁(L, B₁), D₂(B₂)

TR 0(4, 12), 0(15)

is executed, the binary value of each EBCDIC byte is added to the starting address of the table, and the resulting address is used to fetch an ASCII byte:

Table starting address: 1000

First EBCDIC byte: F1

Address of ASCII byte: 10F1

After execution of the instruction:

Locations 2100-2103 contain 31 39 38 34.

Thus, the ASCII code for the digits 1984 has replaced the EBCDIC code in the four-byte field at storage location 2100.

TRANSLATE AND TEST (TRT)

The TRANSLATE AND TEST instruction can be used to scan a data field for characters with a special meaning. To indicate which characters have a special

meaning, a table similar to the one used for the TRANSLATE instruction is set up, except that zeros in the table indicate characters without any special meaning and nonzero values indicate characters with a special meaning.

Figure A-4 on page A-34 has been set up to distinguish alphameric characters (A to Z and 0 to 9) from blanks, certain special symbols, and all other characters which are considered invalid. EBCDIC coding is assumed. The 256-byte table is assumed stored at locations 2000-20FF.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
200	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
201	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
202	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
203	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
204	04	40	40	40	40	40	40	40	40	40	40	08	40	0C	10	40
205	14	40	40	40	40	40	40	40	40	40	40	18	1C	20	40	40
206	24	28	40	40	40	40	40	40	40	40	40	2C	40	40	40	40
207	40	40	40	40	40	40	40	40	40	40	40	30	34	38	3C	40
208	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
209	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20A	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20B	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20C	40	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20D	40	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20E	40	40	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20F	00	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40

Note: If the character codes in the statement being translated occupy a range smaller than 00 through FF₁₆, a table of fewer than 256 bytes can be used.

Figure A-4. Translate and Test Table

The table entries for the alphameric characters in EBCDIC are 00; thus, the letter A (code C1) corresponds to byte location 20C1, which contains 00.

The 15 special symbols have nonzero entries from 04₁₆ to 3C₁₆ in increments of 4. Thus, the blank (code 40) has the entry 04₁₆, the period (code 4B) has the entry 08₁₆, and so on.

All other table positions have the entry 40₁₆ to indicate an invalid character.

The table entries are chosen so that they may be used to select one of a list of 16 words containing

addresses of different routines to be entered for each special symbol or invalid character encountered during the scan.

Assume that this list of 16 branch addresses is stored at locations 3004-3043.

Starting at storage location CA80, there is the following sequence of 21₁₀ EBCDIC characters, where "b" stands for a blank.

Locations CA80-CA94:
UNPKbPROUT(9),WORD(5)

Also assume:

Register 1 contains 00 00 CA 7F.

Register 2 contains 00 00 30 00.

Register 15 contains 00 00 20 00.

As the instruction:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
DD	14	1	001	F	000

Assembler Format

Op Code D₁(L,B₁),D₂(B₂)

TRT 1(21,1),0(15)

is executed, the value of the first source byte, the EBCDIC code for the letter U, is added to the starting address of the table to produce the address of the table entry to be examined:

Table starting address	2000
First source byte (U)	E4
<hr/>	
Address of table entry	20E4

Because zeros were placed in storage location 20E4, no special action occurs. The operation continues with the second and subsequent source bytes until it reaches the blank in location CA84. When this symbol is reached, its value is added to the starting address of the table, as usual:

Table starting address	2000
Source byte (blank)	40
<hr/>	
Address of table entry	2040

Because location 2040 contains a nonzero value, the following actions occur:

The address of the source byte, 00CA84, is placed in the rightmost 24 bits of register 1.

The table entry, 04, is placed in the rightmost eight bits of register 2, which now contains 00 00 30 04.

Condition code 1 is set (scan not completed).

The TRANSLATE AND TEST instruction may be followed by instructions to branch to the routine at the address found at location 3004, which corresponds to the blank character encountered in the scan. When this routine is completed, program control may return to the TRANSLATE AND TEST instruction to continue the scan, except that the length must first be adjusted for the characters already scanned.

For this purpose, the TRANSLATE AND TEST may be executed by the use of an EXECUTE instruction, which supplies the length specification from a general register. In this way, a complete statement scan can be performed with a single TRANSLATE AND TEST instruction used repeatedly by means of EXECUTE, and without modifying any instructions in storage. In the example, after the first execution of TRANSLATE AND TEST, register 1 contains the address of the last source byte translated. It is then a simple matter to subtract this address from the address of the last source byte (CA94) to produce a length specification. This length minus one is placed in the register that is referenced as the R₁ field of the EXECUTE instruction. (Note that the length code in the machine format is one less than the total number of bytes in the field.) The second-operand address of the EXECUTE instruction points to the TRANSLATE AND TEST instruction, which is the same as illustrated above, except for the length (L) which is set to zero.

UNPACK (UNPK)

Assume that storage locations 2501-2502 contain a signed-packed-decimal number that is to be unpacked and placed in storage locations 1000-1004. Also assume:

Register 12 contains 00 00 10 00.

Register 13 contains 00 00 25 00.

Storage locations 2501-2502 contain 12 3D.

The initial contents of storage locations 1000-1004 are not significant.

After the instruction:

Machine Format

Op Code	L ₁	L ₂	B ₁	D ₁	B ₂	D ₂
F3	4	1	C	000	D	001

Assembler Format

Op Code D₁(L₁,B₁),D₂(L₂,B₂)

UNPK 0(5,12),1(2,13)

is executed, the storage locations 1000-1004 contain F0 F0 F1 F2 D3.

UPDATE TREE (UPT)

See "Sorting Instructions" on page A-53.

Decimal Instructions

(See Chapter 8, "Decimal Instructions" for a complete description of the decimal instructions.)

ADD DECIMAL (AP)

Assume that the signed-packed-decimal number at storage locations 500-503 is to be added to the signed-packed-decimal number at locations 2000-2002. Also assume:

Register 12 contains 00 00 20 00.

Register 13 contains 00 00 05 00.

Storage locations 2000-2002 contain 38 46 0D (a negative number).

Storage locations 500-503 contain 01 12 34 5C (a positive number).

After the instruction:

Machine Format

Op Code	L ₁	L ₂	B ₁	D ₁	B ₂	D ₂
FA	2	3	C	000	D	000

Assembler Format

Op Code D₁(L₁,B₁),D₂(L₂,B₂)

AP 0(3,12),0(4,13)

is executed, the storage locations 2000-2002 contain 73 88 5C; condition code 2 is set to indicate that the result is greater than zero. Note that:

1. Because the two numbers had different signs, they were in effect subtracted.
2. Although the second operand is longer than the first operand, no overflow interruption occurs because the result can be entirely contained within the first operand.

COMPARE DECIMAL (CP)

Assume that the signed-packed-decimal contents of storage locations 700-703 are to be algebraically compared with the signed-packed-decimal contents of locations 500-502. Also assume:

Register 12 contains 00 00 06 00.

Register 13 contains 00 00 03 00.

Storage locations 700-703 contain 17 25 35 6D.

Storage locations 500-502 contain 72 14 2D.

After the instruction:

Machine Format

Op Code	L ₁	L ₂	B ₁	D ₁	B ₂	D ₂
F9	3	2	C	100	D	200

Assembler Format

Op Code D₁(L₁,B₁),D₂(L₂,B₂)

CP X'100'(4,12),X'200'(3,13)

is executed, condition code 1 is set, indicating that the first operand (the contents of locations 700-703) is less than the second.

DIVIDE DECIMAL (DP)

Assume that the signed-packed-decimal number at storage locations 2000-2004 (the dividend) is to be divided by the signed-packed-decimal number at locations 3000-3001 (the divisor). Also assume:

Register 12 contains 00 00 20 00.

Register 13 contains 00 00 30 00.

Storage locations 2000-2004 contain 01 23 45 67 8C.

Storage locations 3000-3001 contain 32 1D.

After the instruction:

Machine Format

Op Code	L ₁	L ₂	B ₁	D ₁	B ₂	D ₂
FD	4	1	C	000	D	000

Assembler Format

Op Code D₁(L₁,B₁),D₂(L₂,B₂)

DP 0(5,12),0(2,13)

is executed, the dividend is entirely replaced by the signed quotient and remainder, as follows:

	2000		2004
Locations 2000-2004	38	46	0D 01 8C
	quotient		remainder

Notes:

1. Because the dividend and divisor have different signs, the quotient receives a negative sign.
2. The remainder receives the sign of the dividend and the length of the divisor.
3. If an attempt were made to divide the dividend by the one-byte field at location 3001, the quotient would be too long to fit within the four bytes allotted to it. A decimal-divide exception would exist, causing a program interruption.

EDIT (ED)

Before data in the signed-packed-decimal or unsigned-packed-decimal format can be used in a

printed report, digits and signs must be converted to printable characters. Moreover, punctuation marks, such as commas and decimal points, may have to be inserted in appropriate places. The highly flexible EDIT instruction performs these functions in a single instruction execution.

This example shows step-by-step one way that the EDIT instruction can be used. The field to be edited (the source) is four bytes long; it is edited against a pattern 13 bytes long. The following symbols are used:

Symbol	Meaning
b (Hexadecimal 40)	Blank character
((Hexadecimal 21)	Significance starter
d (Hexadecimal 20)	Digit selector

Assume that register 12 contains:

00 00 10 00

and that the source and pattern fields are:

Source

1200	1203
02	57 42 6C

↑
+

Pattern

1000	100C
40	20 20 6B 20 21 20 4B 20 20 40 C3 D9
b	d d , d (d . d d b C R

Execution of the instruction:

Machine Format

Op Code	L	B ₁	D ₁	B ₂	D ₂
DE	0C	C	000	C	200

Assembler Format

Op Code D₁(L,B₁),D₂(B₂)

ED 0(13,12),X'200'(12)

alters the pattern field as follows:

Pattern

1000					100C							
40	40	40	40	40	5B	F0	4B	F2	F6	40	C3	D9
b	b	b	b	b	\$	0	.	2	6	b	C	R

This pattern field prints as:

\$0.26 CR

Condition code 1 is set because the number is less than zero.

MULTIPLY DECIMAL (MP)

Assume that the signed-packed-decimal number in storage locations 1202-1204 (the multiplicand) is to be multiplied by the signed-packed-decimal number in locations 500-501 (the multiplier).

	1202	1204
Multiplicand	38	46 0D

	500	501
Multiplier	32	1D

The multiplicand must first be extended to have at least two bytes of leftmost zeros, corresponding to the multiplier length, so as to avoid a data exception during the multiplication. ZERO AND ADD can be used to move the multiplicand into a longer field. Assume:

Register 4 contains 00 00 12 00.

Register 6 contains 00 00 05 00.

Then execution of the instruction:

ZAP X'100'(5,4),2(3,4)

sets up a new multiplicand in storage locations 1300-1304:

	1300	1304
Multiplicand (new)	00	00 38 46 0D

Now, after the instruction:

Machine Format

Op Code	L ₁	L ₂	B ₁	D ₁	B ₂	D ₂
FC	4	1	4	100	6	000

Assembler Format

Op Code D₁(L₁,B₁),D₂(L₂,B₂)

MP X'100'(5,4),0(2,6)

is executed, storage locations 1300-1304 contain the product: 01 23 45 66 0C.

SHIFT AND ROUND DECIMAL (SRP)

The SHIFT AND ROUND DECIMAL (SRP) instruction can be used for shifting decimal numbers in storage to the left or right. When a number is shifted right, rounding can also be done.

Decimal Left Shift

In this example, the contents of storage location FIELD1 are shifted three places to the left, effectively multiplying the contents of FIELD1 by 1000. FIELD1 is six bytes long. The following instruction performs the operation:

Machine Format

Op Code	L ₁	I ₃	S ₁	B ₂	D ₂
F0	5	0	* * * *	0	003

Assembler Format

Op Code S₁(L₁),S₂,I₃

SRP FIELD1(6),3,0

FIELD1 (before): 00 01 23 45 67 8C

FIELD1 (after): 12 34 56 78 00 0C

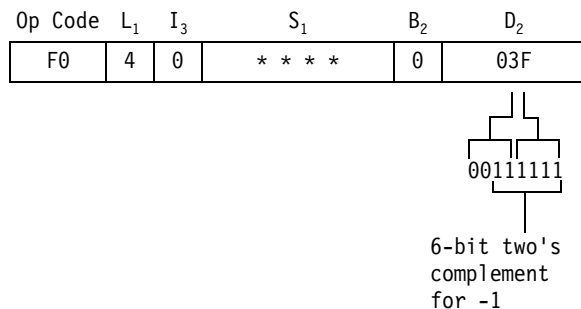
The second-operand address in this instruction specifies the shift amount (three places). The rounding digit, I₃, is not used in a left shift, but it must be a valid decimal digit. After execution, condition code 2 is set to show that the result is greater than zero.

Decimal Right Shift

In this example, the contents of storage location FIELD2 are shifted one place to the right, effectively

dividing the contents of FIELD2 by 10 and discarding the remainder. FIELD2 is five bytes in length. The following instruction performs this operation:

Machine Format



Assembler Format

Op Code S₁(L₁), S₂, I₃

SRP FIELD2(5), 64-1, 0

FIELD 2 (before): 01 23 45 67 8C

FIELD 2 (after): 00 12 34 56 7C

In the SRP instruction, shifts to the right are specified in the second-operand address by negative shift values, which are represented as a six-bit value in two's complement form.

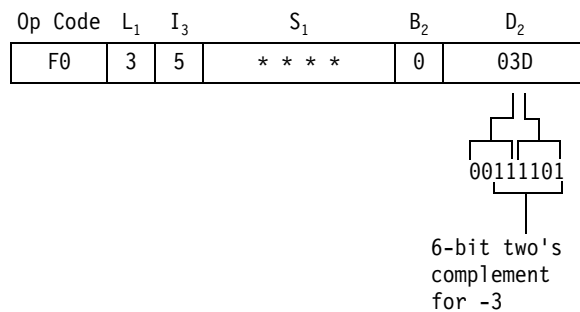
The six-bit two's complement of a number, n, can be specified as 64 - n. In this example, a right shift of one is represented as 64 - 1.

Condition code 2 is set.

Decimal Right Shift and Round

In this example, the contents of storage location FIELD3 are shifted three places to the right and rounded, in effect dividing by 1000 and rounding up. FIELD3 is four bytes in length.

Machine Format



Assembler Format

Op Code S₁(L₁), S₂, I₃

SRP FIELD3(4), 64-3, 5

FIELD 3 (before): 12 39 60 0D

FIELD 3 (after): 00 01 24 0D

The shift amount (three places) is specified in the D₂ field. The I₃ field specifies a rounding digit of 5. The rounding digit is added to the last digit shifted out (which is a 6), and the carry is propagated to the left. The sign is ignored during the addition.

Condition code 1 is set because the result is less than zero.

Multiplying by a Variable Power of 10

Since the shift value specified by the SRP instruction specifies both the direction and amount of the shift, the operation is equivalent to multiplying the decimal first operand by 10 raised to the power specified by the shift value.

If the shift value is to be variable, it may be specified by the B₂ field instead of the displacement D₂ of the SRP instruction. The general register designated by B₂ should contain the shift value (power of 10) as a signed binary integer.

A fixed scale factor modifying the variable power of 10 may be specified by using both the B₂ field (variable part in a general register) and the D₂ field (fixed part in the displacement).

The SRP instruction uses only the rightmost six bits of the effective address D₂(B₂) and interprets them as a six-bit signed binary integer to control the left or right shift as in the preceding shift examples.

ZERO AND ADD (ZAP)

Assume that the signed-packed-decimal number at storage locations 4500-4502 is to be moved to locations 4000-4004 with four leading zeros in the result field. Also assume:

Register 9 contains 00 00 40 00.

Storage locations 4000-4004 contain 12 34 56 78 90.

Storage locations 4500-4502 contain 38 46 0D.

After the instruction:

Machine Format

Op Code	L ₁	L ₂	B ₁	D ₁	B ₂	D ₂
F8	4	2	9	000	9	500

Assembler Format

Op Code D₁(L₁,B₁),D₂(L₂,B₂)

ZAP 0(5,9),X'500'(3,9)

is executed, the storage locations 4000-4004 contain 00 00 38 46 0D; condition code 1 is set to indicate a negative result without overflow.

Note that, because the first operand is not checked for valid sign and digit codes, it may contain any combination of hexadecimal digits before the operation.

Hexadecimal-Floating-Point Instructions

(See Chapter 9, "Floating-Point Overview and Support Instructions" for a complete description of the hexadecimal-floating-point instructions.)

In this section, the abbreviations FPR0, FPR2, FPR4, and FPR6 stand for floating-point registers 0, 2, 4, and 6 respectively.

ADD NORMALIZED (AD, ADR, AE, AER, AXR)

The ADD NORMALIZED instruction performs the addition of two HFP numbers and places the normal-

ized result in a floating-point register. Neither of the two numbers to be added must necessarily be in normalized form before addition occurs. For example, assume that:

FPR6 contains the unnormalized number C3 08 21 00 00 00 00 00 = $-82.1_{16} = -130.06_{10}$ approximately.

Storage locations 2000-2007 contain the normalized number 41 12 34 56 00 00 00 00 = $+1.23456_{16} = +1.14_{10}$ approximately.

Register 13 contains 00 00 20 00.

The instruction:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
7A	6	0	D	000

Assembler Format

Op Code R₁,D₂(X₂,B₂)

AE 6,0(0,13)

performs the short-precision addition of the two operands, as follows.

The characteristics of the two numbers (43 and 41) are compared. Since the number in storage has a characteristic that is smaller by 2, it is right-shifted two hexadecimal digit positions. One guard digit is retained on the right. The fractions of the two numbers are then added algebraically:

	Fraction GD ¹
FPR6	-43 08 21 00
Shifted number from storage	+43 00 12 34 5
Intermediate sum	-43 08 0E CB B
Left-shifted sum	-42 80 EC BB

¹ Guard digit

Because the intermediate sum is unnormalized, it is left-shifted to form the normalized HFP number $-80.ECBB_{16} = -128.92_{10}$ approximately. Combining the sign with the characteristic, the result is C2 80 EC BB, which replaces the left half of FPR6. The right half of FPR6 and the contents of storage locations 2000-2007 are unchanged. Condition code 1 is set to indicate a result less than zero.

If the long-precision instruction AD were used, the result in FPR6 would be C2 80 EC BA A0 00 00 00. Note that use of the long-precision instruction would avoid a loss of precision in this example.

ADD UNNORMALIZED (AU, AUR, AW, AWR)

The ADD UNNORMALIZED instruction operates the same as the ADD NORMALIZED instruction, except that the final result is not normalized. For example, using the same operands as in the example for ADD NORMALIZED, when the short-precision instruction:

Machine Format

Op Code	R ₁	X ₂	B ₂	D ₂
7E	6	0	D	000

Assembler Format

Op Code R₁,D₂(X₂,B₂)

AU 6,0(0,13)

is executed, the two numbers are added as follows:

	Fraction GD ¹
FPR6	-43 08 21 00
Shifted number from storage	+43 00 12 34 5
Intermediate sum	-43 08 0E CB B

¹ Guard digit

The guard digit participates in the addition but is discarded. The unnormalized sum replaces the left half of FPR6. Condition code 1 is set because the result is less than zero.

The truncated result in FPR6 (C3 08 0E CB 00 00 00 00) shows a loss of a significant digit when compared to the result of short-precision normalized addition.

COMPARE (CD, CDR, CE, CER)

Assume that FPR4 contains 43 00 00 00 00 00 00 00 (zero), and FPR6 contains 35 12 34 56 78 9A BC DE (a positive number). The contents of the two registers are to be compared using a long-precision COMPARE instruction.

Machine Format

Op Code	R ₁	R ₂
29	4	6

Assembler Format

Op Code R₁,R₂

CDR 4,6

The number with the smaller characteristic, which is in register FPR6, is right-shifted 43 - 35 hex (67 - 53 decimal) or 14 digit positions, so that the two characteristics agree. The shifted number is 43 00 00 00 00 00 00 00, with a guard digit of one. Therefore, when the two numbers are compared, condition code 1 is set, indicating that operand 1 in FPR4 is less than operand 2 in FPR6.

If the example is changed to a second operand with a characteristic of 34 instead of 35, so that FPR6 contains 34 12 34 56 78 9A BC DE, the operand is right-shifted 15 positions, leaving all fraction digits and the guard digit as zeros. Condition code 0 is set, indicating equality. This example shows that two HFP numbers with different characteristics or fractions may compare equal if the numbers are unnormalized or zero.

As another example of comparing unnormalized HFP numbers, 41 00 12 34 56 78 9A BC compares equal to all numbers of the form 3F 12 34 56 78 9A BC 0X (X represents any hexadecimal digit). When the COMPARE instruction is executed, the two rightmost digits are shifted right two places, the 0 becomes the guard digit, and the X does not participate in the comparison.

However, when two normalized HFP numbers are compared, the relationship between numbers that compare equal is unique: each digit in one number must be the same as the corresponding digit in the other number.

DIVIDE (DD, DDR, DE, DER)

Assume that the first operand (the dividend) is in FPR2 and the second operand (the divisor) in FPR0. If the operands are in the short-precision format, the resulting quotient is returned to FPR2 by the instruction:

Machine Format

Op Code	R ₁	R ₂
3D	2	0

Assembler Format

Op Code	R ₁ ,R ₂
DER	2,0

Several examples of short-precision HFP division, with the dividend in FPR2 and the divisor in FPR0, are shown below. For case A, the result, which replaces the dividend, is obtained in the following steps.

	7.2522F
.123400	821000
	7F6C00
	2A400 0
	24680 0
	5D80 00
	5B04 00
	27C 000
	246 800
	35 8000
	24 6800
	11 18000
	11 10C00
	7400

Case	FPR2 Before (Dividend)	FPR0 (Divisor)	FPR2 After (Quotient)
A	-43 082100	+43 001234	-42 72522F
B	+42 101010	+45 111111	+3D F0F0F0
C	+48 30000F	+41 400000	+47 C0003C
D	+48 30000F	+41 200000	+48 180007
E	+48 180007	+41 200000	+47 C00038

Case C shows a number being divided by 4.0. Case D divides the same number by 2.0, and case E divides the result of case D again by 2.0. The results of cases C and E differ in the rightmost hexadecimal digit position, which illustrates an effect of result truncation.

HALVE (HDR, HER)

HALVE produces the same result as HFP DIVIDE with a divisor of 2.0. Assume FPR2 contains the long-precision number +48 30 00 00 00 00 0F. The following HALVE instruction produces the result +48 18 00 00 00 00 07 in FPR2:

Machine Format

Op Code	R ₁	R ₂
24	2	2

Assembler Format

Op Code	R ₁ ,R ₂
HDR	2,2

MULTIPLY (MD, MDR, MDE, MDER, MXD, MXDR, MXR)

For this example, the following long-precision operands are in FPR0 and FPR2:

FPR0: -33 606060 60606060
FPR2: -5A 200000 20000020

A long-precision product is generated by the instruction:

Machine Format

Op Code	R ₁	R ₂
2C	0	2

Assembler Format

Op Code	R ₁ ,R ₂
MDR	0,2

If the operands were not already normalized, the instruction would first normalize them. It then generates an intermediate result consisting of the full 28-digit hexadecimal product fraction obtained by multiplying the 14-digit hexadecimal operand fractions, together with the appropriate sign and a characteristic that is the sum of the operand characteristics less 64 (40 hex):

The fraction multiplication is performed as follows:

```

      .60606060606060
      .20000020000020
      -----
      C0C0C0C0C0C0C0
    C0C0C0C0C0C0C0
    -----
    C0C0C0C0C0C0C0
  .0C0C0C181818241818180C0C0C00

```

Attaching the sign and characteristic to the fraction gives:

```
+4D 0C0C0C 18181824 1818180C 0C0C00
```

Because this intermediate product has a leading zero, it is then normalized. The truncated final result placed in FPR0 is:

```
+4C C0C0C1 81818241
```

Hexadecimal-Floating-Point-Number Conversion

The following examples illustrate one method of converting between binary fixed-point numbers (32-bit signed binary integers) and normalized HFP numbers. Conversion must provide for the different representations used with negative numbers: the two's-complement form for signed binary integers, and the signed-absolute-value form for the fractions of HFP numbers.

Fixed Point to Hexadecimal Floating Point

The method used here inverts the leftmost bit of the 32-bit signed binary integer, which is equivalent to adding 2^{31} to the number and considering the result to be positive. This changes the number from a signed integer in the range $2^{31} - 1$ through -2^{31} to an unsigned integer in the range $2^{32} - 1$ through 0. After conversion to the long HFP format, the value 2^{31} is subtracted again.

Assume that general register 9 (GR9) contains the integer -59 in two's-complement form:

```
GR9:    FF FF FF C5
```

Further, assume two eight-byte fields in storage: TEMP, for use as temporary storage, and TWO31,

which contains the floating-point constant 2^{31} in the following format:

```
TWO31:  4E 00 00 00 80 00 00 00
```

This is an unnormalized long HFP number with the characteristic 4E, which corresponds to a radix point (hexadecimal point) to the right of the number.

The following instruction sequence performs the conversion:

		Result
X	9,TWO31+4	GR9: 7FFF FFC5
ST	9,TEMP+4	TEMP: xxxx xxxx 7FFF FFC5
MVC	TEMP(4),TWO31	TEMP: 4E00 0000 7FFF FFC5
LD	2,TEMP	FPR2: 4E00 0000 7FFF FFC5
SD	2,TWO31	FPR2: C23B 0000 0000 0000

The EXCLUSIVE OR (X) instruction inverts the leftmost bit in general register 9, using the right half of the constant as the source for a leftmost one bit. The next two instructions assemble the modified number in an unnormalized long HFP format, using the left half of the constant as the plus sign, the characteristic, and the leading zeros of the fraction. LOAD (LD) places the number unchanged in floating-point register 2. The SUBTRACT NORMALIZED (SD) instruction performs the final two steps by subtracting 2^{31} in HFP form and normalizing the result.

Hexadecimal Floating Point to Fixed Point

The procedure described here consists basically in reversing the steps of the previous procedure. Two additional considerations must be taken into account. First: the HFP number may not be an exact integer. Truncating the excess hexadecimal digits on the right requires shifting the number one digit position farther to the right than desired for the final result, so that the units digit occupies the position of the guard digit. Second: the HFP number may have to be tested as to whether it is outside the range of numbers representable as a 32-bit signed binary integer.

Assume that floating-point register 6 contains the number $59.25_{10} = 3B.4_{16}$ in normalized form:

```
FPR6:    42 3B 40 00 00 00 00 00
```

Further, assume three eight-byte fields in storage: TEMP, for use as temporary storage, and the constants 2^{32} (TWO32) and 2^{31} (TWO31R) in the following formats:

```
TWO32:    4E 00 00 01 00 00 00 00
TWO31R:   4F 00 00 00 08 00 00 00
```

The constant TWO31R is shifted right one more position than the constant TWO31 of the previous example, so as to force the units digit into the guard-digit position.

The following instruction sequence performs the integer truncation, range tests, and conversion to a signed binary integer in general register 8 (GR8):

		Result
SD	6,TWO31R	FPR6: C87F FFFF C500 0000
BC	11,OVERFLOW	Branch to overflow routine if result is greater than or equal to zero
AW	6,TWO32	FPR6: 4E00 0000 8000 003B
BC	4,OVERFLOW	Branch to overflow routine if result is less than zero
STD	6,TEMP	TEMP: 4E00 0000 8000 003B
XI	TEMP+4,X'80'	TEMP: 4E00 0000 0000 003B
L	8,TEMP+4	GR8: 0000 003B

The SUBTRACT NORMALIZED (SD) instruction shifts the fraction of the number to the right until it lines up with TWO31R, which causes the fraction digit 4 to fall to the right of the guard digit and be lost; the result of subtracting 2^{31} from the remaining digits is renormalized. The result should be less than zero; if not, the original number was too large in the positive direction. The first BRANCH ON CONDITION (BC) performs this test.

The ADD UNNORMALIZED (AW) instruction adds 2^{32} : 2^{31} to correct for the previous subtraction and another 2^{31} to change to an all-positive range. The second BC tests for a result less than zero, showing that the original number was too large in the negative direction. The unnormalized result is placed in temporary storage by the STORE (STD) instruction. There the leftmost bit of the binary integer is inverted by the EXCLUSIVE OR (XI) instruction to subtract 2^{31}

and thus convert the unsigned number to the signed format. The final result is loaded into GR8.

Multiprogramming and Multiprocessing Examples

When two or more programs sharing common storage locations are being executed concurrently in a multiprogramming or multiprocessing environment, one program may, for example, set a flag bit in the common-storage area for testing by another program. It should be noted that the instructions AND (NI or NC), EXCLUSIVE OR (XI or XC), and OR (OI or OC) could be used to set flag bits in a multiprogramming environment; but the same instructions may cause program logic errors in a multiprocessing configuration where two or more CPUs can fetch, modify, and store data in the same storage locations simultaneously.

Example of a Program Failure Using OR Immediate

Assume that two independent programs try to set different bits to one in a common byte in storage. The following example shows how the use of the instruction OR immediate (OI) can fail to accomplish this, if the programs are executed simultaneously on two different CPUs. One of the possible error situations is depicted.

Execution of instruction OI FLAGS,X'01' on CPU A	FLAGS	Execution of instruction OI FLAGS,X'80' on CPU B
Fetch FLAGS X'00'	X'00'	Fetch FLAGS X'00'
OR X'01' into X'00'	X'00'	OR X'80' into X'00'
Store X'01' into FLAGS	X'80'	Store X'80' into FLAGS
FLAGS should have value of X'81' following both updates.		

The problem shown here is that the value stored by the OI instruction executed on CPU A overlays the value that was stored by CPU B. The X'80' flag bit was erroneously turned off, and the data is now invalid.

The COMPARE AND SWAP instruction has been provided to overcome this and similar problems.

Conditional Swapping Instructions (CS, CDS)

The COMPARE AND SWAP (CS) and COMPARE DOUBLE AND SWAP (CDS) instructions can be used in multiprogramming or multiprocessing environments to serialize access to counters, flags, control words, and other common storage areas.

The following examples of the use of the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions illustrate the applications for which the instructions are intended. It is important to note that these are examples of functions that can be performed by programs while the CPU is enabled for interruption (multiprogramming) or by programs that are being executed in a multiprocessing configuration. That is, the routine allows a program to modify the contents of a storage location while the CPU is enabled, even though the routine may be interrupted by another program on the same CPU that will update the location, and even though the possibility exists that another CPU may simultaneously update the same location.

The COMPARE AND SWAP instruction first checks the value of a storage location and then modifies it only if the value is what the program expects; normally this would be a previously fetched value. If the value in storage is not what the program expects, then the location is not modified; instead, the current value of the location is loaded into a general register, in preparation for the program to loop back and try again. During the execution of COMPARE AND SWAP, no other CPU can perform a store access or interlocked-update access at the specified location.

To ensure successful updating of a common storage field by two or more CPUs, all updates must be done by means of an interlocked-update reference. See the programming notes of COMPARE AND SWAP for an example of how COMPARE AND SWAP can be unsuccessful due to an OR IMMEDIATE instruction executed by another CPU.

Setting a Single Bit

The following instruction sequence shows how the COMPARE AND SWAP instruction can be used to set a single bit in storage to one. Assume that the first byte of a word in storage called "WORD" contains eight flag bits.

```

LA 6,X'80'    Put bit to be ORed into GR6
SLL 6,24      Shift left 24 places to
               align the byte to be ORed
               with the location of the
               flag bits within WORD
L 7,WORD      Fetch current flag values
RETRY LR 8,7   Load flags into GR8
OR 8,6        Set bit to one
CS 7,8,WORD   Store new flags if current
               flags unchanged, or re-
               fetch current flag values
               if changed
BC 4,RETRY    If new flags are not stored,
               try again
  
```

The format of the COMPARE AND SWAP instruction is:

Machine Format

Op Code	R ₁	R ₃	S ₂
BA	7	8	* * * *

Assembler Format

Op Code	R ₁ , R ₃ , S ₂
CS	7,8,WORD

The COMPARE AND SWAP instruction compares the first operand (general register 7 containing the current flag values) to the second operand in storage (WORD) while no CPU other than the one executing the COMPARE AND SWAP instruction is permitted to perform a store access or interlocked-update access at the specified storage location.

If the comparison is successful, indicating that the flag bits have not been changed since they were fetched, the modified copy in general register 8 is stored into WORD. If the flags have been changed, the compare will not be successful, and their new values are loaded into general register 7.

The conditional branch (BC) instruction tests the condition code and reexecutes the flag-modifying instructions if the COMPARE AND SWAP instruction indicated an unsuccessful comparison (condition code 1). When the COMPARE AND SWAP instruction is successful (condition code 0), the flags contain valid data, and the program exits from the loop.

The branch to RETRY will be taken only if some other program modifies the contents of WORD. This

type of a loop differs from the typical “bit-spin” loop. In a bit-spin loop, the program continues to loop until the bit changes. In this example, the program continues to loop only if the value does change during each iteration. If a number of CPUs simultaneously attempt to modify a single location by using the sample instruction sequence, one CPU will fall through on the first try, another will loop once, and so on until all CPUs have succeeded.

Updating Counters

In this example, a 32-bit counter is updated by a program using the COMPARE AND SWAP instruction to ensure that the counter will be correctly updated. The original value of the counter is obtained by loading the word containing the counter into general register 7. This value is moved into general register 8 to provide a modifiable copy, and general register 6 (containing an increment to the counter) is added to the modifiable copy to provide the updated counter value. The COMPARE AND SWAP instruction is used to ensure valid storing of the counter.

The program updating the counter checks the result by examining the condition code. The condition code 0 indicates a successful update, and the program can proceed. If the counter had been changed between the time that the program loaded its original value and the time that it executed the COMPARE AND SWAP instruction, the execution would have loaded the new counter value into general register 7 and set the condition code to 1, indicating an unsuccessful update. The program must then repeat the update sequence until the execution of the COMPARE AND SWAP instruction results in a successful update.

The following instruction sequence performs the above procedure:

```

        LA 6,1      Put increment (1) into GR6
        L 7,CNTR    Put original counter value
                    into GR7
LOOP LR 8,7        Set up copy in GR8 to modify
        AR 8,6      Increment copy
        CS 7,8,CNTR Update counter in storage
        BC 4,LOOP   If original value had
                    changed, update new value

```

The following shows two CPUs, A and B, executing this instruction sequence simultaneously: both CPUs attempt to add one to CNTR.

CPU A	CPU B	Comments
GR7 GR8 CNTR	GR7 GR8	
	16	
16 16		CPU A loads GR7 and GR8 from CNTR
	16 16	CPU B loads GR7 and GR8 from CNTR
	17	CPU B adds one to GR8
17		CPU A adds one to GR8
	17	CPU A executes CS; successful match, store
	17	CPU B executes CS; no match, GR7 changed to CNTR value
	18	CPU B loads GR8 from GR7, adds one to GR8
	18	CPU B executes CS; successful match, store

Bypassing Post and Wait

See the *MVS Programming: Authorized Assembler Services Guide* (SA22-7608) for details on bypassing the z/OS WAIT and POST routines.

Lock/Unlock

When a common storage area larger than a doubleword is to be updated, it is usually necessary to provide special interlocks to ensure that a single program at a time updates the common area. Such an area is called a serially reusable resource (SRR).

In general, updating a list, or even scanning a list, cannot be safely accomplished without first “freezing” the list. However, the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions can be used in certain restricted situations to perform queuing and list manipulation. Of prime importance is the capability to perform the lock/unlock functions and to provide sufficient queuing to resolve contentions, either in a LIFO or FIFO manner. The lock/unlock functions can then be used as the interlock mechanism for updating an SRR of any complexity.

The lock/unlock functions are based on the use of a “header” associated with the SRR. The header is the common starting point for determining the states of the SRR, either free or in use, and also is used for queuing requests when contentions occur. Conten-

tions are resolved using WAIT and POST. The general programming technique requires that the program that encounters a “locked” SRR must “leave a mark on the wall” indicating the address of an ECB on which it will WAIT. The “unlocking” program sees the mark and posts the ECB, thus permitting the waiting program to continue. In the two examples given, all programs using a particular SRR must use either the LIFO queuing scheme or the FIFO scheme; the two cannot be mixed. When more complex queuing is required, it is suggested that the queue for the SRR be locked using one of the two methods shown.

Lock/Unlock with LIFO Queuing for Contentions

The header consists of a word, that is, a four-byte field aligned on a word boundary. The word can contain zero, a positive value, or a negative value.

- A zero value indicates that the serially reusable resource (SRR) is free.
- A negative value indicates that the SRR is in use but no additional programs are waiting for the SRR.
- A positive value indicates that the SRR is in use and that one or more additional programs are waiting for the SRR. Each waiting program is identified by an element in a chained list. The positive value in the header is the address of the element most recently added to the list.

Each element consists of two words. The first word is used as an ECB; the second word is used as a pointer to the next element in the list. A negative value in a pointer indicates that the element is the last element in the list. The element is required only if the program finds the SRR locked and desires to be placed in the list.

The following chart describes the action taken for LIFO LOCK and LIFO UNLOCK routines. The rou-

tines following the chart allow enabled code to perform the actions described in the chart.

Function	Action		
	Header Contains Zero	Header Contains Positive Value	Header Contains Negative Value
LIFO LOCK (the incoming element is at location A)	SRR is free. Set the header to a negative value. Use the SRR.	SRR is in use. Store the contents of the header into location A+4. Store address A into the header. WAIT; the ECB is at location A.	
LIFO UNLOCK	Error	Some program is waiting for the SRR. Move the pointer from the “last in” element into the header. POST; the ECB is in the “last in” element.	The list is empty. Store zeros into the header. The SRR is free.

LIFO LOCK Routine:

Initial Conditions:

GR1 contains the address of the incoming element.

GR2 contains the address of the header.

```

LLOCK  SR    3,3      GR3 = 0
        ST    3,0(1)   Initialize the ECB
        LNR   0,1      GR0 = a negative value
TRYAGN CS    3,0,0(2)  Set the header to a
                        negative value if the
                        header contains zeros
        BC    8,USE     Did the header contain
                        zeros?
        ST    3,4(1)   No, store the value of
                        the header into the
                        pointer in the in-
                        coming element
        CS    3,1,0(2)  Store the address of
                        the incoming element
                        into the header
        LA    3,0(0)    GR3 = 0
        BC    7,TRYAGN  Did the header get up-
                        dated?
        WAIT  ECB=(1)   Yes, wait for the re-
                        source; the ECB is in
                        the incoming element
USE     [Any instruction]
```


LIFO UNLOCK Routine:

Initial Conditions:

GR2 contains the address of the header.

```
LUNLK L    1,0(2)    GR1 = the contents of
                     the header
A    LTR  1,1        Does the header contain
    BC   4,B         a negative value?
    L    0,4(1)      No, load the pointer
    CS   1,0,0(2)    from the oldest in
                     element and store it in
                     the header
    BC   7,A         Did the header get
                     updated?
    POST (1)         Yes, post the oldest in
                     element
    BC   15,EXIT     Continue
B    SR    0,0        The header contains a
    CS   1,0,0(2)    negative value; free
    BC   7,A         the header and continue
EXIT [Any instruction]
```

Note that the LOAD instruction L 1,0(2) at location LUNLK would have to be CS 1,1,0(2) if it were not for the rule concerning storage-operand consistency. This rule requires the LOAD instruction to fetch a four-byte operand aligned on a word boundary such that, if another CPU changes the word being fetched by an operation which is also at least word-consistent, either the entire new or the entire old value of the word is obtained, and not a combination of the two. (See "Storage-Operand Consistency" on page 5-95.)

Lock/Unlock with FIFO Queuing for Contentions

The header always contains the address of the most recently entered element. The header is originally initialized to contain the address of a posted ECB. Each program using the serially reusable resource (SRR) must provide an element regardless of whether contention occurs. Each program then enters the address of the element which it has provided into the header, while simultaneously it removes the address previously contained in the header. Thus, associated with any particular program attempting to use the SRR are two elements, called the "entered element" and the "removed element". The "entered element" of one program becomes the "removed element" for the immediately following program. Each program then waits on the removed element, uses the SRR, and then posts the entered element.

When no contention occurs, that is, when the second program does not attempt to use the SRR until after the first program is finished, then the POST of the first program occurs before the WAIT of the second program. In this case, the bypass-post and bypass-wait routines described in the preceding section are applicable. For simplicity, these two routines are shown only by name rather than as individual instructions.

In the example, the element need be only a single word, that is, an ECB. However, in actual practice, the element could be made larger to include a pointer to the previous element, along with a program identification. Such information would be useful in an error situation to permit starting with the header and chaining through the list of elements to find the program currently holding the SRR.

It should be noted that the element provided by the program remains pointed to by the header until the next program attempts to lock. Thus, in general, the entered element cannot be reused by the program. However, the removed element is available, so each program gives up one element and gains a new one. It is expected that the element removed by a particular program during one use of the SRR would then be used by that program as the entry element for the next request to the SRR.

It should be noted that, since the elements are exchanged from one program to the next, the elements cannot be allocated from storage that would be freed and reused when the program ends. It is expected that a program would obtain its first element and release its last element by means of the routines described in "Free-Pool Manipulation".

The following chart describes the action taken for FIFO LOCK and FIFO UNLOCK.

Function	Action
FIFO LOCK (the incoming element is at location A)	Store address A into the header. WAIT; the ECB is at the location addressed by the old contents of the header.
FIFO UNLOCK	POST; the ECB is at location A.

The following routines allow enabled code to perform the actions described in the previous chart.

FIFO Lock Routine:

Initial conditions:

GR3 contains the address of the header.

GR4 contains the address, A, of the element currently owned by this program. This element becomes the entered element.

```

FLOCK  LR   2,4      GR2 now contains
                     address of element to
                     be entered
        SR   1,1      GR1 = 0
        ST   1,0(2)   Initialize the ECB
        L    1,0(3)   GR1 = contents of the
                     header
TRYAGN CS   1,2,0(3)  Enter address A into
                     header while remember-
        BC   7,TRYAGN ing old contents of
                     header into GR1; GR1
                     now contains address
                     of removed element
        LR   4,1      Removed element becomes
                     new currently owned
                     element
        HSWAIT        Perform bypass-wait
                     routine; if ECB
                     already posted, con-
                     tinue; if not, wait;
                     GR1 contains the
                     address of the ECB
USE     [Any instruction]

```

FIFO Unlock Routine:

Initial conditions:

GR2 contains the address of the removed element, obtained during the FLOCK routine.

GR5 contains 40 00 00 00₁₆

```

FUNLK  LR   1,2      Place address of entered
                     element in GR1; GR1 = ad-
                     dress of ECB to be posted
        SR   0,0      GR0 = 0; GR0 has a post code
                     of zero
        OR   0,5      Set bit 1 of GR0 to one
        HSPOST        Perform bypass-post routine;
                     if ECB has not been waited
                     on, then mark posted and
                     continue; if it has been
                     waited on, then post
CONTINUE [Any instruction]

```

Free-Pool Manipulation

It is anticipated that a program will need to add and delete items from a free list without using the lock/unlock routines. This is especially likely since the lock/unlock routines require storage elements for queuing and may require working storage. The lock/unlock routines discussed previously allow simultaneous lock routines but permit only one unlock routine at a time. In such a situation, multiple additions and a single deletion to the list may all occur simultaneously, but multiple deletions cannot occur at the same time. In the case of a chain of pointers containing free storage buffers, multiple deletions along with additions can occur simultaneously. In this case, the removal cannot be done using the COMPARE AND SWAP instruction without a certain degree of exposure.

Consider a chained list of the type used in the LIFO lock/unlock example. Assume that the first two elements are at locations A and B, respectively. If one program attempted to remove the first element and was interrupted between the fourth and fifth instructions of the LUNLK routine, the list could be changed so that elements A and C are the first two elements when the interrupted program resumes execution. The COMPARE AND SWAP instruction would then succeed in storing the value B into the header, thereby destroying the list.

The probability of the occurrence of such list destruction can be reduced to *near* zero by appending to the header a counter that indicates the number of times elements have been added to the list. The use of a 32-bit counter guarantees that the list will not be destroyed unless the following events occur, in the exact sequence:

1. An unlock routine is interrupted between the fetch of the pointer from the first element and the update of the header.
2. The list is manipulated, including the deletion of the element referenced in 1, and exactly 2^{32} (or an integer multiple of 2^{32}) additions to the list are performed. Note that this takes on the order of days to perform in any practical situation.
3. The element referenced in 1 is added to the list.
4. The unlock routine interrupted in 1 resumes execution.

The following routines use such a counter in order to allow multiple, simultaneous additions and removals at the head of a chain of pointers.

The list consists of a doubleword header and a chain of elements. The first word of the header contains a pointer to the first element in the list. The second word of the header contains a 32-bit counter indicating the number of additions that have been made to the list. Each element contains a pointer to the next element in the list. A zero value indicates the end of the list.

The following chart describes the free-pool-list manipulation.

Function	Action	
	Header = 0,Count	Header = A,Count
ADD TO LIST (the incoming element is at location A)	Store the first word of the header into location A. Store the address A into the first word of the header. Decrement the second word of the header by one.	
DELETE FROM LIST	The list is empty.	Set the first word of the header to the value of the contents of location A. Use element A.

The following routines allow enabled code to perform the free-pool-list manipulation described in the above chart.

ADD TO FREE LIST Routine:

Initial Conditions:

GR2 contains the address of the element to be added.

GR4 contains the address of the header.

```

ADDQ  LM  0,1,0(4)  GR0,GR1 = contents of
                    the header
TRYAGN ST  0,0(2)   Point the new element to
                    the top of the list
        LR  3,1      Move the count to GR3
        BCTR 3,0      Decrement the count
        CDS  0,2,0(4) Update the header
        BC  7,TRYAGN

```

DELETE FROM FREE LIST Routine:

Initial conditions:

GR4 contains the address of the header.

```

DELETQ LM  2,3,0(4)  GR2,GR3 = contents
                    of the header
TRYAGN LTR  2,2      Is the list empty?
        BC  8,EMPTY  Yes, get help
        L   0,0(2)   No, GR0 = the pointer
                    from the first
                    element
        LR  1,3      Move the count to GR1
        CDS  2,0,0(4) Update the header
        BC  7,TRYAGN
USE  [Any instruction] The address of the re-
                    moved element is in
                    GR2

```

Note that the LM (LOAD MULTIPLE) instructions at locations ADDQ and DELETQ would have to be CDS (COMPARE DOUBLE AND SWAP) instructions if it were not for the rule concerning storage-operand consistency. This rule requires the LOAD MULTIPLE instructions to fetch an eight-byte operand aligned on a doubleword boundary such that, if another CPU changes the doubleword being fetched by an operation which is also at least doubleword-consistent, either the entire new or the entire old value of the doubleword is obtained, and not a combination of the two. (See "Storage-Operand Consistency" on page 5-95.)

PERFORM LOCKED OPERATION (PLO)

The PERFORM LOCKED OPERATION instruction can be used in a multiprogramming or multiprocessing environment to perform compare, load, compare-and-swap, and store operations on two or more discontinuous locations that can be words or doublewords. The operations are performed as an atomic set of operations under the control of a lock that is held only for the duration of the execution of a single PERFORM LOCKED OPERATION instruction, as opposed to across the execution of multiple instructions. Since lock contention is resolved by the CPU and is very brief, the program need not include a method for dealing with the case when the lock to be used is held by a program being executed by another CPU. Also, there need be no concern that the program may be interrupted while it holds a lock, since PERFORM LOCKED OPERATION will complete its operation and release its lock before an interruption can occur.

PERFORM LOCKED OPERATION can be thought of as performing concurrent interlocked updates of multiple operands. However, the instruction does not actually perform any interlocked update, and a serially reusable resource cannot be updated predictably through the use of both PERFORM LOCKED OPERATION and conditional-swapping instructions (CS and CDS).

Following is an example of how PERFORM LOCKED OPERATION can be used to add an element at the beginning of a queue.

Assume the following variables associated with the queue: S, which is a sequence number that is incremented anytime the queue is changed; H (for head), which is the address of the first element on the queue; and C, which is a count of the number of elements on the queue. Assume a queue element contains a variable, F (for forward), which is the address of the next element on the queue. If a new element, N, is to be enqueued at the head of the queue, that can be done by setting F in N to H and then performing the following atomic set of operations:

$$\begin{aligned} S+1 &\rightarrow S \\ A(N) &\rightarrow H \\ C+1 &\rightarrow C \end{aligned}$$

where A(N) is the address of N.

The enqueueing of N can be done by means of the following steps:

1. Obtain consistent values of S, H, and C, meaning obtain S and obtain the H and C that are consistent with that value of S.
2. Store H in N.F.
3. By means of PLO.csdst (PERFORM LOCKED OPERATION performing compare and swap and double store), with S as the swap variable and H and C as the store variables, add one to S, set H to A(N), and add one to C, provided that S still has the value obtained in step 1. If S has already been changed, go back to step 1.

Consistent values of S, H, and C cannot necessarily be obtained simply by using three LOAD instructions because a PERFORM LOCKED OPERATION instruction being executed by another CPU may have completed an update of S but not yet of H or C. In this case, the three LOAD instructions will obtain the

new S but the old H or C. However, as will be described, it may be possible to use three LOAD instructions.

If S is obtained while holding the lock, meaning by means of PERFORM LOCKED OPERATION, then H and C can be obtained by LOAD instructions since no other CPU can subsequently change H or C without changing S, as observed when the lock is held.

The parameter list used by the PLO.csdst is as follows, assuming the access-register mode is not used:

0	
8	
:	
48	
56	A(N)
64	
72	A(H)
80	
88	C+1
96	
104	A(C)

The program is as follows:

LA	RT,H	Initialize addresses in PL (T = temp)
ST	RT,PL+76	Op4 address (address of H)
LA	RT,C	
ST	RT,PL+108	Op6 address (address of C)
LA	RN,N	Address of N
ST	RN,PL+60	Initialize op3 in PL (address of N)
LA	R1,S	PLT address = address of S

SR	RS,RS	Dummy S. CC1 will probably be set
SR	R0,R0	Function code 0 (compare and load)
PLO	RS,S,RS,S	Obtain S while holding lock

LA	R0,16	Function code 16 (csdst)
L	RT,H	Consistent H
ST	RT,OFSTF(,RN)	OFSTF = offset of

```

                                F in N
L      RT,C      Consistent C
LA     RT,1(,RT) C+1
ST     RT,PL+92  Initialize op5 in PL
                        (C+1)
LA     RSP,1(,RS) RS/RSP = even-odd pair.
                        S+1 in RSP
PLO    RS,S,0,PL
BNZ    LOOP      Br if S changed (if CC
                        not 0)

```

Note the following about the first PERFORM LOCKED OPERATION instruction (PLO.cl). If S is not zero (which is probably true), S (the second operand, op2) is loaded into RS (the first-operand comparison value, op1c). If S is zero, S (the fourth operand, op4) is loaded into RS (the third operand, op3). Either of these loads occurs while the lock is held. It is unnecessary to test the condition code to determine which load occurred.

The above program may be a simplification. If the queue has associated with it a variable, T (for tail), that is the address of the last element on the queue, and the queue is currently empty, T also must be set when N is added to the queue. This would require a different program using a compare-and-swap-and-triple-store operation.

If the queue is added to, deleted from, and rearranged by means of PERFORM LOCKED OPERATION instructions in which the sequence number, S, is always the second operand, then, since the definition of PERFORM LOCKED OPERATION specifies that the second operand is always stored last, the first PERFORM LOCKED OPERATION instruction in the above program can be replaced by a LOAD instruction. The three instructions within the dashed lines would be replaced by L RS,S.

Sorting Instructions

Tree Format

Two instructions, COMPARE AND FORM CODE-WORD and UPDATE TREE, refer to a tree — a data structure with a specific format. A tree consists of some number (always odd) of consecutively numbered nodes. Node 1 is the root of the tree. Every node except the root has one parent node in the same tree. Every parent node has two son nodes.

Every even-numbered node is the *leftson* of its parent node, and every odd-numbered node (except node 1) is the *rightson* of its parent node. Division by two (ignoring remainder) of the node number gives the parent node number. Nodes with sons are also called internal nodes, and nodes without sons are called terminal nodes. Figure A-5 on page A-54 illustrates schematically a 21-node tree with arrows drawn from each parent node to each son node.

A tree is used for merging several sorted sequences of records into a single merged sequence of records. At each step in the merging process, there exists the initial part of the merged sequence and the remaining parts of each of the sorted sequences that are being merged. Each step consists in selecting the lowest record (the record with the lowest key when sorting in ascending sequence) from all of the as yet unmerged parts of the sorted sequences and adding it to the merged sequence. Each terminal node in the tree represents one of the sorted sequences. The number of internal nodes in the tree is one less than the number of sorted sequences. Each internal node conceptually contains one record from each of the sorted sequences but one; these are the lowest records, from all but one of the sorted sequences, that have not yet been added to the merged sequence. In addition, there is the lowest record from the one remaining sorted sequence. This additional record is compared and interchanged with nodes of the tree to select the record to be added next to the merged sequence. This processing begins with the parent of the terminal node that represents the one remaining sorted sequence, and it continues from that node along the path to the root of the tree. The selected record emerges from the root of the tree.

The tree may perhaps be most easily explained by considering each node to represent a comparison operation in an “elimination tournament” to find the lowest record. After the tournament has been completed, each node has an associated “loser” record which had a higher key in the comparison represented by that node. Besides a loser record at each node, there is one record (the “winner”) which is not associated with any node since it never compared high. The next step would be to introduce a new record from the same sorted sequence from which the winner record originated and replay the tournament with the new record in place of the former winner. It can be seen that it is unnecessary to do all the comparisons represented by all the nodes in the tree — most of them are unaffected by the new record replacing the former winner. In fact, it is suffi-

cient to redo only those node comparisons in which the former winner record participated. Each new record is inserted into the tree at the terminal node that represents the sorted sequence containing the record. The use of the tree assumes that programming provides a method of remembering at which terminal node each winning record originated. The instruction UPDATE TREE allows for a new record to be inserted at a terminal node and the tree to be updated so that a new winner record is left in the general registers.

Rather than comparing the actual keys of records, much of the merge logic can be performed using "codewords" to represent a record key rather than referring to actual keys. The value of a codeword at a node in the tree depends not only on the record's key but also on the key of the winning record in the last comparison at that node. The codeword consists of two parts:

1. Bits 16-31 contain the one's complement of the first halfword in which the record key differs from that of the node's winning record.
2. Bits 0-15 specify the byte offset of the halfword in this record's key just beyond the halfword value (complemented) in bit positions 16-31.

When comparing records in the path of the last winner record, if the new record is also represented by a codeword resulting from a comparison with the last winner, all codewords in the update path are with respect to the same winner. When comparing such codewords, a high codeword represents a low key and vice versa. Thus, when codewords are unequal, a node entry with a high codeword (representing a low actual key) should move up the tree.

In the case of a tie value of codewords, it is necessary to refer to the actual keys. This is done by the instruction COMPARE AND FORM CODEWORD, which resolves the ambiguity and computes a new codeword for the high-key (loser) record.

The eight bytes at each node of a tree consist of (1) a codeword for this record, computed with respect to the last record which compared low against this record and (2) a parameter usable to locate this record, for example, a direct or indirect address.

The instruction UPDATE TREE is so defined that tree updating stops after equal codewords are detected and the tie-breaking instruction COMPARE AND

FORM CODEWORD can be used, after which UPDATE TREE can resume tree updating at the point where equal codewords were previously found.

COMPARE AND FORM CODEWORD may alternatively be used for merging in descending sequence. In that case, bits 16-31 of the codeword at a node contain the true value of the first halfword in which the record key differs from that of the node's winning record. When the descending option of COMPARE AND FORM CODEWORD is used, the higher of two codewords represents the higher key.

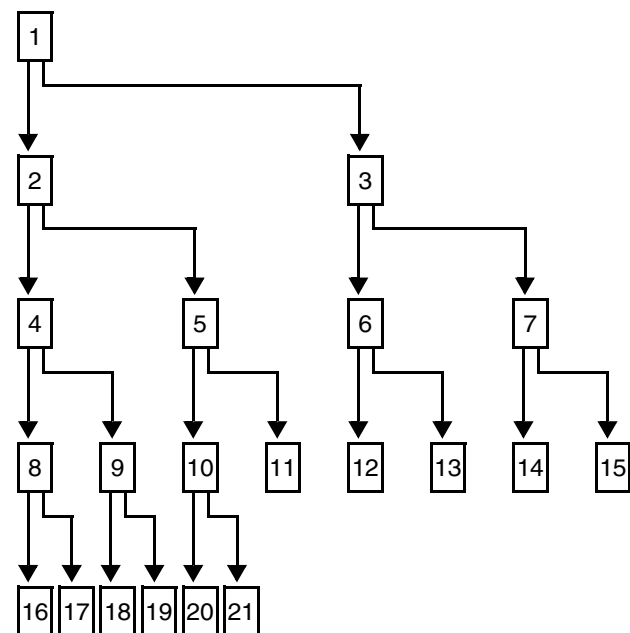


Figure A-5. Schematic Diagram of Merge Control Tree with 21 Nodes

Example of Use of Sort Instructions

An example illustrates how the instructions UPDATE TREE and COMPARE AND FORM CODEWORD may be used in the merge operation within a sort program. A five-way merge requires a tree data structure with four internal nodes and five terminal-node positions. The schematic diagram shown later in this section illustrates such a tree, containing four internal nodes (not counting the dummy node) and five input sequences for a merge, one sequence at each terminal-node position. Each record in an input sequence in the diagram is indicated by its address. The actual record contents are shown in Figure A-7 on

page A-58. Each record contains 16 bytes, consisting of the following fields:

Byte Offset (hexadecimal)	Field
0-5	Six-byte record key.
6-7	Halfword node index specifying the input sequence of the next record of this input sequence.
8-B	Address of the next record in the same input sequence.
C-F	This chaining field is initially zero. At the completion of the merge, this field is to contain the address of the next record in the merged sequence.

The merge process forms a single sorted sequence from five input sequences, each of which is in sorted order. This process can be subdivided into three steps:

1. A priming step takes the first record from each of the five input sequences and places them in the tree data structure. For each record to be introduced into the tree, first its codeword value is computed with respect to the lowest possible key value of all zeros. This codeword, with a second word which contains the address of the actual record, forms a doubleword node value that can be placed at the appropriate node. After priming, the node values, one each from each of the five input sequences, will have been placed in the tree so that each of the four internal nodes contains one node value and the node value for a winner record has emerged from the root of the tree.
2. After each winner emerges from the tree, the main merge process is performed repeatedly. Each iteration introduces the node value for one new record into the tree and produces a node value for a new winner record. The tree plus the winner must at all times contain precisely one node value from each input sequence being merged. Therefore, the new node value that is introduced into the tree on each iteration must come from the same input sequence from which the winner node value in the preceding iteration originated.
3. When the node value for the last record of an input sequence emerges as a winner, there is no successor record from that input sequence to be introduced into the tree on the next iteration. Hence, the order of the merge must be reduced

by one for each such occurrence. This runout process will consist of one or more iterations for each of a four-way, three-way, two-way, and one-way merge. The onset of runout occurs in the example when it is found that the next input record from a sequence is lower than its predecessor (a sequence break).

The priming process is discussed next, and the state of the tree is shown after priming is complete. Then, a short program that uses the instructions UPDATE TREE and COMPARE AND FORM CODEWORD to perform the main merge is described. An abbreviated trace is then presented to show the status of the tree and certain general registers for 16 iterations of the main merge. The runout process is not discussed in this example.

Priming begins by forming the node value for the first record of each input sequence. The first word of the node value is the codeword formed by executing COMPARE AND FORM CODEWORD on a record key containing all binary zeros. The second word of the node value is the address of the record represented by that node value. The node values for the first record of each input sequence are:

Sequence Index	Node Values
28	0006 FFFC 0000 1030
30	0006 FFFB 0000 1040
38	0006 FFFA 0000 1050
40	0004 FFFE 0000 1080
48	0006 FFF0 0000 1060

In the example, the tree data structure is assumed to have base address X'1000', which is kept in general register 4 (to match the expected use in UPDATE TREE). Similarly, internal-node index values and input-sequence index values are always used from general register 5.

Although the tree-priming program is not part of this example, the UPDATE TREE instruction is used in creating it as follows. First, the codeword position for each internal node of the tree is initialized to all ones (X'FFFF FFFF'). This artifice fills the tree with dummy low records. Then, for each record in the table, (1) the sequence index is loaded into general register 5, (2) the node value is loaded into general registers 0 and 1, and (3) UPDATE TREE is executed. At the completion of this priming process, the tree-node contents in the example are as shown on line 0 of Figure A-9 on page A-60. The contents of

the general registers are as shown on the first line of Figure A-8 on page A-59.

The figure illustrating the program for the main merge is divided into three groups of columns, containing the absolute program, the general-register trace, and the symbolic program. The first part of the program extends from symbolic locations L1 through L2; it introduces a new record into the tree and executes an UPDATE TREE instruction. If no tied codewords are encountered in UPDATE TREE, then the BRANCH ON CONDITION instruction following UPDATE TREE loops back to L1 to introduce the next record into the tree. This BRANCH ON CONDITION instruction is suitable for use when UPDATE TREE operates in accordance with either its method 1 (setting condition code 1) or its method 2 (setting condition code 3). (The preceding sentence applies to 370-XA. In ESA/370 and ESA/390, UPDATE TREE operates in accordance with only method 2, which is not to say that it cannot set condition code 1. Method 2, but not method 1, tests for the condition that sets condition code 3.)

If UPDATE TREE encounters tied codewords, then the UPDATE TREE instruction is completed, the subsequent BRANCH ON CONDITION instruction does not branch, and control falls through to the second part of the program, which handles entries with tied codewords. This part then branches back to UPDATE TREE at L2, which resumes the tree updating. It is possible for tied codewords to be encountered at any level in the tree (or indeed at all levels), so that the tied-codeword part of the program may be entered up to three times for each record introduced.

The general-register trace for the first part of the main merge shows the contents of the first seven general registers after each instruction is executed during the first iteration. Note that the merged-chain field (at 1140) serves as the anchor for the merged-chain address chain through the records. The trace shows only the lower half of certain general registers, whose upper half is always zero.

Figure A-9 on page A-60 gives an abbreviated trace of the entire main merge of 16 records. For each record introduced into the tree, there are one or more lines (always an odd number) given in the figure to show the tree updating, which results finally in a winner in GR0 and GR1. The first line for each record shows the values of GR5, GR2, and GR3 before the first or only execution of UPDATE TREE. For the even-numbered lines, the storage updating by

UPDATE TREE of tree nodes is shown (read left to right to follow the order of swapping). For example, consider line 10 and the corresponding UPDATE TREE: since GR5 contains 28, the first storage node examined is 1010 (refer to the schematic diagram). Since the codeword in GR0 is 0004 FFFE (same as for GR2), which is less than that of the word at 1010 (0006 FFF0), the doubleword at 1010 is swapped with that in GR0 and GR1. A second comparison at 1008 in the same execution of UPDATE TREE causes another register-storage doubleword swap, which leaves the winner (record 1040) in GR1 at the completion of UPDATE TREE (see the column at the far right of Figure A-9 on page A-60).

When a codeword comparison is made which does not result in a tie or a swap (that is, when the storage-codeword value is low), an asterisk appears in the trace for that storage entry.

When equal codewords are found, the execution of UPDATE TREE is completed. The following line in each such case shows the result of the tied-codeword routine, which always stores a new codeword and may also store a new record address before branching back to L2 to execute UPDATE TREE again. In this line, the notation “loses” or “wins” means that the node loses or wins, respectively.

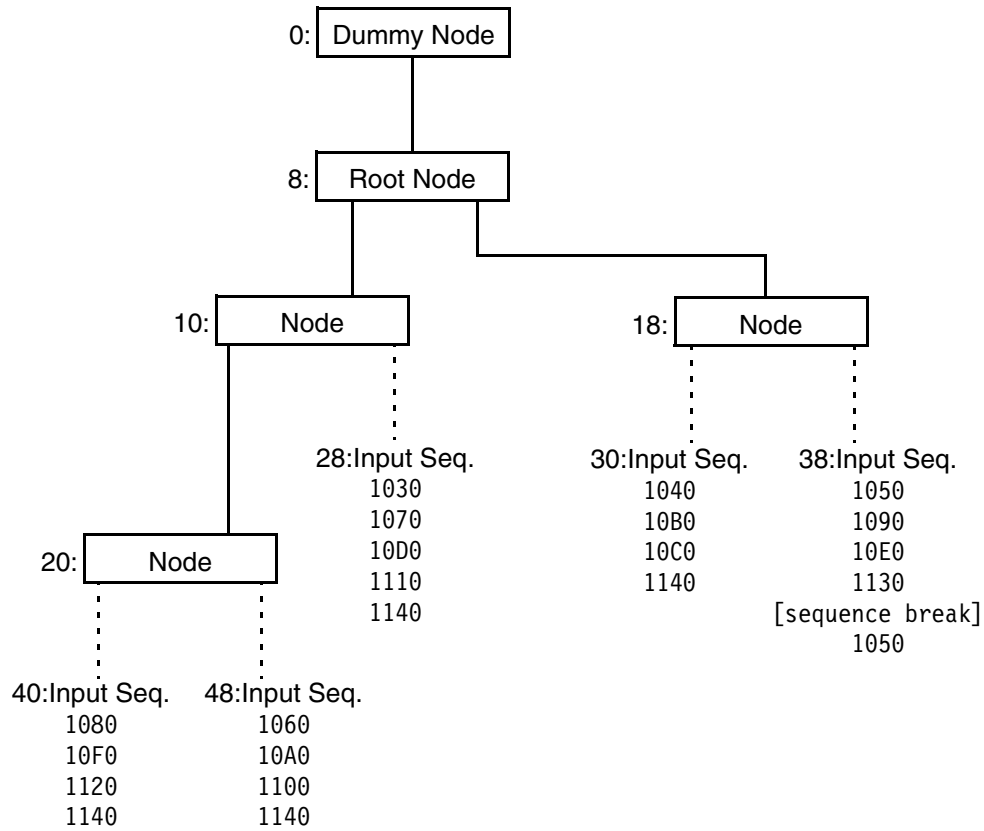
The tie-break trace part of Figure A-8 on page A-59 shows the treatment of the third record (that is, the first record for which UPDATE TREE encounters a tied codeword). This corresponds to line 31 in Figure A-9 on page A-60.

The following is a summary of the steps that are needed to use this example for verification purposes:

1. Initialize storage as follows:
 - a. 1008 through 102F from line 0 of Figure A-9 on page A-60
 - b. 1030 through 114F from Figure A-7 on page A-58
 - c. 1150 through 1189 from Figure A-8 on page A-59
2. Initialize GRs per first line in Figure A-8 and trace first record per Figure A-8.
3. Trace to completion of each UPT or BC 15,L2 (once for each line of Figure A-9). A detailed trace of the GRs for the tied-codeword part of

line 31 of Figure A-9 is given in the lower part of Figure A-8.

4. Verify that addresses in the chain beginning at 103C and continuing through 114C are as shown in the right-hand column of Figure A-7.



Note: Each node and input sequence is identified by a number which is the hexadecimal node index. Each input sequence is given as a list of record addresses (also in hexadecimal).

Figure A-6. Schematic Diagram for Example of Merge to Be Performed

Location	Record Key at Hex Byte Offset						Successor Record						Merged-Chain Address			
							Index		Location							
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1030	0 0 0 0	0 0 0 0	0 0 0 3	0 0 2 8	0 0 0 0	1 0 7 0	0 0 0 0	1 0 4 0								
1040	0 0 0 0	0 0 0 0	0 0 0 4	0 0 3 0	0 0 0 0	1 0 B 0	0 0 0 0	1 0 5 0								
1050	0 0 0 0	0 0 0 0	0 0 0 5	0 0 3 8	0 0 0 0	1 0 9 0	0 0 0 0	1 0 6 0								
1060	0 0 0 0	0 0 0 0	0 0 0 F	0 0 4 8	0 0 0 0	1 0 A 0	0 0 0 0	1 0 8 0								
1070	0 0 0 0	0 0 0 1	F F F F	0 0 2 8	0 0 0 0	1 0 D 0	0 0 0 0	1 0 9 0								
1080	0 0 0 0	0 0 0 1	F F F F	0 0 4 0	0 0 0 0	1 0 F 0	0 0 0 0	1 0 7 0								
1090	0 0 0 0	F F F F	0 0 0 0	0 0 3 8	0 0 0 0	1 0 E 0	0 0 0 0	1 0 A 0								
10A0	0 0 0 0	F F F F	0 0 0 1	0 0 4 8	0 0 0 0	1 1 0 0	0 0 0 0	1 0 B 0								
10B0	0 0 0 0	F F F F	0 0 0 2	0 0 3 0	0 0 0 0	1 0 C 0	0 0 0 0	1 0 C 0								
10C0	0 0 0 0	F F F F	0 0 0 2	0 0 3 0	0 0 0 0	1 1 4 0	0 0 0 0	1 0 D 0								
10D0	0 0 0 1	0 0 0 0	0 0 0 0	0 0 2 8	0 0 0 0	1 1 1 0	0 0 0 0	1 0 E 0								
10E0	0 0 8 0	0 0 0 0	0 0 0 0	0 0 3 8	0 0 0 0	1 1 3 0	0 0 0 0	1 0 F 0								
10F0	0 0 8 0	0 0 0 2	0 0 4 0	0 0 4 0	0 0 0 0	1 1 2 0	0 0 0 0	1 1 0 0								
1100	0 0 8 0	0 0 0 2	0 0 5 0	0 0 4 8	0 0 0 0	1 1 4 0	0 0 0 0	1 1 1 0								
1110	0 0 8 0	0 0 0 3	0 0 0 0	0 0 2 8	0 0 0 0	1 1 4 0	0 0 0 0	1 1 2 0								
1120	0 0 9 0	0 0 0 0	0 0 0 0	0 0 4 0	0 0 0 0	1 1 4 0	0 0 0 0	1 1 3 0								
1130	F F F F	F F F F	F F F E	0 0 3 8	0 0 0 0	1 0 5 0	0 0 0 0	0 0 0 0								
1140	F F F F	F F F F	F F F F	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 0 3 0								

Figure A-7. Contents of Records to Be Merged

Absolute		General-Register Trace							Symbolic Program	
Loc	INSTR	GR0	GR1	GR2	GR3	GR4	GR5	GR6	Loc	Instruction
		0006FFFC	1030		0000	1000	0000	1140		Using X'1000',4
1150	5010600C	↓	↓		↓	↓	↓	↓	L1	ST 1,12(,6) Store merged-chain address
1154	48501006	↓	↓		↓	↓	0028	↓	LH	5,6(,1) Load node index of input sequence of winner
1158	58301008	↓	↓		↓	↓	↓	↓	L	3,8(,1) Load successor-record address
115C	1861	↓	↓		↓	↓	↓	1030	LR	6,1 Save old winner address for next merged-chain store
115E	1B22	↓	↓	00000000	↓	↓	↓	↓	SR	2,2 Zero GR2 as initial offset
1160	B21A0004	↓	↓	0004FFFE	↓	↓	↓	↓	CFC	4 Compute codeword of new record based on new winner
1164	4720418A	↓	↓	↓	↓	↓	↓	↓	BC	2,L3 Exit on CC=2 (sequence break)
1168	1813	↓	1070	↓	↓	↓	↓	↓	LR	1,3 } Move new record entry to GRs 0-1
116A	1802	0004FFFE	↓	↓	↓	↓	↓	↓	LR	0,2 }
116C	0102	0006FFFB	1040	↓	↓	↓	0000	↓	L2	UPT Update tree data structure
116E	47504150	↓	↓	↓	↓	↓	↓	↓	BC	5,L1 If no codeword tie found, branch to next iteration
		00040000	1090	00040000	10B0	↓	0018	1050	*	Fall through on tied codewords
		↓	↓	↓	↓	↓	↓	↓	*	← GR values for tie-break trace
1172	88200010	↓	↓	00000004	↓	↓	↓	↓	SRL	2,16 Shift codeword offset to initial offset position for CFC
1176	B21A0004	↓	↓	0006FFFD	↓	↓	↓	↓	CFC	4 Compute loser codeword
117A	50254000	↓	↓	[CC=1]	↓	↓	↓	↓	ST	2,0(5,4) Store loser codeword in current storage node
117E	47C0416C	↓	↓	branch taken	↓	↓	↓	↓	BC	12,L2 Resume tree update if old storage-node entry is loser
1182	50354004	↓	↓	↓	↓	↓	↓	↓	ST	3,4(5,4) Store loser record address
1186	47F0416C	↓	↓	↓	↓	↓	↓	↓	BC	15,L2 Resume tree update
118A	...	↓	↓	↓	↓	↓	↓	↓	L3	... Control reaches here at end

Figure A-8. Program for Main Merge

L#	General Regs after CFC at Location 1160				Storage Trace of Node Entries								General Regs after UPT or BC 15,L2	
	GR5	GR2	GR3	Comment	1020		1018		1010		1008		GR0	GR1
0 ¹					0004FFFE	1080	0006FFFA	1050	0006FFF0	1060	0006FFFB	1040	0006FFFC	1030
10	28	0004FFFE	1070	No tie					0004FFFE	1070	0006FFF0	1060	0006FFFB	1040
20	30	00040000	10B0	No tie			00040000	10B0			*		0006FFFA	1050
30	38	00040000	1090	CC = 0			Tie						00040000	1090
31				Loses			0006FFFD						00040000	1090
32				No tie							00040000	1090	0006FFF0	1060
40	48	00040000	10A0	CC = 0	00040000	10A0			Tie				0004FFFE	1080
41				Equal					80001070				0004FFFE	1080
42				No tie							*		0004FFFE	1080
50	40	0002FF7F	10F0	No tie	0002FF7F	10F0			00040000	10A0	**		80001070	1070
60	28	0002FFFE	10D0	CC = 0					0002FFFE	10D0	Tie		00040000	10A0
61				Wins							0006FFFE	10A0	00040000	1090
62				No comp									00040000	1090
70	38	0002FF7F	10E0	No tie			0002FF7F	10E0			0006FFFD	10B0	0006FFFE	10A0
80	48	0002FF7F	1100	CC = 0	Tie								0002FF7F	1100
81				Wins	0006FFAF	1100							0002FF7F	10F0
82				No tie					0002FF7F	10F0	0002FFFE	10D0	0006FFFD	10B0
90	30	800010C0	10C0	No tie			*				*		800010C0	10C0
100	30	00020000	1140	No tie			00020000	1140			0002FF7F	10E0	0002FFFE	10D0
110	28	0002FF7F	1110	CC = 0					Tie				0002FF7F	1110
111				Wins					0004FFFC	1110			0002FF7F	10F0
112				CC = 0							Tie		0002FF7F	10F0
113				Wins							0004FFFD	10F0	0002FF7F	10E0
114				No comp									0002FF7F	10E0
120	38	00020000	1130	CC = 0			Tie						00020000	1130
121				Loses			00060000						00020000	1130
122				No tie							00020000	1130	0004FFFD	10F0
130	40	0002FF6F	1120	No tie	0002FF6F	1120			*		*		0006FFAF	1100
140	48	00020000	1140	No tie	00020000	1140			0002FF6F	1120	*		0004FFFC	1110
150	28	00020000	1140	No tie					00020000	1140	*		0002FF6F	1120
160	40	00020000	1140	CC = 0	Tie								00020000	1140
161				Equal	80001140								00020000	1140
162				CC = 0					Tie				00020000	1140
163				Equal					80001140				00020000	1140
164				CC = 0									00020000	1140
165				Wins							Tie		00020000	1140
166				No comp							00060000	1140	00020000	1130
170	38	00020000	1050	Branch										

Explanation:

¹ Line 0 shows the values in the tree after it is primed.

* Means no swap.

** Means no swap if UPDATE TREE method 1 is used or no examination if UPDATE TREE method 2 is used. Only method 2 is included in ESA/370 and ESA/390.

CC = 0 UPDATE TREE finds a tie and sets condition code 0.

Loses The tied-codeword routine finds that the node loses.

Wins The tied-codeword routine finds that the node wins.

Equal The tied-codeword routine finds that the keys are equal.

Branch Branches to terminate at 118A on sequence break.

No comp No compare.

Figure A-9. Abbreviated Trace of Main Merge Processing

Appendix B. Lists of Instructions

The following figures list instructions by name, mnemonic, and operation code. Some models may offer instructions that do not appear in the figures, such as those provided for assists or as part of special or custom features.

The operation code for the interpretive execution facility is not included in this appendix. See the publication *IBM System/370 Extended Architecture Interpretive Execution*, SA22-7095, for the operation code associated with that facility.

The operation code 00 hex with a two-byte instruction format is allocated for use by the program when an indication of an invalid operation is required. Operation code 00 hex will never be assigned to an instruction implemented in the CPU.

Explanation of Symbols in “Characteristics” and “Page” Columns:

\$	Causes serialization.
¢	Causes serialization and checkpoint synchronization.
¢ ¹	Causes serialization and checkpoint synchronization when the M ₁ and R ₂ fields contain all ones and all zeros, respectively.
¢ ²	Causes serialization and checkpoint synchronization when the state entry to be unstacked is a program-call state entry.
¢ ³	Causes serialization and checkpoint synchronization when the set-key control is one.
A	Access exceptions for logical addresses.
A ¹	Access exceptions; not all access exceptions may occur; see instruction description for details.
AI	Access exceptions for instruction address.
AS	ASN-translation-specification and special-operation exceptions.
AT	ASN-translation-specification exception.
B	PER branch event.
B ₁	B ₁ field designates an access register in the access-register mode.
B ₂	B ₂ field designates an access register in the access-register mode.
B ₄	B ₄ field designates an access register in the access-register mode.

B†	B ₁ field designates an access register when bit 47 of GR0 is zero, and bits 16-17 of the current PSW are 01 binary; or when bit 47 of GR0 is one, and bits 40-41 of GR0 are 01 binary.
B‡	B ₂ field designates an access register when bit 63 of GR0 is zero, and bits 16-17 of the current PSW are 01 binary; or when bit 63 of GR0 is one, and bits 56-57 of GR0 are 01 binary.
BP	B ₂ field designates an access register when PSW bits 16 and 17 have the value 01.
C	Condition code is set.
CL	Conditional-load/store facility
CS	Compare-and-swap-and-store facility.
CT	Configuration-topology facility.
D2	DAT-enhancement facility 2.
Da	AFP-register data exception.
Db	BFP-instruction data exception.
Dc	Compare-and-trap data exception.
Dd	Decimal-operand data exception.
DE	DAT-enhancement facility.
DF	Decimal-overflow exception.
DK	Decimal-divide exception.
DM	Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code.
DO	Distinct-operands facility
Dt	DFP-instruction data exception.
E	E instruction format.
E2	Extended-translation facility 2.
E3	Extended-translation facility 3.
ED	Enhanced-DAT facility.
EI	Extended-immediate facility.
EO	HFP-exponent-overflow exception.
ES	Expanded-storage facility.
ET	Extract-CPU-time facility.
EU	HFP-exponent-underflow exception.
EX	Execute exception.
FC	Designation of access registers depends on the function code of the instruction.
FG	FPR-GR-transfer facility.
FK	HFP-floating-point-divide exception.
FL	Store-facility-list-extended facility.

FS	Floating-point-support-sign-handling facility.	PF	PFPO facility.
G0	Instruction execution includes the implied use of general register 0.	Q	Privileged-operation exception for semiprivileged instructions.
G1	Instruction execution includes the implied use of general register 1.	R ₁	R ₁ field designates an access register in the access-register mode.
G2	Instruction execution includes the implied use of general register 2.	R ₂	R ₂ field designates an access register in the access-register mode.
G4	Instruction execution includes the implied use of general register 4.	R ₃	R ₃ field designates an access register in the access-register mode.
GE	General-instructions-extension facility.	RA	Reusable-ASN-and-LX facility.
GM	Instruction execution includes the implied use of multiple general registers.	RB	Reset-reference-bits-multiple facility.
GS	Instruction execution includes the implied use of general register 1 as the subsystem-identification word.	RI	RI instruction format.
HM	HFP-multiply-add/subtract facility.	RIE	RIE instruction format.
HW	High-word facility	RIL	RIL instruction format.
I	I instruction format.	RIS	RIS instruction format.
I1	Access register 1 is implicitly designated in the access-register mode.	RM	R ₁ field designates an access register in the access-register mode, and access-register 1 also is used in the access-register mode.
I4	Access register 4 is implicitly designated in the access-register mode.	RR	RR instruction format.
IC	Condition code alternative to interruptible instruction	RRE	RRE instruction format.
IF	Fixed-point-overflow exception.	RRF	RRF instruction format.
IA	Interlocked-access facility	RRR	RRR instruction format.
II	Interruptible instruction.	RRS	RRS instruction format.
IK	Fixed-point-divide exception.	RS	RS instruction format.
IS	Interruptible instruction and special-operation exception.	RSI	RSI instruction format.
L	New condition code is loaded.	RSL	RSL instruction format.
LD	Long-displacement facility.	RSY	RSY instruction format.
LS	HFP-significance exception.	RX	RX instruction format.
MD	Designation of access registers in the access-register mode is model-dependent.	RXE	RXE instruction format.
ME	Monitor event.	RXF	RXF instruction format.
MO	Move-with-optional-specifications facility.	RXY	RXY instruction format.
MS	Message-security assist.	S	S instruction format.
M3	Message-security assist extension 3.	SC	Store-clock-fast facility.
M4	Message-security assist extension 4.	SE	Special operation, stack-empty, stack-specification, and stack-type exceptions.
N	Instruction is new in z/Architecture as compared to ESA/390.	SF	Special-operation, stack-full, and stack-specification exceptions.
N3	Instruction is new in z/Architecture and has been added to ESA/390.	SI	SI instruction format.
OP	Operand exception.	SIL	SIL instruction format.
P	Privileged-operation exception.	SIY	SIY instruction format.
PC	Population-count facility	SO	Special-operation exception.
PE	Parsing-enhancement facility.	SP	Specification exception.
		SQ	HFP-square-root exception.
		SS	SS instruction format.
		SSE	SSE instruction format.
		SSF	SSF instruction format.
		ST	PER storage-alteration event.

SU	PER store-using-real-address event.	Z ⁵	Additional exceptions for BRANCH AND STACK (which include special operation, stack full, and stack specification)
SW	Special-operation exception and space-switch event.		
T	Trace exceptions (which include trace table, addressing, and low-address protection).	Z ⁶	Additional exceptions and events for PROGRAM TRANSFER WITH INSTANCE (which include AFX-translation, ASTE-instance, ASX-translation, primary-authority, special-operation, and subspace-replacement exceptions and space-switch event).
TF	Decimal-floating-point facility.		
TR	Decimal-floating-point-rounding facility.		
TS	TOD-clock-steering facility.		
U	Condition code is unpredictable.	Z ⁷	Additional exceptions for SET SECONDARY ASN WITH INSTANCE (which include AFX translation, ASTE instance, ASX translation, secondary authority, special operation, and subspace replacement).
U ₁	R ₁ field designates an access register unconditionally.		
U ₂	R ₂ field designates an access register unconditionally.		
UB	R ₁ and R ₃ fields designate access registers unconditionally, and B ₂ field designates an access register in the access-register mode.		
UE	HFP unnormalized-extensions facility.		
WE	Space-switch event.		
XF	IEEE-exception-simulation facility.		
Xg	Simulated IEEE exception.		
Xi	IEEE invalid-operation exception.		
Xo	IEEE overflow exception.		
Xu	IEEE underflow exception.		
XX	Execute-extension facility.		
Xx	IEEE inexact exception.		
Xz	IEEE division-by-zero exception.		
Z ¹	Additional exceptions and events for PROGRAM CALL (which include ASX-translation, EX-translation, LX-translation, PC-translation-specification, special-operation, stack-full, and stack-specification exceptions and space-switch event).		
Z ²	Additional exceptions and events for PROGRAM TRANSFER (which include AFX-translation, ASX-translation, primary-authority, and special-operation exceptions and space-switch event).		
Z ³	Additional exceptions for SET SECONDARY ASN (which include AFX translation, ASX translation, secondary authority, and special operation).		
Z ⁴	Additional exceptions and events for PROGRAM RETURN (which include AFX-translation, ASX-translation, secondary-authority, special-operation, stack-empty, stack-operation, stack-specification, and stack-type exceptions and space-switch event).		

Instructions Arranged by Name

Name	Mne- monic	Characteristics						Op- code	Page
ADD (32)	A	RX-a C	A	IF			B ₂	5A	7-25
ADD (32)	AR	RR C		IF				1A	7-24
ADD (32)	ARK	RRF-a C DO		IF				B9F8	7-25
ADD (32)	AY	RXY-a C LD	A	IF			B ₂	E35A	7-25
ADD (64)	AG	RXY-a C N	A	IF			B ₂	E308	7-25
ADD (64)	AGR	RRE C N		IF				B908	7-25
ADD (64)	AGRK	RRF-a C DO		IF				B9E8	7-25
ADD (64←32)	AGF	RXY-a C N	A	IF			B ₂	E318	7-25
ADD (64←32)	AGFR	RRE C N		IF				B918	7-25
ADD (extended BFP)	AXBR	RRE C	SP	Db Xi Xo Xu Xx				B34A	19-11
ADD (extended DFP)	AXTR	RRF-a C TF	SP	Dt Xi Xo Xu Xx				B3DA	20-19
ADD (extended DFP)	AXTRA	RRF-a C F	SP	Dt Xi Xo Xu Xx Xq				B3DA	20-19
ADD (long BFP)	ADB	RXE C	A	Db Xi Xo Xu Xx			B ₂	ED1A	19-11
ADD (long BFP)	ADBR	RRE C		Db Xi Xo Xu Xx				B31A	19-11
ADD (long DFP)	ADTR	RRF-a C TF		Dt Xi Xo Xu Xx				B3D2	20-19
ADD (long DFP)	ADTRA	RRF-a C F		Dt Xi Xo Xu Xx Xq				B3D2	20-19
ADD (short BFP)	AEB	RXE C	A	Db Xi Xo Xu Xx			B ₂	ED0A	19-11
ADD (short BFP)	AEBR	RRE C		Db Xi Xo Xu Xx				B30A	19-11
ADD DECIMAL	AP	SS-b C	A	Dd DF	ST	B ₁	B ₂	FA	8-6
ADD HALFWORD	AH	RX-a C	A	IF			B ₂	4A	7-27
ADD HALFWORD	AHY	RXY-a C LD	A	IF			B ₂	E37A	7-27
ADD HALFWORD IMMEDIATE (32)	AHI	RI-a C		IF				A7A	7-27
ADD HALFWORD IMMEDIATE (64)	AGHI	RI-a C N		IF				A7B	7-27
ADD HIGH (32)	AHHHR	RRF-a C HW		IF				B9C8	7-27
ADD HIGH (32)	AHHLR	RRF-a C HW		IF				B9D8	7-27
ADD IMMEDIATE (32)	AFI	RIL-a C EI		IF				C29	7-25
ADD IMMEDIATE (32←16)	AHIK	RIE-d C DO		IF				ECD8	7-25
ADD IMMEDIATE (32←8)	ASI	SIY C GE	A	IF		ST	B ₁	EB6A	7-25
ADD IMMEDIATE (64←16)	AGHIK	RIE-d C DO		IF				ECD9	7-25
ADD IMMEDIATE (64←32)	AGFI	RIL-a C EI		IF				C28	7-25
ADD IMMEDIATE (64←8)	AGSI	SIY C GE	A	IF		ST	B ₁	EB7A	7-25
ADD IMMEDIATE HIGH (32)	AIH	RIL-a C HW		IF				CC8	7-25
ADD LOGICAL (32)	AL	RX-a C	A				B ₂	5E	7-28
ADD LOGICAL (32)	ALR	RR C						1E	7-28
ADD LOGICAL (32)	ALRK	RRF-a C DO						B9FA	7-28
ADD LOGICAL (32)	ALY	RXY-a C LD	A				B ₂	E35E	7-28
ADD LOGICAL (64)	ALG	RXY-a C N	A				B ₂	E30A	7-28
ADD LOGICAL (64)	ALGR	RRE C N						B90A	7-28
ADD LOGICAL (64)	ALGRK	RRF-a C DO						B9EA	7-28
ADD LOGICAL (64←32)	ALGF	RXY-a C N	A				B ₂	E31A	7-28
ADD LOGICAL (64←32)	ALGFR	RRE C N						B91A	7-28
ADD LOGICAL HIGH (32)	ALHHHR	RRF-a C HW						B9CA	7-29
ADD LOGICAL HIGH (32)	ALHHLR	RRF-a C HW						B9DA	7-29
ADD LOGICAL IMMEDIATE (32)	ALFI	RIL-a C EI						C2B	7-28
ADD LOGICAL IMMEDIATE (64←32)	ALGFI	RIL-a C EI						C2A	7-28
ADD LOGICAL WITH CARRY (32)	ALC	RXY-a C N3	A				B ₂	E398	7-29
ADD LOGICAL WITH CARRY (32)	ALCR	RRE C N3						B998	7-29
ADD LOGICAL WITH CARRY (64)	ALCG	RXY-a C N	A				B ₂	E388	7-29
ADD LOGICAL WITH CARRY (64)	ALCGR	RRE C N						B988	7-29
ADD LOGICAL WITH SIGNED IMMEDIATE (32←16)	ALHSIK	RIE-d C DO						ECDA	7-30
ADD LOGICAL WITH SIGNED IMMEDIATE (32←8)	ALSI	SIY C GE	A			ST	B ₁	EB6E	7-30

Figure B-1. Instructions Arranged by Name (Part 1 of 18)

Name	Mne- monic	Characteristics						Op- code	Page						
ADD LOGICAL WITH SIGNED IMMEDIATE (64←16)	ALGHSIK	RIE-d	C	DO	A		ST	B ₁	ECDB	7-30					
ADD LOGICAL WITH SIGNED IMMEDIATE (64←8)	ALGSI	SIY	C	GE					EB7E	7-30					
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	ALSIH	RIL-a	C	HW					CCA	7-31					
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	ALSIHN	RIL-a		HW					CCB	7-31					
ADD NORMALIZED (extended HFP)	AXR	RR	C		A	SP		B ₂	36	18-8					
ADD NORMALIZED (long HFP)	AD	RX-a	C						Da	EU	EO	LS	6A	18-8	
ADD NORMALIZED (long HFP)	ADR	RR	C		A			B ₂	2A	18-8					
ADD NORMALIZED (short HFP)	AE	RX-a	C						Da	EU	EO	LS	7A	18-8	
ADD NORMALIZED (short HFP)	AER	RR	C						Da	EU	EO	LS	3A	18-8	
ADD UNNORMALIZED (long HFP)	AW	RX-a	C		A			B ₂	6E	18-9					
ADD UNNORMALIZED (long HFP)	AWR	RR	C						Da		EO	LS	2E	18-9	
ADD UNNORMALIZED (short HFP)	AU	RX-a	C		A			B ₂	7E	18-9					
ADD UNNORMALIZED (short HFP)	AUR	RR	C						Da		EO	LS	3E	18-9	
AND (32)	N	RX-a	C		A			B ₂	54	7-31					
AND (32)	NR	RR	C									14	7-31		
AND (32)	NRK	RRF-a	C	DO					A			B ₂	B9F4	7-31	
AND (32)	NY	RXY-a	C	LD									A	E354	7-32
AND (64)	NG	RXY-a	C	N	A			B ₂	E380	7-32					
AND (64)	NGR	RRE	C	N								B980	7-31		
AND (64)	NGRK	RRF-a	C	DO	A		ST	B ₁ B ₂	B9E4	7-31					
AND (character)	NC	SS-a	C									D4	7-32		
AND (immediate)	NI	SI	C						A			94	7-32		
AND (immediate)	NIY	SIY	C	LD					A			EB54	7-32		
AND IMMEDIATE (high high)	NIHH	RI-a	C	N					A54	7-32					
AND IMMEDIATE (high low)	NIHL	RI-a	C	N								A55	7-33		
AND IMMEDIATE (high)	NIHF	RIL-a	C	EI								C0A	7-32		
AND IMMEDIATE (low high)	NILH	RI-a	C	N								A56	7-33		
AND IMMEDIATE (low low)	NILL	RI-a	C	N								A57	7-33		
AND IMMEDIATE (low)	NILF	RIL-a	C	EI								C0B	7-33		
BRANCH AND LINK	BAL	RX-a					B		45	7-33					
BRANCH AND LINK	BALR	RR							T	B		05	7-33		
BRANCH AND SAVE	BAS	RX-a								B		4D	7-34		
BRANCH AND SAVE	BASR	RR							T	B		0D	7-34		
BRANCH AND SAVE AND SET MODE	BASSM	RR							T	B		0C	7-35		
BRANCH AND SET AUTHORITY	BSA	RRE		Q A ¹	A ¹	SO	T	B		B25A	10-6				
BRANCH AND SET MODE	BSM	RR								T	B		0B	7-36	
BRANCH AND STACK	BAKR	RRE								Z ⁵	T	B ST	B240	10-10	
BRANCH IN SUBSPACE GROUP	BSG	RRE								SO	T	B	R ₂	B258	10-13
BRANCH ON CONDITION	BC	RX-b										B		47	7-36
BRANCH ON CONDITION	BCR	RR									c ¹	B		07	7-36
BRANCH ON COUNT (32)	BCT	RX-a			A ¹		B		46	7-37					
BRANCH ON COUNT (32)	BCTR	RR								B		06	7-37		
BRANCH ON COUNT (64)	BCTG	RXY-a		N						B		E346	7-37		
BRANCH ON COUNT (64)	BCTGR	RRE		N						B		B946	7-37		
BRANCH ON INDEX HIGH (32)	BXH	RS-a			A ¹		B		86	7-38					
BRANCH ON INDEX HIGH (64)	BXHG	RSY-a		N						B		EB44	7-38		
BRANCH ON INDEX LOW OR EQUAL (32)	BXLE	RS-a								B		87	7-38		
BRANCH ON INDEX LOW OR EQUAL (64)	BXLEG	RSY-a		N						B		EB45	7-38		
BRANCH RELATIVE AND SAVE	BRAS	RI-b								B		A75	7-39		
BRANCH RELATIVE AND SAVE LONG	BRASL	RIL-b		N3	A ¹		B		C05	7-39					
BRANCH RELATIVE ON CONDITION	BRC	RI-c								B		A74	7-40		
BRANCH RELATIVE ON CONDITION LONG	BRCL	RIL-c		N3						B		C04	7-40		
BRANCH RELATIVE ON COUNT (32)	BRCT	RI-b								B		A76	7-41		
BRANCH RELATIVE ON COUNT (64)	BRCTG	RI-b		N						B		A77	7-41		
BRANCH RELATIVE ON COUNT HIGH (32)	BRCTH	RIL-b		HW			B		CC6	7-41					

Figure B-1. Instructions Arranged by Name (Part 2 of 18)

Name	Mne- monic	Characteristics						Op- code	Page
BRANCH RELATIVE ON INDEX HIGH (32)	BRXH	RSI				B		84	7-41
BRANCH RELATIVE ON INDEX HIGH (64)	BRXHG	RIE-e	N			B		EC44	7-41
BRANCH RELATIVE ON INDEX LOW OR EQ. (32)	BRXLE	RSI				B		85	7-41
BRANCH RELATIVE ON INDEX LOW OR EQ. (64)	BRXLG	RIE-e	N			B		EC45	7-42
CANCEL SUBCHANNEL	XSCH	S	C	P	OP	¢	GS	B276	14-4
CHECKSUM	CKSM	RRE	C	A	SP	IC		R ₂	B241 7-43
CIPHER MESSAGE	KM	RRE	C	MS	A	SP	IC	GM I1	ST R ₁ R ₂ B92E 7-46
CIPHER MESSAGE WITH CFB	KMF	RRE	C	M4	A	SP	IC	GM I1	ST R ₁ R ₂ B92A 7-71
CIPHER MESSAGE WITH CHAINING	KMC	RRE	C	MS	A	SP	IC	GM I1	ST R ₁ R ₂ B92F 7-46
CIPHER MESSAGE WITH COUNTER	KMCTR	RRF-b	C	M4	A	SP	IC	GM I1	ST R ₁ ,R ₂ ,R ₃ B92D 7-86
CIPHER MESSAGE WITH OFB	KMO	RRE	C	M4	A	SP	IC	GM I1	ST R ₁ R ₂ B92B 7-98
CLEAR SUBCHANNEL	CSCH	S	C	P	OP	¢	GS		B230 14-5
COMPARE (32)	C	RX-a	C	A				B ₂	59 7-112
COMPARE (32)	CR	RR	C						19 7-112
COMPARE (32)	CY	RXY-a	C	LD	A			B ₂	E359 7-112
COMPARE (64)	CG	RXY-a	C	N	A			B ₂	E320 7-112
COMPARE (64)	CGR	RRE	C	N					B920 7-112
COMPARE (64←32)	CGF	RXY-a	C	N	A			B ₂	E330 7-112
COMPARE (64←32)	CGFR	RRE	C	N					B930 7-112
COMPARE (extended BFP)	CXBR	RRE	C		SP	Db	Xi		B349 19-16
COMPARE (extended DFP)	CXTR	RRE	C	TF	SP	Dt	Xi		B3EC 20-21
COMPARE (extended HFP)	CXR	RRE	C		SP	Da			B369 18-10
COMPARE (long BFP)	CDB	RXE	C	A		Db	Xi	B ₂	ED19 19-16
COMPARE (long BFP)	CDBR	RRE	C			Db	Xi		B319 19-16
COMPARE (long DFP)	CDTR	RRE	C	TF		Dt	Xi		B3E4 20-21
COMPARE (long HFP)	CD	RX-a	C	A		Da		B ₂	69 18-10
COMPARE (long HFP)	CDR	RR	C			Da			29 18-10
COMPARE (short BFP)	CEB	RXE	C	A		Db	Xi	B ₂	ED09 19-16
COMPARE (short BFP)	CEBR	RRE	C			Db	Xi		B309 19-16
COMPARE (short HFP)	CE	RX-a	C	A		Da		B ₂	79 18-10
COMPARE (short HFP)	CER	RR	C			Da			39 18-10
COMPARE AND BRANCH (32)	CRB	RRS		GE				B	ECF6 7-113
COMPARE AND BRANCH (64)	CGRB	RRS		GE				B	ECE4 7-113
COMPARE AND BRANCH RELATIVE (32)	CRJ	RIE-b		GE				B	EC76 7-113
COMPARE AND BRANCH RELATIVE (64)	CGRJ	RIE-b		GE				B	EC64 7-113
COMPARE AND FORM CODEWORD	CFC	S	C	A	SP	II	GM I1		B21A 7-115
COMPARE AND SIGNAL (extended BFP)	KXBR	RRE	C		SP	Db	Xi		B348 19-17
COMPARE AND SIGNAL (extended DFP)	KXTR	RRE	C	TF	SP	Dt	Xi		B3E8 20-22
COMPARE AND SIGNAL (long BFP)	KDB	RXE	C	A		Db	Xi	B ₂	ED18 19-17
COMPARE AND SIGNAL (long BFP)	KDBR	RRE	C			Db	Xi		B318 19-17
COMPARE AND SIGNAL (long DFP)	KDTR	RRE	C	TF		Dt	Xi		B3E0 20-22
COMPARE AND SIGNAL (short BFP)	KEB	RXE	C	A		Db	Xi	B ₂	ED08 19-17
COMPARE AND SIGNAL (short BFP)	KEBR	RRE	C			Db	Xi		B308 19-17
COMPARE AND SWAP (32)	CS	RS-a	C	A	SP	\$		ST B ₂	BA 7-122
COMPARE AND SWAP (32)	CSY	RSY-a	C	LD	A	SP	\$	ST B ₂	EB14 7-122
COMPARE AND SWAP (64)	CSG	RSY-a	C	N	A	SP	\$	ST B ₂	EB30 7-122
COMPARE AND SWAP AND PURGE	CSP	RRE	C	P A ¹	SP	\$		ST R ₂	B250 10-17
COMPARE AND SWAP AND PURGE	CSPG	RRE	C	DE P A ¹	SP	\$		ST R ₂	B98A 10-17
COMPARE AND SWAP AND STORE	CSST	SSF	C	CS	A	SP	\$ GM	ST B ₁ B ₂	C82 7-124
COMPARE AND TRAP (32)	CRT	RRF-c		GE		Dc			B972 7-127
COMPARE AND TRAP (64)	CGRT	RRF-c		GE		Dc			B960 7-127
COMPARE BIASED EXPONENT (extended DFP)	CEXT	RRE	C	TF	SP	Dt			B3FC 20-23
COMPARE BIASED EXPONENT (long DFP)	CEDTR	RRE	C	TF		Dt			B3F4 20-23
COMPARE DECIMAL	CP	SS-b	C	A		Dd		B ₁ B ₂	F9 8-6
COMPARE DOUBLE AND SWAP (32)	CDS	RS-a	C	A	SP	\$		ST B ₂	BB 7-122

Figure B-1. Instructions Arranged by Name (Part 3 of 18)

Name	Mne- monic	Characteristics							Op- code	Page
COMPARE DOUBLE AND SWAP (32)	CDSY	RSY-a C LD	A SP	\$	ST	B ₂	EB31	7-122		
COMPARE DOUBLE AND SWAP (64)	CDSG	RSY-a C N	A SP	\$	ST	B ₂	EB3E	7-122		
COMPARE HALFWORD (32←16)	CH	RX-a C	A			B ₂	49	7-128		
COMPARE HALFWORD (32←16)	CHY	RXY-a C LD	A			B ₂	E379	7-128		
COMPARE HALFWORD (64←16)	CGH	RXY-a C GE	A			B ₂	E334	7-128		
COMPARE HALFWORD IMMEDIATE (16)	CHHSI	SIL C GE	A			B ₁	E554	7-128		
COMPARE HALFWORD IMMEDIATE (32←16)	CHI	RI-a C					A7E	7-128		
COMPARE HALFWORD IMMEDIATE (32←16)	CHSI	SIL C GE	A			B ₁	E55C	7-128		
COMPARE HALFWORD IMMEDIATE (64←16)	CGHI	RI-a C N					A7F	7-128		
COMPARE HALFWORD IMMEDIATE (64←16)	CGHSI	SIL C GE	A			B ₁	E558	7-128		
COMPARE HALFWORD RELATIVE LONG (32←16)	CHRL	RIL-b C GE	A				C65	7-128		
COMPARE HALFWORD RELATIVE LONG (64←16)	CGHRL	RIL-b C GE	A				C64	7-128		
COMPARE HIGH (32)	CHF	RXY-a C HW	A			B ₂	E3CD	7-129		
COMPARE HIGH (32)	CHHR	RRE C HW					B9CD	7-129		
COMPARE HIGH (32)	CHLR	RRE C HW					B9DD	7-129		
COMPARE IMMEDIATE (32)	CFI	RIL-a C EI					C2D	7-112		
COMPARE IMMEDIATE (64←32)	CGFI	RIL-a C EI					C2C	7-112		
COMPARE IMMEDIATE AND BRANCH (32←8)	CIB	RIS GE			B		ECFE	7-113		
COMPARE IMMEDIATE AND BRANCH (64←8)	CGIB	RIS GE			B		ECFC	7-113		
COMPARE IMMEDIATE AND BRANCH RELATIVE (32←8)	CIJ	RIE-c GE			B		EC7E	7-114		
COMPARE IMMEDIATE AND BRANCH RELATIVE (64←8)	CGIJ	RIE-c GE			B		EC7C	7-114		
COMPARE IMMEDIATE AND TRAP (32←16)	CIT	RIE-a GE		Dc			EC72	7-127		
COMPARE IMMEDIATE AND TRAP (64←16)	CGIT	RIE-a GE		Dc			EC70	7-127		
COMPARE IMMEDIATE HIGH (32)	CIH	RIL-a C HW					CCD	7-129		
COMPARE LOGICAL (32)	CL	RX-a C	A			B ₂	55	7-130		
COMPARE LOGICAL (32)	CLR	RR C					15	7-130		
COMPARE LOGICAL (32)	CLY	RXY-a C LD	A			B ₂	E355	7-130		
COMPARE LOGICAL (64)	CLG	RXY-a C N	A			B ₂	E321	7-130		
COMPARE LOGICAL (64)	CLGR	RRE C N					B921	7-130		
COMPARE LOGICAL (64←32)	CLGF	RXY-a C N	A			B ₂	E331	7-130		
COMPARE LOGICAL (64←32)	CLGFR	RRE C N					B931	7-130		
COMPARE LOGICAL (character)	CLC	SS-a C	A			B ₁ B ₂	D5	7-130		
COMPARE LOGICAL (immediate)	CLI	SI C	A			B ₁	95	7-130		
COMPARE LOGICAL (immediate)	CLiy	SIY C LD	A			B ₁	EB55	7-130		
COMPARE LOGICAL AND BRANCH (32)	CLRB	RRS GE			B		ECF7	7-132		
COMPARE LOGICAL AND BRANCH (64)	CLGRB	RRS GE			B		ECE5	7-132		
COMPARE LOGICAL AND BRANCH RELATIVE (32)	CLRJ	RIE-b GE			B		EC77	7-132		
COMPARE LOGICAL AND BRANCH RELATIVE (64)	CLGRJ	RIE-b GE			B		EC65	7-132		
COMPARE LOGICAL AND TRAP (32)	CLRT	RRF-c GE		Dc			B973	7-133		
COMPARE LOGICAL AND TRAP (64)	CLGRT	RRF-c GE		Dc			B961	7-133		
COMPARE LOGICAL CHAR. UNDER MASK (high)	CLMH	RSY-b C N	A			B ₂	EB20	7-134		
COMPARE LOGICAL CHAR. UNDER MASK (low)	CLM	RS-b C	A			B ₂	BD	7-134		
COMPARE LOGICAL CHAR. UNDER MASK (low)	CLMY	RSY-b C LD	A			B ₂	EB21	7-134		
COMPARE LOGICAL HIGH (32)	CLHF	RXY-a C HW	A			B ₂	E3CF	7-135		
COMPARE LOGICAL HIGH (32)	CLHHR	RRE C HW					B9CF	7-135		
COMPARE LOGICAL HIGH (32)	CLHLR	RRE C HW					B9DF	7-135		
COMPARE LOGICAL IMMEDIATE (16)	CLHHSI	SIL C GE	A			B ₁	E555	7-130		
COMPARE LOGICAL IMMEDIATE (32)	CLFI	RIL-a C EI					C2F	7-130		
COMPARE LOGICAL IMMEDIATE (32←16)	CLFHSI	SIL C GE	A			B ₁	E55D	7-130		
COMPARE LOGICAL IMMEDIATE (64←16)	CLGHSI	SIL C GE	A			B ₁	E559	7-130		
COMPARE LOGICAL IMMEDIATE (64←32)	CLGFI	RIL-a C EI					C2E	7-130		
COMPARE LOGICAL IMMEDIATE AND BRANCH (32←8)	CLIB	RIS GE			B		ECFF	7-132		

Figure B-1. Instructions Arranged by Name (Part 4 of 18)

Name	Mne- monic	Characteristics						Op- code	Page
COMPARE LOGICAL IMMEDIATE AND BRANCH (64←8)	CLGIB	RIS	GE				B		ECFD 7-132
COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (32←8)	CLIJ	RIE-c	GE				B		EC7F 7-132
COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (64←8)	CLGIJ	RIE-c	GE				B		EC7D 7-132
COMPARE LOGICAL IMMEDIATE AND TRAP (32←16)	CLFIT	RIE-a	GE			Dc			EC73 7-133
COMPARE LOGICAL IMMEDIATE AND TRAP (64←16)	CLGIT	RIE-a	GE			Dc			EC71 7-133
COMPARE LOGICAL IMMEDIATE HIGH (32)	CLIH	RIL-a	C HW						CCF 7-135
COMPARE LOGICAL LONG	CLCL	RR	C	A	SP	II		R ₁ R ₂	0F 7-135
COMPARE LOGICAL LONG EXTENDED	CLCLE	RS-a	C	A	SP	IC		R ₁ R ₃	A9 7-138
COMPARE LOGICAL LONG UNICODE	CLCLU	RSY-a	C E2	A	SP	IC		R ₁ R ₂	EB8F 7-140
COMPARE LOGICAL RELATIVE LONG (32)	CLRL	RIL-b	C GE	A	SP				C6F 7-130
COMPARE LOGICAL RELATIVE LONG (32←16)	CLHRL	RIL-b	C GE	A					C67 7-131
COMPARE LOGICAL RELATIVE LONG (64)	CLGRL	RIL-b	C GE	A	SP				C6A 7-131
COMPARE LOGICAL RELATIVE LONG (64←16)	CLGHRL	RIL-b	C GE	A					C66 7-131
COMPARE LOGICAL RELATIVE LONG (64←32)	CLGFRL	RIL-b	C GE	A	SP			R ₁ R ₂	C6E 7-131
COMPARE LOGICAL STRING	CLST	RRE	C	A	SP	IC	G0		B25D 7-143
COMPARE RELATIVE LONG (32)	CRL	RIL-b	C GE	A	SP				C6D 7-112
COMPARE RELATIVE LONG (64)	CGRL	RIL-b	C GE	A	SP				C68 7-112
COMPARE RELATIVE LONG (64←32)	CGFRL	RIL-b	C GE	A	SP				C6C 7-112
COMPARE UNTIL SUBSTRING EQUAL	CUSE	RRE	C	A	SP	II	GM	R ₁ R ₂	B257 7-144
COMPRESSION CALL	CMPSC	RRE	C	A	SP	II	Dd GM	ST R ₁ R ₂	B263 7-147
COMPUTE INTERMEDIATE MESSAGE DIGEST	KIMD	RRE	C MS	A	SP	IC	GM I1	ST R ₂	B93E 7-159
COMPUTE LAST MESSAGE DIGEST	KLMD	RRE	C MS	A	SP	IC	GM I1	ST R ₂	B93F 7-159
COMPUTE MESSAGE AUTHENTICATION CODE	KMAC	RRE	C MS	A	SP	IC	GM I1	ST R ₂	B91E 7-171
CONVERT BFP TO HFP (long)	THDR	RRE	C			Da			B359 9-27
CONVERT BFP TO HFP (short to long)	THDER	RRE	C			Da			B358 9-27
CONVERT FROM FIXED (32 to extended BFP)	CXFBR	RRE			SP	Db			B396 19-18
CONVERT FROM FIXED (32 to extended BFP)	CXFBRA	RRF-e	F		SP	Db			B396 19-18
CONVERT FROM FIXED (32 to extended DFP)	CXFTR	RRE	F		SP	Dt			B959 20-24
CONVERT FROM FIXED (32 to extended HFP)	CXFR	RRE			SP	Da			B3B6 18-10
CONVERT FROM FIXED (32 to long BFP)	CDFBR	RRE				Db			B395 19-18
CONVERT FROM FIXED (32 to long BFP)	CDFBRA	RRF-e	F		SP	Db			B395 19-18
CONVERT FROM FIXED (32 to long DFP)	CDFTR	RRE	F			Dt			B951 20-24
CONVERT FROM FIXED (32 to long HFP)	CDFR	RRE				Da			B3B5 18-10
CONVERT FROM FIXED (32 to short BFP)	CEFBR	RRE				Db	Xx		B394 19-18
CONVERT FROM FIXED (32 to short BFP)	CEFBRA	RRF-e	F		SP	Db	Xx		B394 19-18
CONVERT FROM FIXED (32 to short HFP)	CEFR	RRE				Da			B3B4 18-10
CONVERT FROM FIXED (64 to extended BFP)	CXGBR	RRE	N		SP	Db			B3A6 19-18
CONVERT FROM FIXED (64 to extended BFP)	CXGBRA	RRF-e	F		SP	Db			B3A6 19-18
CONVERT FROM FIXED (64 to extended DFP)	CXGTR	RRE	TF		SP	Dt			B3F9 20-24
CONVERT FROM FIXED (64 to extended DFP)	CXGTRA	RRF-e	F		SP	Dt			B3F9 20-24
CONVERT FROM FIXED (64 to extended HFP)	CXGR	RRE	N		SP	Da			B3C6 18-10
CONVERT FROM FIXED (64 to long BFP)	CDGBR	RRE	N			Db	Xx		B3A5 19-18
CONVERT FROM FIXED (64 to long BFP)	CDGBRA	RRF-e	F		SP	Db	Xx		B3A5 19-18
CONVERT FROM FIXED (64 to long DFP)	CDGTR	RRE	TF			Dt	Xx		B3F1 20-24
CONVERT FROM FIXED (64 to long DFP)	CDGTRA	RRF-e	F			Dt	Xx Xq		B3F1 20-24
CONVERT FROM FIXED (64 to long HFP)	CDGR	RRE	N			Da			B3C5 18-10
CONVERT FROM FIXED (64 to short BFP)	CEGBR	RRE	N			Db	Xx		B3A4 19-18
CONVERT FROM FIXED (64 to short BFP)	CEGBRA	RRF-e	F		SP	Db	Xx		B3A4 19-18
CONVERT FROM FIXED (64 to short HFP)	CEGR	RRE	N			Da			B3C4 18-10
CONVERT FROM LOGICAL (32 to extended BFP)	CXLFBR	RRF-e	F		SP	Db			B392 19-20
CONVERT FROM LOGICAL (32 to extended DFP)	CXLFTTR	RRF-e	F		SP	Dt			B95B 20-25

Figure B-1. Instructions Arranged by Name (Part 5 of 18)

Name	Mnemonic	Characteristics						Op-code	Page
CONVERT FROM LOGICAL (32 to long BFP)	CDLFBR	RRF-e	F	SP	Db			B391	19-20
CONVERT FROM LOGICAL (32 to long DFP)	CDLFTR	RRF-e	F		Dt			B953	20-25
CONVERT FROM LOGICAL (32 to short BFP)	CELFBR	RRF-e	F	SP	Db	Xx		B390	19-20
CONVERT FROM LOGICAL (64 to extended BFP)	CXLGBR	RRF-e	F	SP	Db			B3A2	19-20
CONVERT FROM LOGICAL (64 to extended DFP)	CXLGTR	RRF-e	F	SP	Dt			B95A	20-25
CONVERT FROM LOGICAL (64 to long BFP)	CDLGBR	RRF-e	F	SP	Db	Xx		B3A1	19-20
CONVERT FROM LOGICAL (64 to long DFP)	CDLGTR	RRF-e	F		Dt	Xx Xq		B952	20-25
CONVERT FROM LOGICAL (64 to short BFP)	CELGBR	RRF-e	F	SP	Db	Xx		B3A0	19-20
CONVERT FROM SIGNED PACKED (128 to extended DFP)	CXSTR	RRE	TF	SP	Dt Dd			B3FB	20-26
CONVERT FROM SIGNED PACKED (64 to long DFP)	CDSTR	RRE	TF		Dt Dd			B3F3	20-26
CONVERT FROM UNSIGNED PACKED (128 to ext. DFP)	CXUTR	RRE	TF	SP	Dt Dd			B3FA	20-27
CONVERT FROM UNSIGNED PACKED (64 to long DFP)	CDUTR	RRE	TF		Dt Dd			B3F2	20-27
CONVERT HFP TO BFP (long to short)	TBEDR	RRF-e	C	SP	Da			B350	9-28
CONVERT HFP TO BFP (long)	TBDR	RRF-e	C	SP	Da			B351	9-28
CONVERT TO BINARY (32)	CVB	RX-a		A	Dd IK		B ₂	4F	7-181
CONVERT TO BINARY (32)	CVBY	RXY-a	LD	A	Dd IK		B ₂	E306	7-181
CONVERT TO BINARY (64)	CVBG	RXY-a	N	A	Dd IK		B ₂	E30E	7-181
CONVERT TO DECIMAL (32)	CVD	RX-a		A			ST B ₂	4E	7-182
CONVERT TO DECIMAL (32)	CVDY	RXY-a	LD	A			ST B ₂	E326	7-182
CONVERT TO DECIMAL (64)	CVDG	RXY-a	N	A			ST B ₂	E32E	7-182
CONVERT TO FIXED (extended BFP to 32)	CFXBR	RRF-e	C	SP	Db Xi	Xx		B39A	19-21
CONVERT TO FIXED (extended BFP to 32)	CFXBRA	RRF-e	C F	SP	Db Xi	Xx		B39A	19-21
CONVERT TO FIXED (extended BFP to 64)	CGXBR	RRF-e	C N	SP	Db Xi	Xx		B3AA	19-21
CONVERT TO FIXED (extended BFP to 64)	CGXBRA	RRF-e	C F	SP	Db Xi	Xx		B3AA	19-21
CONVERT TO FIXED (extended DFP to 32)	CFXTR	RRF-e	C F	SP	Dt Xi	Xx		B949	20-27
CONVERT TO FIXED (extended DFP to 64)	CGXTR	RRF-e	C TF	SP	Dt Xi	Xx		B3E9	20-27
CONVERT TO FIXED (extended DFP to 64)	CGXTRA	RRF-e	C F	SP	Dt Xi	Xx		B3E9	20-27
CONVERT TO FIXED (extended HFP to 32)	CFXR	RRF-e	C	SP	Da			B3BA	18-11
CONVERT TO FIXED (extended HFP to 64)	CGXR	RRF-e	C N	SP	Da			B3CA	18-11
CONVERT TO FIXED (long BFP to 32)	CFDBR	RRF-e	C	SP	Db Xi	Xx		B399	19-21
CONVERT TO FIXED (long BFP to 32)	CFDBRA	RRF-e	C F	SP	Db Xi	Xx		B399	19-21
CONVERT TO FIXED (long BFP to 64)	CGDBR	RRF-e	C N	SP	Db Xi	Xx		B3A9	19-21
CONVERT TO FIXED (long BFP to 64)	CGDBRA	RRF-e	C F	SP	Db Xi	Xx		B3A9	19-21
CONVERT TO FIXED (long DFP to 32)	CFDTR	RRF-e	C F		Dt Xi	Xx		B941	20-27
CONVERT TO FIXED (long DFP to 64)	CGDTR	RRF-e	C TF		Dt Xi	Xx		B3E1	20-27
CONVERT TO FIXED (long DFP to 64)	CGDTRA	RRF-e	C F		Dt Xi	Xx		B3E1	20-27
CONVERT TO FIXED (long HFP to 32)	CFDR	RRF-e	C	SP	Da			B3B9	18-11
CONVERT TO FIXED (long HFP to 64)	CGDR	RRF-e	C N	SP	Da			B3C9	18-11
CONVERT TO FIXED (short BFP to 32)	CFEBR	RRF-e	C	SP	Db Xi	Xx		B398	19-21
CONVERT TO FIXED (short BFP to 32)	CFEBRA	RRF-e	C F	SP	Db Xi	Xx		B398	19-21
CONVERT TO FIXED (short BFP to 64)	CGEBR	RRF-e	C N	SP	Db Xi	Xx		B3A8	19-21
CONVERT TO FIXED (short BFP to 64)	CGEBRA	RRF-e	C F	SP	Db Xi	Xx		B3A8	19-21
CONVERT TO FIXED (short HFP to 32)	CFER	RRF-e	C	SP	Da			B3B8	18-11
CONVERT TO FIXED (short HFP to 64)	CGER	RRF-e	C N	SP	Da			B3C8	18-11
CONVERT TO LOGICAL (extended BFP to 32)	CLFXBR	RRF-e	C F	SP	Db Xi	Xx		B39E	19-23
CONVERT TO LOGICAL (extended BFP to 64)	CLGXBR	RRF-e	C F	SP	Db Xi	Xx		B3AE	19-23
CONVERT TO LOGICAL (extended DFP to 32)	CLFXTR	RRF-e	C F	SP	Dt Xi	Xx		B94B	20-29
CONVERT TO LOGICAL (extended DFP to 64)	CLGXTR	RRF-e	C F	SP	Dt Xi	Xx		B94A	20-29
CONVERT TO LOGICAL (long BFP to 32)	CLFDBR	RRF-e	C F	SP	Db Xi	Xx		B39D	19-23
CONVERT TO LOGICAL (long BFP to 64)	CLGDBR	RRF-e	C F	SP	Db Xi	Xx		B3AD	19-23
CONVERT TO LOGICAL (long DFP to 32)	CLFDTR	RRF-e	C F		Dt Xi	Xx		B943	20-29
CONVERT TO LOGICAL (long DFP to 64)	CLGDTR	RRF-e	C F		Dt Xi	Xx		B942	20-29

Figure B-1. Instructions Arranged by Name (Part 6 of 18)

Name	Mne- monic	Characteristics						Op- code	Page
CONVERT TO LOGICAL (short BFP to 32)	CLFEBR	RRF-e C F	SP	Db Xi	Xx			B39C	19-23
CONVERT TO LOGICAL (short BFP to 64)	CLGEBR	RRF-e C F	SP	Db Xi	Xx			B3AC	19-23
CONVERT TO SIGNED PACKED (extended DFP to 128)	CSXTR	RRF-d TF	SP	Dt				B3EB	20-31
CONVERT TO SIGNED PACKED (long DFP to 64)	CSDTR	RRF-d TF		Dt				B3E3	20-31
CONVERT TO UNSIGNED PACKED (extended DFP to 128)	CUXTR	RRE TF	SP	Dt				B3EA	20-32
CONVERT TO UNSIGNED PACKED (long DFP to 64)	CUDTR	RRE TF		Dt				B3E2	20-32
CONVERT UNICODE TO UTF-8	CUUTF	RRF-c C	A SP	IC		ST	R ₁ R ₂	B2A6	7-186
CONVERT UTF-16 TO UTF-32	CU24	RRF-c C E3	A SP	IC		ST	R ₁ R ₂	B9B1	7-183
CONVERT UTF-16 TO UTF-8	CU21	RRF-c C	A SP	IC		ST	R ₁ R ₂	B2A6	7-186
CONVERT UTF-32 TO UTF-16	CU42	RRE C E3	A SP	IC		ST	R ₁ R ₂	B9B3	7-189
CONVERT UTF-32 TO UTF-8	CU41	RRE C E3	A SP	IC		ST	R ₁ R ₂	B9B2	7-192
CONVERT UTF-8 TO UNICODE	CUTFU	RRF-c C	A SP	IC		ST	R ₁ R ₂	B2A7	7-195
CONVERT UTF-8 TO UTF-16	CU12	RRF-c C	A SP	IC		ST	R ₁ R ₂	B2A7	7-195
CONVERT UTF-8 TO UTF-32	CU14	RRF-c C E3	A SP	IC		ST	R ₁ R ₂	B9B0	7-199
COPY ACCESS	CPYA	RRE					U ₁ U ₂	B24D	7-203
COPY SIGN (long)	CPSDR	RRF-b FS		Da				B372	9-29
DIAGNOSE		DM	P DM				MD	83	10-19
DIVIDE (32←64)	D	RX-a	A SP	IK			B ₂	5D	7-203
DIVIDE (32←64)	DR	RR	SP	IK				1D	7-203
DIVIDE (extended BFP)	DXBR	RRE	SP	Db Xi Xz Xo Xu Xx				B34D	19-26
DIVIDE (extended DFP)	DXTR	RRF-a TF	SP	Dt Xi Xz Xo Xu Xx				B3D9	20-32
DIVIDE (extended DFP)	DXTRA	RRF-a F	SP	Dt Xi Xz Xo Xu Xx Xq				B3D9	20-32
DIVIDE (extended HFP)	DXR	RRE	SP	Da EU EO FK				B2D2	18-12
DIVIDE (long BFP)	DDB	RXE	A	Db Xi Xz Xo Xu Xx			B ₂	ED1D	19-26
DIVIDE (long BFP)	DDBR	RRE		Db Xi Xz Xo Xu Xx				B31D	19-26
DIVIDE (long DFP)	DDTR	RRF-a TF		Dt Xi Xz Xo Xu Xx				B3D1	20-32
DIVIDE (long DFP)	DDTRA	RRF-a F		Dt Xi Xz Xo Xu Xx Xq				B3D1	20-32
DIVIDE (long HFP)	DD	RX-a	A	Da EU EO FK			B ₂	6D	18-12
DIVIDE (long HFP)	DDR	RR		Da EU EO FK				2D	18-11
DIVIDE (short BFP)	DEB	RXE	A	Db Xi Xz Xo Xu Xx			B ₂	ED0D	19-26
DIVIDE (short BFP)	DEBR	RRE		Db Xi Xz Xo Xu Xx				B30D	19-26
DIVIDE (short HFP)	DE	RX-a	A	Da EU EO FK			B ₂	7D	18-12
DIVIDE (short HFP)	DER	RR		Da EU EO FK				3D	18-11
DIVIDE DECIMAL	DP	SS-b	A SP	Dd DK		ST	B ₁ B ₂	FD	8-7
DIVIDE LOGICAL (32←64)	DL	RXY-a N3	A SP	IK			B ₂	E397	7-204
DIVIDE LOGICAL (32←64)	DLR	RRE N3	SP	IK				B997	7-204
DIVIDE LOGICAL (64←128)	DLG	RXY-a N	A SP	IK			B ₂	E387	7-204
DIVIDE LOGICAL (64←128)	DLGR	RRE N	SP	IK				B987	7-204
DIVIDE SINGLE (64)	DSG	RXY-a N	A SP	IK			B ₂	E30D	7-205
DIVIDE SINGLE (64)	DSGR	RRE N	SP	IK				B90D	7-204
DIVIDE SINGLE (64←32)	DSGF	RXY-a N	A SP	IK			B ₂	E31D	7-205
DIVIDE SINGLE (64←32)	DSGFR	RRE N	SP	IK				B91D	7-204
DIVIDE TO INTEGER (long BFP)	DIDBR	RRF-b C	SP	Db Xi Xu Xx				B35B	19-27
DIVIDE TO INTEGER (short BFP)	DIEBR	RRF-b C	SP	Db Xi Xu Xx				B353	19-27
EDIT	ED	SS-a C	A	Dd		ST	B ₁ B ₂	DE	8-7
EDIT AND MARK	EDMK	SS-a C	A	Dd	G1	ST	B ₁ B ₂	DF	8-10
EXCLUSIVE OR (32)	X	RX-a C	A				B ₂	57	7-205
EXCLUSIVE OR (32)	XR	RR C						17	7-205
EXCLUSIVE OR (32)	XRK	RRF-a C DO						B9F7	7-205
EXCLUSIVE OR (32)	XY	RXY-a C LD	A				B ₂	E357	7-205
EXCLUSIVE OR (64)	XG	RXY-a C N	A				B ₂	E382	7-205
EXCLUSIVE OR (64)	XGR	RRE C N						B982	7-205
EXCLUSIVE OR (64)	XGRK	RRF-a C DO						B9E7	7-205

Figure B-1. Instructions Arranged by Name (Part 7 of 18)

Name	Mne- monic	Characteristics						Op- code	Page
EXCLUSIVE OR (character)	XC	SS-a	C	A		ST	B ₁ B ₂	D7	7-206
EXCLUSIVE OR (immediate)	XI	SI	C	A		ST	B ₁	97	7-205
EXCLUSIVE OR (immediate)	XIY	SIY	C LD	A		ST	B ₁	EB57	7-205
EXCLUSIVE OR IMMEDIATE (high)	XIHF	RIL-a	C EI					C06	7-206
EXCLUSIVE OR IMMEDIATE (low)	XILF	RIL-a	C EI					C07	7-206
EXECUTE	EX	RX-a		AI SP	EX			44	7-207
EXECUTE RELATIVE LONG	EXRL	RIL-b	XX	AI	EX			C60	7-207
EXTRACT ACCESS	EAR	RRE					U ₂	B24F	7-208
EXTRACT AND SET EXTENDED AUTHORITY	ESEA	RRE	N	P				B99D	10-20
EXTRACT BIASED EXPONENT (extended DFP to 64)	EEXTR	RRE	TF		SP Dt			B3ED	20-34
EXTRACT BIASED EXPONENT (long DFP to 64)	EEDTR	RRE	TF		Dt			B3E5	20-34
EXTRACT CACHE ATTRIBUTE	ECAG	RSY-a	GE					EB4C	7-208
EXTRACT CPU TIME	ECTG	SSF	ET	A	GM	R ₃	B ₁ B ₂	C81	7-210
EXTRACT FPC	EFPC	RRE			Db			B38C	9-30
EXTRACT PRIMARY ASN	EPAR	RRE		Q	SO			B226	10-20
EXTRACT PRIMARY ASN AND INSTANCE	EPAIR	RRE	RA	Q	SO			B99A	10-20
EXTRACT PSW	EPSW	RRE	N3					B98D	7-210
EXTRACT SECONDARY ASN	ESAR	RRE		Q	SO			B227	10-21
EXTRACT SECONDARY ASN AND INSTANCE	ESAIR	RRE	RA	Q	SO			B99B	10-21
EXTRACT SIGNIFICANCE (extended DFP)	ESXTR	RRE	TF		SP Dt			B3EF	20-35
EXTRACT SIGNIFICANCE (long DFP)	ESDTR	RRE	TF		Dt			B3E7	20-35
EXTRACT STACKED REGISTERS (32)	EREG	RRE		A ¹	SE		U ₁ U ₂	B249	10-21
EXTRACT STACKED REGISTERS (64)	EREGG	RRE	N	A ¹	SE		U ₁ U ₂	B90E	10-21
EXTRACT STACKED STATE	ESTA	RRE	C	A ¹ SP	SE			B24A	10-23
FIND LEFTMOST ONE	FLOGR	RRE	C EI		SP			B983	7-211
HALT SUBCHANNEL	HSCH	S	C	P	OP ¢ GS			B231	14-6
HALVE (long HFP)	HDR	RR			Da EU			24	18-13
HALVE (short HFP)	HER	RR			Da EU			34	18-13
INSERT ADDRESS SPACE CONTROL	IAC	RRE	C	Q	SO			B224	10-25
INSERT BIASED EXPONENT (64 to extended DFP)	IEXTR	RRF-b	TF		SP Dt			B3FE	20-35
INSERT BIASED EXPONENT (64 to long DFP)	IEDTR	RRF-b	TF		Dt			B3F6	20-35
INSERT CHARACTER	IC	RX-a		A			B ₂	43	7-211
INSERT CHARACTER	ICY	RXY-a	LD	A			B ₂	E373	7-211
INSERT CHARACTERS UNDER MASK (high)	ICMH	RSY-b	C N	A			B ₂	EB80	7-212
INSERT CHARACTERS UNDER MASK (low)	ICM	RS-b	C	A			B ₂	BF	7-212
INSERT CHARACTERS UNDER MASK (low)	ICMY	RSY-b	C LD	A			B ₂	EB81	7-212
INSERT IMMEDIATE (high high)	IIHH	RI-a	N					A50	7-212
INSERT IMMEDIATE (high low)	IIHL	RI-a	N					A51	7-213
INSERT IMMEDIATE (high)	IIHF	RIL-a	EI					C08	7-212
INSERT IMMEDIATE (low high)	IILH	RI-a	N					A52	7-213
INSERT IMMEDIATE (low low)	IILL	RI-a	N					A53	7-213
INSERT IMMEDIATE (low)	IILF	RIL-a	EI					C09	7-213
INSERT PROGRAM MASK	IPM	RRE						B222	7-213
INSERT PSW KEY	IPK	S		Q	G2			B20B	10-26
INSERT STORAGE KEY EXTENDED	ISKE	RRE		P A ¹				B229	10-26
INSERT VIRTUAL STORAGE KEY	IVSK	RRE		Q A ¹	SO		R ₂	B223	10-27
INVALIDATE DAT TABLE ENTRY	IDTE	RRF-b	DE	P A ¹ SP	\$			B98E	10-28
INVALIDATE PAGE TABLE ENTRY	IPTE	RRF-a		P A ¹ SP	\$			B221	10-32
LOAD (32)	L	RX-a		A			B ₂	58	7-213
LOAD (32)	LR	RR						18	7-213
LOAD (32)	LY	RXY-a	LD	A			B ₂	E358	7-213
LOAD (64)	LG	RXY-a	N	A			B ₂	E304	7-213
LOAD (64)	LGR	RRE	N					B904	7-213
LOAD (64←32)	LGF	RXY-a	N	A			B ₂	E314	7-213
LOAD (64←32)	LGFR	RRE	N					B914	7-213

Figure B-1. Instructions Arranged by Name (Part 8 of 18)

Name	Mne- monic	Characteristics							Op- code	Page	
LOAD (extended)	LXR	RRE		SP	Da				B365	9-30	
LOAD (long)	LD	RX-a		A	Da			B ₂	68	9-30	
LOAD (long)	LDR	RR			Da				28	9-30	
LOAD (long)	LDY	RXY-a	LD	A	Da			B ₂	ED65	9-30	
LOAD (short)	LE	RX-a		A	Da			B ₂	78	9-30	
LOAD (short)	LER	RR			Da				38	9-30	
LOAD (short)	LEY	RXY-a	LD	A	Da			B ₂	ED64	9-30	
LOAD ACCESS MULTIPLE	LAM	RS-a		A	SP			UB	9A	7-214	
LOAD ACCESS MULTIPLE	LAMY	RSY-a	LD	A	SP			UB	EB9A	7-214	
LOAD ADDRESS	LA	RX-a							41	7-215	
LOAD ADDRESS	LAY	RXY-a	LD						E371	7-215	
LOAD ADDRESS EXTENDED	LAE	RX-a						U ₁	BP	51	7-215
LOAD ADDRESS EXTENDED	LAEY	RXY-a	GE					U ₁	BP	E375	7-215
LOAD ADDRESS RELATIVE LONG	LARL	RIL-b	N3						C00	7-216	
LOAD ADDRESS SPACE PARAMETERS	LASP	SSE	C	P	A ¹	SP	SO	B ₁	E500	10-34	
LOAD AND ADD (32)	LAA	RSY-a	C	IA	A	SP	IF	ST	B ₂	EBF8	7-217
LOAD AND ADD (64)	LAAG	RSY-a	C	IA	A	SP	IF	ST	B ₂	EBE8	7-217
LOAD AND ADD LOGICAL (32)	LAAL	RSY-a	C	IA	A	SP		ST	B ₂	EBFA	7-217
LOAD AND ADD LOGICAL (64)	LAALG	RSY-a	C	IA	A	SP		ST	B ₂	EBEA	7-217
LOAD AND AND (32)	LAN	RSY-a	C	IA	A	SP		ST	B ₂	EBF4	7-218
LOAD AND AND (64)	LANG	RSY-a	C	IA	A	SP		ST	B ₂	EBE4	7-218
LOAD AND EXCLUSIVE OR (32)	LAX	RSY-a	C	IA	A	SP		ST	B ₂	EBF7	7-218
LOAD AND EXCLUSIVE OR (64)	LAXG	RSY-a	C	IA	A	SP		ST	B ₂	EBE7	7-218
LOAD AND OR (32)	LAO	RSY-a	C	IA	A	SP		ST	B ₂	EBF6	7-219
LOAD AND OR (64)	LAOG	RSY-a	C	IA	A	SP		ST	B ₂	EBE6	7-219
LOAD AND TEST (32)	LT	RXY-a	C	EI	A				B ₂	E312	7-220
LOAD AND TEST (32)	LTR	RR	C							12	7-219
LOAD AND TEST (64)	LTG	RXY-a	C	EI	A				B ₂	E302	7-220
LOAD AND TEST (64)	LTGR	RRE	C	N						B902	7-219
LOAD AND TEST (64←32)	LTGF	RXY-a	C	GE	A				B ₂	E332	7-220
LOAD AND TEST (64←32)	LTGFR	RRE	C	N						B912	7-220
LOAD AND TEST (extended BFP)	LTXBR	RRE	C		SP	Db	Xi			B342	19-30
LOAD AND TEST (extended DFP)	LTXTR	RRE	C	TF	SP	Dt	Xi			B3DE	20-36
LOAD AND TEST (extended HFP)	LTXR	RRE	C		SP	Da				B362	18-13
LOAD AND TEST (long BFP)	LTDBR	RRE	C			Db	Xi			B312	19-30
LOAD AND TEST (long DFP)	LTDT	RRE	C	TF		Dt	Xi			B3D6	20-36
LOAD AND TEST (long HFP)	LTDR	RR	C			Da				22	18-13
LOAD AND TEST (short BFP)	LTEBR	RRE	C			Db	Xi			B302	19-30
LOAD AND TEST (short HFP)	LTER	RR	C			Da				32	18-13
LOAD BYTE (32)	LB	RXY-a	LD	A						E376	7-220
LOAD BYTE (32)	LBR	RRE	EI							B926	7-220
LOAD BYTE (64)	LGB	RXY-a	LD	A						E377	7-220
LOAD BYTE (64)	LGBR	RRE	EI							B906	7-220
LOAD BYTE HIGH (32←8)	LBH	RXY-a	HW	A				B ₂	E3C0	7-221	
LOAD COMPLEMENT (32)	LCR	RR	C			IF			13	7-221	
LOAD COMPLEMENT (64)	LCGR	RRE	C	N		IF				B903	7-221
LOAD COMPLEMENT (64←32)	LCGFR	RRE	C	N						B913	7-221
LOAD COMPLEMENT (extended BFP)	LCXBR	RRE	C		SP	Db				B343	19-30
LOAD COMPLEMENT (extended HFP)	LCXR	RRE	C		SP	Da				B363	18-14
LOAD COMPLEMENT (long BFP)	LCDBR	RRE	C			Db				B313	19-30
LOAD COMPLEMENT (long HFP)	LCDR	RR	C			Da				23	18-14
LOAD COMPLEMENT (long)	LCDFR	RRE	FS			Da				B373	9-30
LOAD COMPLEMENT (short BFP)	LCEBR	RRE	C			Db				B303	19-30
LOAD COMPLEMENT (short HFP)	LCER	RR	C			Da				33	18-14
LOAD CONTROL (32)	LCTL	RS-a		P	A	SP			B ₂	B7	10-44

Figure B-1. Instructions Arranged by Name (Part 9 of 18)

Name	Mnemonic	Characteristics						Op-code	Page
LOAD CONTROL (64)	LCTLG	RSY-a	N	P	A	SP		B ₂	EB2F 10-44
LOAD FP INTEGER (extended BFP)	FIXBR	RRF-e				SP	Db Xi Xx		B347 19-31
LOAD FP INTEGER (extended BFP)	FIXBRA	RRF-e	F			SP	Db Xi Xx		B347 19-31
LOAD FP INTEGER (extended DFP)	FIXTR	RRF-e	TF			SP	Dt Xi Xx Xq		B3DF 20-37
LOAD FP INTEGER (extended HFP)	FIXR	RRE				SP	Da		B367 18-14
LOAD FP INTEGER (long BFP)	FIDBR	RRF-e				SP	Db Xi Xx		B35F 19-31
LOAD FP INTEGER (long BFP)	FIDBRA	RRF-e	F			SP	Db Xi Xx		B35F 19-31
LOAD FP INTEGER (long DFP)	FIDTR	RRF-e	TF				Dt Xi Xx Xq		B3D7 20-37
LOAD FP INTEGER (long HFP)	FIDR	RRE					Da		B37F 18-14
LOAD FP INTEGER (short BFP)	FIEBR	RRF-e				SP	Db Xi Xx		B357 19-31
LOAD FP INTEGER (short BFP)	FIEBRA	RRF-e	F			SP	Db Xi Xx		B357 19-31
LOAD FP INTEGER (short HFP)	FIER	RRE					Da		B377 18-14
LOAD FPC	LFPC	S		A		SP	Db	B ₂	B29D 9-31
LOAD FPC AND SIGNAL	LFAS	S	XF	A		SP	Dt Xg	B ₂	B2BD 9-31
LOAD FPR FROM GR (64 to long)	LDGR	RRE	FG				Da		B3C1 9-33
LOAD GR FROM FPR (long to 64)	LGDR	RRE	FG				Da		B3CD 9-33
LOAD HALFWORD (32)	LH	RX-a		A				B ₂	48 7-221
LOAD HALFWORD (32)	LHR	RRE	EI						B927 7-221
LOAD HALFWORD (32)	LHY	RXY-a	LD	A				B ₂	E378 7-221
LOAD HALFWORD (64)	LGH	RXY-a	N	A				B ₂	E315 7-222
LOAD HALFWORD (64)	LGHR	RRE	EI						B907 7-221
LOAD HALFWORD HIGH (32←16)	LHH	RXY-a	HW	A				B ₂	E3C4 7-222
LOAD HALFWORD IMMEDIATE (32)	LHI	RI-a							A78 7-222
LOAD HALFWORD IMMEDIATE (64)	LGHI	RI-a	N						A79 7-222
LOAD HALFWORD RELATIVE LONG (32←16)	LHRL	RIL-b	GE	A					C45 7-222
LOAD HALFWORD RELATIVE LONG (64←16)	LGHRL	RIL-b	GE	A					C44 7-222
LOAD HIGH (32)	LFH	RXY-a	HW	A				B ₂	E3CA 7-223
LOAD IMMEDIATE (64←32)	LGFI	RIL-a	EI						C01 7-214
LOAD LENGTHENED (long to extended BFP)	LXDB	RXE		A		SP	Db Xi	B ₂	ED05 19-33
LOAD LENGTHENED (long to extended BFP)	LXDDBR	RRE				SP	Db Xi		B305 19-32
LOAD LENGTHENED (long to extended DFP)	LXDTR	RRF-d	TF			SP	Dt Xi		B3DC 20-40
LOAD LENGTHENED (long to extended HFP)	LXD	RXE		A		SP	Da	B ₂	ED25 18-15
LOAD LENGTHENED (long to extended HFP)	LXDR	RRE				SP	Da		B325 18-15
LOAD LENGTHENED (short to extended BFP)	LXEB	RXE		A		SP	Db Xi	B ₂	ED06 19-33
LOAD LENGTHENED (short to extended BFP)	LXEBr	RRE				SP	Db Xi		B306 19-32
LOAD LENGTHENED (short to extended HFP)	LXE	RXE		A		SP	Da	B ₂	ED26 18-15
LOAD LENGTHENED (short to extended HFP)	LXER	RRE				SP	Da		B326 18-15
LOAD LENGTHENED (short to long BFP)	LDEB	RXE		A			Db Xi	B ₂	ED04 19-33
LOAD LENGTHENED (short to long BFP)	LDEBR	RRE					Db Xi		B304 19-32
LOAD LENGTHENED (short to long DFP)	LDETR	RRF-d	TF				Dt Xi		B3D4 20-40
LOAD LENGTHENED (short to long HFP)	LDE	RXE		A			Da	B ₂	ED24 18-15
LOAD LENGTHENED (short to long HFP)	LDER	RRE					Da		B324 18-15
LOAD LOGICAL (64←32)	LLGF	RXY-a	N	A				B ₂	E316 7-223
LOAD LOGICAL (64←32)	LLGFR	RRE	N						B916 7-223
LOAD LOGICAL CHARACTER (32)	LLC	RXY-a	EI	A				B ₂	E394 7-224
LOAD LOGICAL CHARACTER (32)	LLCR	RRE	EI						B994 7-224
LOAD LOGICAL CHARACTER (64)	LLGC	RXY-a	N	A				B ₂	E390 7-224
LOAD LOGICAL CHARACTER (64)	LLGCR	RRE	EI						B984 7-223
LOAD LOGICAL CHARACTER HIGH (32←8)	LLCH	RXY-a	HW	A				B ₂	E3C2 7-224
LOAD LOGICAL HALFWORD (32)	LLH	RXY-a	EI	A				B ₂	E395 7-224
LOAD LOGICAL HALFWORD (32)	LLHR	RRE	EI						B995 7-224
LOAD LOGICAL HALFWORD (64)	LLGH	RXY-a	N	A				B ₂	E391 7-224
LOAD LOGICAL HALFWORD (64)	LLGHR	RRE	EI						B985 7-224
LOAD LOGICAL HALFWORD HIGH (32←16)	LLHH	RXY-a	HW	A				B ₂	E3C6 7-225

Figure B-1. Instructions Arranged by Name (Part 10 of 18)

Name	Mne- monic	Characteristics						Op- code	Page
LOAD LOGICAL HALFWORD RELATIVE LONG (32←16)	LLHRL	RIL-b	GE	A				C42	7-224
LOAD LOGICAL HALFWORD RELATIVE LONG (64←16)	LLGHRL	RIL-b	GE	A				C46	7-224
LOAD LOGICAL IMMEDIATE (high high)	LLIHH	RI-a	N					A5C	7-225
LOAD LOGICAL IMMEDIATE (high low)	LLIHL	RI-a	N					A5D	7-225
LOAD LOGICAL IMMEDIATE (high)	LLIHF	RIL-a	EI					C0E	7-225
LOAD LOGICAL IMMEDIATE (low high)	LLILH	RI-a	N					A5E	7-225
LOAD LOGICAL IMMEDIATE (low low)	LLILL	RI-a	N					A5F	7-225
LOAD LOGICAL IMMEDIATE (low)	LLILF	RIL-a	EI					C0F	7-225
LOAD LOGICAL RELATIVE LONG (64←32)	LLGFRL	RIL-b	GE	A	SP			C4E	7-223
LOAD LOGICAL THIRTY ONE BITS	LLGT	RXY-a	N	A			B ₂	E317	7-226
LOAD LOGICAL THIRTY ONE BITS	LLGTR	RRE	N					B917	7-226
LOAD MULTIPLE (32)	LM	RS-a		A			B ₂	98	7-226
LOAD MULTIPLE (32)	LMY	RSY-a	LD	A			B ₂	EB98	7-226
LOAD MULTIPLE (64)	LMG	RSY-a	N	A			B ₂	EB04	7-226
LOAD MULTIPLE DISJOINT	LMD	SS-e	N	A			B ₂ B ₄	EF	7-227
LOAD MULTIPLE HIGH	LMH	RSY-a	N	A			B ₂	EB96	7-227
LOAD NEGATIVE (32)	LNR	RR	C					11	7-227
LOAD NEGATIVE (64)	LNGR	RRE	C	N				B901	7-227
LOAD NEGATIVE (64←32)	LNGFR	RRE	C	N				B911	7-227
LOAD NEGATIVE (extended BFP)	LNxBR	RRE	C		SP	Db		B341	19-33
LOAD NEGATIVE (extended HFP)	LNxR	RRE	C		SP	Da		B361	18-15
LOAD NEGATIVE (long BFP)	LNDBR	RRE	C			Db		B311	19-33
LOAD NEGATIVE (long HFP)	LNDR	RR	C			Da		21	18-15
LOAD NEGATIVE (long)	LNDFR	RRE	C	FS		Da		B371	9-33
LOAD NEGATIVE (short BFP)	LNEBR	RRE	C			Db		B301	19-33
LOAD NEGATIVE (short HFP)	LNER	RR	C			Da		31	18-15
LOAD ON CONDITION (32)	LOC	RSY-b	CL	A			B ₂	EBF2	7-228
LOAD ON CONDITION (32)	LOCR	RRF-c	CL	A				B9F2	7-228
LOAD ON CONDITION (64)	LOCG	RSY-b	CL	A			B ₂	EBE2	7-228
LOAD ON CONDITION (64)	LOCGR	RRF-c	CL	A				B9E2	7-228
LOAD PAGE TABLE ENTRY ADDRESS	LPTEA	RRF-b	C	D2	P A ¹	SP SO	R ₂	B9AA	10-44
LOAD PAIR DISJOINT (32)	LPD	SSF	C	IA	A	SP	B ₁ B ₂	C84	7-229
LOAD PAIR DISJOINT (64)	LPDG	SSF	C	IA	A	SP	B ₁ B ₂	C85	7-229
LOAD PAIR FROM QUADWORD	LPQ	RXY-a	N	A	SP		B ₂	E38F	7-230
LOAD POSITIVE (32)	LPR	RR	C			IF		10	7-230
LOAD POSITIVE (64)	LPGR	RRE	C	N		IF		B900	7-230
LOAD POSITIVE (64←32)	LPGFR	RRE	C	N				B910	7-230
LOAD POSITIVE (extended BFP)	LPXBR	RRE	C		SP	Db		B340	19-33
LOAD POSITIVE (extended HFP)	LPXR	RRE	C		SP	Da		B360	18-16
LOAD POSITIVE (long BFP)	LPDBR	RRE	C			Db		B310	19-33
LOAD POSITIVE (long HFP)	LPDR	RR	C			Da		20	18-16
LOAD POSITIVE (long)	LPDFR	RRE	C	FS		Da		B370	9-34
LOAD POSITIVE (short BFP)	LPEBR	RRE	C			Db		B300	19-33
LOAD POSITIVE (short HFP)	LPER	RR	C			Da		30	18-16
LOAD PSW	LPSW	S	L		P A	SP	B ₂	82	10-47
LOAD PSW EXTENDED	LPSWE	S	L	N	P A	SP	B ₂	B2B2	10-48
LOAD REAL ADDRESS (32)	LRA	RX-a	C		P A ¹	SO	BP	B1	10-49
LOAD REAL ADDRESS (32)	LRAY	RXY-a	C	LD	P A ¹	SO	BP	E313	10-49
LOAD REAL ADDRESS (64)	LRAG	RXY-a	C	N	P A ¹		BP	E303	10-49
LOAD RELATIVE LONG (32)	LRL	RIL-b	GE	A	SP			C4D	7-214
LOAD RELATIVE LONG (64)	LGRL	RIL-b	GE	A	SP			C48	7-214
LOAD RELATIVE LONG (64←32)	LGFR	RIL-b	GE	A	SP			C4C	7-214
LOAD REVERSED (16)	LRVH	RXY-a	N3	A			B ₂	E31F	7-231

Figure B-1. Instructions Arranged by Name (Part 11 of 18)

Name	Mne-monic	Characteristics										Op-code	Page
LOAD REVERSED (32)	LRV	RXY-a	N3	A							B ₂	E31E	7-231
LOAD REVERSED (32)	LRVR	RRE	N3									B91F	7-230
LOAD REVERSED (64)	LRVG	RXY-a	N	A							B ₂	E30F	7-231
LOAD REVERSED (64)	LRVGR	RRE	N									B90F	7-231
LOAD ROUNDED (extended to long BFP)	LDXBR	RRE			SP	Db	Xi	Xo	Xu	Xx		B345	19-34
LOAD ROUNDED (extended to long BFP)	LDXBRA	RRF-e	F		SP	Db	Xi	Xo	Xu	Xx		B345	19-34
LOAD ROUNDED (extended to long DFP)	LDXTR	RRF-e	TF		SP	Dt	Xi	Xo	Xu	Xx Xq		B3DD	20-41
LOAD ROUNDED (extended to long HFP)	LDXR	RR			SP	Da		EO				25	18-16
LOAD ROUNDED (extended to long HFP)	LRDR	RR			SP	Da		EO				25	18-16
LOAD ROUNDED (extended to short BFP)	LEXBR	RRE			SP	Db	Xi	Xo	Xu	Xx		B346	19-34
LOAD ROUNDED (extended to short BFP)	LEXBRA	RRF-e	F		SP	Db	Xi	Xo	Xu	Xx		B346	19-34
LOAD ROUNDED (extended to short HFP)	LEXR	RRE			SP	Da		EO				B366	18-16
LOAD ROUNDED (long to short BFP)	LEDBR	RRE				Db	Xi	Xo	Xu	Xx		B344	19-34
LOAD ROUNDED (long to short BFP)	LEDBRA	RRF-e	F		SP	Db	Xi	Xo	Xu	Xx		B344	19-34
LOAD ROUNDED (long to short DFP)	LEDTR	RRF-e	TF			Dt	Xi	Xo	Xu	Xx Xq		B3D5	20-41
LOAD ROUNDED (long to short HFP)	LEDR	RR				Da		EO				35	18-16
LOAD ROUNDED (long to short HFP)	LRER	RR				Da		EO				35	18-16
LOAD USING REAL ADDRESS (32)	LURA	RRE		P A ¹	SP							B24B	10-53
LOAD USING REAL ADDRESS (64)	LURAG	RRE	N	P A ¹	SP							B905	10-53
LOAD ZERO (extended)	LZXR	RRE			SP	Da						B376	9-34
LOAD ZERO (long)	LZDR	RRE				Da						B375	9-34
LOAD ZERO (short)	LZER	RRE				Da						B374	9-34
MODIFY STACKED STATE	MSTA	RRE		A ¹	SP	SE					ST	B247	10-53
MODIFY SUBCHANNEL	MSCH	S	C	P A	SP	OP	¢	GS			B ₂	B232	14-8
MONITOR CALL	MC	SI			SP			ME				AF	7-231
MOVE (16←16)	MVHHI	SIL	GE	A							ST B ₁	E544	7-232
MOVE (32←16)	MVHI	SIL	GE	A							ST B ₁	E54C	7-233
MOVE (64←16)	MVGHI	SIL	GE	A							ST B ₁	E548	7-233
MOVE (character)	MVC	SS-a		A							ST B ₁ B ₂	D2	7-232
MOVE (immediate)	MVI	SI		A							ST B ₁	92	7-233
MOVE (immediate)	MVIY	SIY	LD	A							ST B ₁	EB52	7-233
MOVE INVERSE	MVCIN	SS-a		A							ST B ₁ B ₂	E8	7-233
MOVE LONG	MVCL	RR	C	A	SP	II					ST R ₁ R ₂	0E	7-234
MOVE LONG EXTENDED	MVCLE	RS-a	C	A	SP	IC					ST R ₁ R ₃	A8	7-238
MOVE LONG UNICODE	MVCLU	RSY-a	C E2	A	SP	IC					ST R ₁ R ₃	EB8E	7-241
MOVE NUMERICS	MVN	SS-a		A							ST B ₁ B ₂	D1	7-244
MOVE PAGE	MVPG	RRE	C	Q A	SP			G0			ST R ₁ R ₂	B254	10-54
MOVE STRING	MVST	RRE	C	A	SP	IC		G0			ST R ₁ R ₂	B255	7-245
MOVE TO PRIMARY	MVCP	SS-d	C	Q A		SO	¢				ST	DA	10-56
MOVE TO SECONDARY	MVCS	SS-d	C	Q A		SO	¢				ST	DB	10-56
MOVE WITH DESTINATION KEY	MVCDK	SSE		Q A				GM			ST B ₁ B ₂	E50F	10-58
MOVE WITH KEY	MVCK	SS-d	C	Q A							ST B ₁ B ₂	D9	10-59
MOVE WITH OFFSET	MVO	SS-b		A							ST B ₁ B ₂	F1	7-246
MOVE WITH OPTIONAL SPECIFICATIONS	MVCOS	SSF	C MO	Q A		SO		G0			ST B ₁ B ₂	C80	10-60
MOVE WITH SOURCE KEY	MVCSK	SSE		Q A				GM			ST B ₁ B ₂	E50E	10-63
MOVE ZONES	MVZ	SS-a		A							ST B ₁ B ₂	D3	7-246
MULTIPLY & ADD UNNORMALIZED (long to ext. HFP)	MAY	RXF	UE	A		Da					B ₂	ED3A	18-20
MULTIPLY & ADD UNNORMALIZED (long to ext. HFP)	MAYR	RRD	UE			Da						B33A	18-20
MULTIPLY (64←32)	M	RX-a		A	SP						B ₂	5C	7-247
MULTIPLY (64←32)	MFY	RXY-a	GE	A	SP						B ₂	E35C	7-247
MULTIPLY (64←32)	MR	RR			SP							1C	7-247
MULTIPLY (extended BFP)	MXBR	RRE			SP	Db	Xi	Xo	Xu	Xx		B34C	19-35
MULTIPLY (extended DFP)	MXTR	RRF-a	TF		SP	Dt	Xi	Xo	Xu	Xx		B3D8	20-43
MULTIPLY (extended DFP)	MXTRA	RRF-a	F		SP	Dt	Xi	Xo	Xu	Xx Xq		B3D8	20-43
MULTIPLY (extended HFP)	MXR	RR			SP	Da	EU	EO				26	18-17

Figure B-1. Instructions Arranged by Name (Part 12 of 18)

Name	Mne- monic	Characteristics						Op- code	Page
MULTIPLY (long BFP)	MDB	RXE		A	Db Xi	Xo Xu Xx		B ₂	ED1C 19-35
MULTIPLY (long BFP)	MDBR	RRE			Db Xi	Xo Xu Xx			B31C 19-35
MULTIPLY (long DFP)	MDTR	RRF-a	TF		Dt Xi	Xo Xu Xx			B3D0 20-43
MULTIPLY (long DFP)	MDTRA	RRF-a	F		Dt Xi	Xo Xu Xx Xq			B3D0 20-43
MULTIPLY (long HFP)	MD	RX-a		A	Da EU EO			B ₂	6C 18-17
MULTIPLY (long HFP)	MDR	RR			Da EU EO				2C 18-17
MULTIPLY (long to extended BFP)	MXDB	RXE		A SP	Db Xi			B ₂	ED07 19-35
MULTIPLY (long to extended BFP)	MXDBR	RRE		SP	Db Xi				B307 19-35
MULTIPLY (long to extended HFP)	MXD	RX-a		A SP	Da EU EO			B ₂	67 18-17
MULTIPLY (long to extended HFP)	MXDR	RR		SP	Da EU EO				27 18-17
MULTIPLY (short BFP)	MEEB	RXE		A	Db Xi	Xo Xu Xx		B ₂	ED17 19-35
MULTIPLY (short BFP)	MEEBR	RRE			Db Xi	Xo Xu Xx			B317 19-35
MULTIPLY (short HFP)	MEE	RXE		A	Da EU EO			B ₂	ED37 18-17
MULTIPLY (short HFP)	MEER	RRE			Da EU EO				B337 18-17
MULTIPLY (short to long BFP)	MDEB	RXE		A	Db Xi			B ₂	ED0C 19-35
MULTIPLY (short to long BFP)	MDEBR	RRE			Db Xi				B30C 19-35
MULTIPLY (short to long HFP)	MDE	RX-a		A	Da EU EO			B ₂	7C 18-17
MULTIPLY (short to long HFP)	MDER	RR			Da EU EO				3C 18-17
MULTIPLY (short to long HFP)	ME	RX-a		A	Da EU EO			B ₂	7C 18-17
MULTIPLY (short to long HFP)	MER	RR			Da EU EO				3C 18-17
MULTIPLY AND ADD (long BFP)	MADB	RXF		A	Db Xi	Xo Xu Xx		B ₂	ED1E 19-37
MULTIPLY AND ADD (long BFP)	MADBR	RRD			Db Xi	Xo Xu Xx			B31E 19-37
MULTIPLY AND ADD (long HFP)	MAD	RXF	HM	A	Da EU EO			B ₂	ED3E 18-19
MULTIPLY AND ADD (long HFP)	MADR	RRD	HM		Da EU EO				B33E 18-18
MULTIPLY AND ADD (short BFP)	MAEB	RXF		A	Db Xi	Xo Xu Xx		B ₂	ED0E 19-37
MULTIPLY AND ADD (short BFP)	MAEBR	RRD			Db Xi	Xo Xu Xx			B30E 19-37
MULTIPLY AND ADD (short HFP)	MAE	RXF	HM	A	Da EU EO			B ₂	ED2E 18-19
MULTIPLY AND ADD (short HFP)	MAER	RRD	HM		Da EU EO				B32E 18-18
MULTIPLY AND ADD UNNRM. (long to ext. high HFP)	MAYH	RXF	UE	A	Da			B ₂	ED3C 18-20
MULTIPLY AND ADD UNNRM. (long to ext. high HFP)	MAYHR	RRD	UE		Da				B33C 18-20
MULTIPLY AND ADD UNNRM. (long to ext. low HFP)	MAYL	RXF	UE	A	Da			B ₂	ED38 18-20
MULTIPLY AND ADD UNNRM. (long to ext. low HFP)	MAYLR	RRD	UE		Da				B338 18-20
MULTIPLY AND SUBTRACT (long BFP)	MSDB	RXF		A	Db Xi	Xo Xu Xx		B ₂	ED1F 19-37
MULTIPLY AND SUBTRACT (long BFP)	MSDBR	RRD			Db Xi	Xo Xu Xx			B31F 19-37
MULTIPLY AND SUBTRACT (long HFP)	MSD	RXF	HM	A	Da EU EO			B ₂	ED3F 18-19
MULTIPLY AND SUBTRACT (long HFP)	MSDR	RRD	HM		Da EU EO				B33F 18-19
MULTIPLY AND SUBTRACT (short BFP)	MSEB	RXF		A	Db Xi	Xo Xu Xx		B ₂	ED0F 19-37
MULTIPLY AND SUBTRACT (short BFP)	MSEBR	RRD			Db Xi	Xo Xu Xx			B30F 19-37
MULTIPLY AND SUBTRACT (short HFP)	MSE	RXF	HM	A	Da EU EO			B ₂	ED2F 18-19
MULTIPLY AND SUBTRACT (short HFP)	MSER	RRD	HM		Da EU EO				B32F 18-19
MULTIPLY DECIMAL	MP	SS-b		A SP	Dd		ST	B ₁ B ₂	FC 8-11
MULTIPLY HALFWORD (32)	MH	RX-a		A				B ₂	4C 7-248
MULTIPLY HALFWORD (32)	MHY	RXY-a	GE	A				B ₂	E37C 7-248
MULTIPLY HALFWORD IMMEDIATE (32)	MHI	RI-a							A7C 7-248
MULTIPLY HALFWORD IMMEDIATE (64)	MGHI	RI-a	N						A7D 7-248
MULTIPLY LOGICAL (128←64)	MLG	RXY-a	N	A SP				B ₂	E386 7-249
MULTIPLY LOGICAL (128←64)	MLGR	RRE	N	SP					B986 7-249
MULTIPLY LOGICAL (64←32)	ML	RXY-a	N3	A SP				B ₂	E396 7-249
MULTIPLY LOGICAL (64←32)	MLR	RRE	N3	SP					B996 7-248
MULTIPLY SINGLE (32)	MS	RX-a		A				B ₂	71 7-249
MULTIPLY SINGLE (32)	MSR	RRE							B252 7-249
MULTIPLY SINGLE (32)	MSY	RXY-a	LD	A				B ₂	E351 7-249
MULTIPLY SINGLE (64)	MSG	RXY-a	N	A				B ₂	E30C 7-249
MULTIPLY SINGLE (64)	MSGR	RRE	N						B90C 7-249
MULTIPLY SINGLE (64←32)	MSGF	RXY-a	N	A				B ₂	E31C 7-249

Figure B-1. Instructions Arranged by Name (Part 13 of 18)

Name	Mnemonic	Characteristics						Op-code	Page
MULTIPLY SINGLE (64←32)	MSGFR	RRE	N					B91C	7-249
MULTIPLY SINGLE IMMEDIATE (32)	MSFI	RIL-a	GE					C21	7-250
MULTIPLY SINGLE IMMEDIATE (64←32)	MSGFI	RIL-a	GE					C20	7-250
MULTIPLY UNNORM. (long to ext. high HFP)	MYH	RXF	UE	A	Da		B ₂	ED3D	18-21
MULTIPLY UNNORM. (long to ext. high HFP)	MYHR	RRD	UE		Da			B33D	18-21
MULTIPLY UNNORM. (long to ext. low HFP)	MYL	RXF	UE	A	Da		B ₂	ED39	18-21
MULTIPLY UNNORM. (long to ext. low HFP)	MYLR	RRD	UE		Da			B339	18-21
MULTIPLY UNNORMALIZED (long to ext. HFP)	MY	RXF	UE	A	SP Da		B ₂	ED3B	18-21
MULTIPLY UNNORMALIZED (long to ext. HFP)	MYR	RRD	UE		SP Da			B33B	18-21
OR (32)	O	RX-a	C	A			B ₂	56	7-250
OR (32)	OR	RR	C					16	7-250
OR (32)	ORK	RRF-a	C DO					B9F6	7-250
OR (32)	OY	RXY-a	C LD	A			B ₂	E356	7-250
OR (64)	OG	RXY-a	C N	A			B ₂	E381	7-250
OR (64)	OGR	RRE	C N					B981	7-250
OR (64)	OGRK	RRF-a	C DO					B9E6	7-250
OR (character)	OC	SS-a	C	A		ST	B ₁ B ₂	D6	7-251
OR (immediate)	OI	SI	C	A		ST	B ₁	96	7-251
OR (immediate)	OIY	SIY	C LD	A		ST	B ₁	EB56	7-251
OR IMMEDIATE (high high)	OIHH	RI-a	C N					A58	7-251
OR IMMEDIATE (high low)	OIHL	RI-a	C N					A59	7-251
OR IMMEDIATE (high)	OIHF	RIL-a	C EI					C0C	7-251
OR IMMEDIATE (low high)	OILH	RI-a	C N					A5A	7-252
OR IMMEDIATE (low low)	OILL	RI-a	C N					A5B	7-252
OR IMMEDIATE (low)	OILF	RIL-a	C EI					C0D	7-251
PACK	PACK	SS-b		A		ST	B ₁ B ₂	F2	7-252
PACK ASCII	PKA	SS-f	E2	A	SP	ST	B ₁ B ₂	E9	7-253
PACK UNICODE	PKU	SS-f	E2	A	SP	ST	B ₁ B ₂	E1	7-254
PAGE IN	PGIN	RRE	C ES	P A ¹				B22E	10-64
PAGE OUT	PGOUT	RRE	C ES	P A ¹				B22F	10-65
PERFORM CRYPTOGRAPHIC COMPUTATION	PCC	RRE	C M4	A	SP IC			B92C	7-255
PERFORM CRYPTOGRAPHIC KEY MGMT. OPERATIONS	PCKMO	RRE	M3	P A	SP GM	ST		B928	10-66
PERFORM FLOATING-POINT OPERATION	PFPO	E	PF		SP Da Xi			010A	9-34
PERFORM FRAME MANAGEMENT FUNCTION	PFMF	RRE	ED	P A ¹	SP II			B9AF	10-70
PERFORM LOCKED OPERATION	PLO	SS-e	C	A	SP \$ GM	ST	FC	EE	7-270
PERFORM TIMING FACILITY FUNCTION	PTFF	E	C TS	Q A	SP GM	ST		0104	10-73
PERFORM TOPOLOGY FUNCTION	PTF	RRE	C CT	P	SP			B9A2	10-80
POPULATION COUNT	POPCNT	RRE	C PK					B9E1	7-284
PREFETCH DATA	PFD	RXY-b	GE				B ₂	E336	7-285
PREFETCH DATA RELATIVE LONG	PFDR	RIL-c	GE					C62	7-285
PROGRAM CALL	PC	S		Q A ¹	Z ¹ T		B ST	B218	10-81
PROGRAM RETURN	PR	E	L	Q A ¹	SP Z ⁴ T		B ST	0101	10-94
PROGRAM TRANSFER	PT	RRE		Q A ¹	SP Z ² T		B	B228	10-98
PROGRAM TRANSFER WITH INSTANCE	PTI	RRE	RA	Q A ¹	SP Z ⁶ T		B	B99E	10-98
PURGE ALB	PALB	RRE		P		\$		B248	10-107
PURGE TLB	PTLB	S		P		\$		B20D	10-107
QUANTIZE (extended DFP)	QAXTR	RRF-b	TF		SP Dt Xi			B3FD	20-45
QUANTIZE (long DFP)	QADTR	RRF-b	TF		SP Dt Xi			B3F5	20-45
REROUND (extended DFP)	RRXTR	RRF-b	TF		SP Dt Xi			B3FF	20-47
REROUND (long DFP)	RRDTR	RRF-b	TF		SP Dt Xi			B3F7	20-47
RESET CHANNEL PATH	RCHP	S	C	P				B23B	14-9
RESET REFERENCE BIT EXTENDED	RRBE	RRE	C	P A ¹				B22A	10-107
RESET REFERENCE BITS MULTIPLE	RRBM	RRE	RB	P A ¹				B9AE	10-108
RESUME PROGRAM	RP	S	L	Q A	SP WE T		B	B277	10-108

Figure B-1. Instructions Arranged by Name (Part 14 of 18)

Name	Mnemonic	Characteristics						Op-code	Page
RESUME SUBCHANNEL	RSCH	S C	P		OP	¢ GS		B238	14-10
ROTATE LEFT SINGLE LOGICAL (32)	RLL	RSY-a N3						EB1D	7-286
ROTATE LEFT SINGLE LOGICAL (64)	RLLG	RSY-a N						EB1C	7-286
ROTATE THEN AND SELECTED BITS	RNSBG	RIE-f C GE						EC54	7-287
ROTATE THEN EXCLUSIVE OR SELECTED BITS	RXSBG	RIE-f C GE						EC57	7-287
ROTATE THEN INSERT SELECTED BITS	RISBG	RIE-f C GE						EC55	7-288
ROTATE THEN INSERT SELECTED BITS HIGH	RISBHG	RIE-f HW						EC5D	7-290
ROTATE THEN INSERT SELECTED BITS LOW	RISBLG	RIE-f HW						EC51	7-290
ROTATE THEN OR SELECTED BITS	ROSBG	RIE-f C GE						EC56	7-287
SEARCH STRING	SRST	RRE C	A SP	IC	G0		R ₂	B25E	7-291
SEARCH STRING UNICODE	SRSTU	RRE C E3	A SP	IC	G0		R ₁ R ₂	B9BE	7-293
SET ACCESS	SAR	RRE					U ₁	B24E	7-295
SET ADDRESS LIMIT	SAL	S	P	OP	¢ G1			B237	14-12
SET ADDRESS SPACE CONTROL	SAC	S	Q SP	SW	¢			B219	10-110
SET ADDRESS SPACE CONTROL FAST	SACF	S	Q SP	SW				B279	10-111
SET ADDRESSING MODE (24)	SAM24	E N3	SP	T				010C	7-295
SET ADDRESSING MODE (31)	SAM31	E N3	SP	T				010D	7-295
SET ADDRESSING MODE (64)	SAM64	E N		T				010E	7-295
SET BFP ROUNDING MODE (2 bit)	SRNM	S		Db				B299	9-45
SET BFP ROUNDING MODE (3 bit)	SRNMB	S F	SP	Db				B2B8	9-45
SET CHANNEL MONITOR	SCHM	S C	P A SP	OP	¢ GM		B ₂	B23C	14-13
SET CLOCK	SCK	S C	P A SP				B ₂	B204	10-112
SET CLOCK COMPARATOR	SCKC	S	P A SP				B ₂	B206	10-113
SET CLOCK PROGRAMMABLE FIELD	SCKPF	E	P SP		G0			0107	10-113
SET CPU TIMER	SPT	S	P A SP				B ₂	B208	10-114
SET DFP ROUNDING MODE	SRNMT	S TR		Dt				B2B9	9-45
SET FPC	SFPC	RRE	SP	Db				B384	9-46
SET FPC AND SIGNAL	SFASR	RRE XF	SP	Dt Xg				B385	9-46
SET PREFIX	SPX	S	P A SP	\$			B ₂	B210	10-114
SET PROGRAM MASK	SPM	RR L						04	7-296
SET PSW KEY FROM ADDRESS	SPKA	S	Q					B20A	10-115
SET SECONDARY ASN	SSAR	RRE	A ¹	Z ³ T ¢				B225	10-115
SET SECONDARY ASN WITH INSTANCE	SSAIR	RRE RA	A ¹	Z ⁷ T ¢				B99F	10-115
SET STORAGE KEY EXTENDED	SSKE	RRF-c C	P A ¹	II ¢				B22B	10-121
SET SYSTEM MASK	SSM	S	P A SP	SO			B ₂	80	10-124
SHIFT AND ROUND DECIMAL	SRP	SS-c C	A	Dd DF		ST	B ₁ B ₂	F0	8-12
SHIFT LEFT DOUBLE	SLDA	RS-a C	SP	IF				8F	7-296
SHIFT LEFT DOUBLE LOGICAL	SDDL	RS-a	SP					8D	7-297
SHIFT LEFT SINGLE (32)	SLA	RS-a C		IF				8B	7-297
SHIFT LEFT SINGLE (32)	SLAK	RSY-a C DO		IF				EBDD	7-297
SHIFT LEFT SINGLE (64)	SLAG	RSY-a C N		IF				EB0B	7-297
SHIFT LEFT SINGLE LOGICAL (32)	SLL	RS-a						89	7-298
SHIFT LEFT SINGLE LOGICAL (32)	SLLK	RSY-a DO						EBDF	7-298
SHIFT LEFT SINGLE LOGICAL (64)	SLLG	RSY-a N						EB0D	7-298
SHIFT RIGHT DOUBLE	SRDA	RS-a C	SP					8E	7-299
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS-a	SP					8C	7-299
SHIFT RIGHT SINGLE (32)	SRA	RS-a C						8A	7-300
SHIFT RIGHT SINGLE (32)	SRAK	RSY-a C DO						EBDC	7-300
SHIFT RIGHT SINGLE (64)	SRAG	RSY-a C N						EB0A	7-300
SHIFT RIGHT SINGLE LOGICAL (32)	SRL	RS-a						88	7-301
SHIFT RIGHT SINGLE LOGICAL (32)	SRLK	RSY-a DO						EBDE	7-301
SHIFT RIGHT SINGLE LOGICAL (64)	SRLG	RSY-a N						EB0C	7-301
SHIFT SIGNIFICAND LEFT (extended DFP)	SLXT	RXF TF	SP	Dt				ED48	20-49
SHIFT SIGNIFICAND LEFT (long DFP)	SLDT	RXF TF		Dt				ED40	20-49
SHIFT SIGNIFICAND RIGHT (extended DFP)	SRXT	RXF TF	SP	Dt				ED49	20-50

Figure B-1. Instructions Arranged by Name (Part 15 of 18)

Name	Mnemonic	Characteristics						Op-code	Page
SHIFT SIGNIFICAND RIGHT (long DFP)	SRDT	RXF	TF		Dt			ED41	20-50
SIGNAL PROCESSOR	SIGP	RS-a	C	P	\$			AE	10-124
SQUARE ROOT (extended BFP)	SQXBR	RRE		SP	Db Xi	Xx		B316	19-38
SQUARE ROOT (extended HFP)	SQXR	RRE		SP	Da	SQ		B336	18-23
SQUARE ROOT (long BFP)	SQDB	RXE		A	Db Xi	Xx	B ₂	ED15	19-38
SQUARE ROOT (long BFP)	SQDBR	RRE			Db Xi	Xx		B315	19-38
SQUARE ROOT (long HFP)	SQD	RXE		A	Da	SQ	B ₂	ED35	18-23
SQUARE ROOT (long HFP)	SQDR	RRE			Da	SQ		B244	18-23
SQUARE ROOT (short BFP)	SQEB	RXE		A	Db Xi	Xx	B ₂	ED14	19-38
SQUARE ROOT (short BFP)	SQEBR	RRE			Db Xi	Xx		B314	19-38
SQUARE ROOT (short HFP)	SQE	RXE		A	Da	SQ	B ₂	ED34	18-23
SQUARE ROOT (short HFP)	SQER	RRE			Da	SQ		B245	18-23
START SUBCHANNEL	SSCH	S	C	P A SP	OP	¢ GS	B ₂	B233	14-15
STORE (32)	ST	RX-a		A			ST B ₂	50	7-301
STORE (32)	STY	RXY-a	LD	A			ST B ₂	E350	7-301
STORE (64)	STG	RXY-a	N	A			ST B ₂	E324	7-301
STORE (long)	STD	RX-a		A	Da		ST B ₂	60	9-47
STORE (long)	STDY	RXY-a	LD	A	Da		ST B ₂	ED67	9-47
STORE (short)	STE	RX-a		A	Da		ST B ₂	70	9-47
STORE (short)	STEY	RXY-a	LD	A	Da		ST B ₂	ED66	9-47
STORE ACCESS MULTIPLE	STAM	RS-a		A SP			ST UB	9B	7-302
STORE ACCESS MULTIPLE	STAMY	RSY-a	LD	A SP			ST UB	EB9B	7-302
STORE CHANNEL PATH STATUS	STCPS	S		P A SP	¢		ST B ₂	B23A	14-17
STORE CHANNEL REPORT WORD	STCRW	S	C	P A SP	¢		ST B ₂	B239	14-17
STORE CHARACTER	STC	RX-a		A			ST B ₂	42	7-302
STORE CHARACTER	STCY	RXY-a	LD	A			ST B ₂	E372	7-302
STORE CHARACTER HIGH (8)	STCH	RXY-a	HW	A			ST B ₂	E3C3	7-303
STORE CHARACTERS UNDER MASK (high)	STCMH	RSY-b	N	A			ST B ₂	EB2C	7-303
STORE CHARACTERS UNDER MASK (low)	STCM	RS-b		A			ST B ₂	BE	7-303
STORE CHARACTERS UNDER MASK (low)	STCMY	RSY-b	LD	A			ST B ₂	EB2D	7-303
STORE CLOCK	STCK	S	C	A	\$		ST B ₂	B205	7-304
STORE CLOCK COMPARATOR	STCKC	S		P A SP			ST B ₂	B207	10-126
STORE CLOCK EXTENDED	STCKE	S	C	A	\$		ST B ₂	B278	7-305
STORE CLOCK FAST	STCKF	S	C SC	A			ST B ₂	B27C	7-304
STORE CONTROL (32)	STCTL	RS-a		P A SP			ST B ₂	B6	10-126
STORE CONTROL (64)	STCTG	RSY-a	N	P A SP			ST B ₂	EB25	10-126
STORE CPU ADDRESS	STAP	S		P A SP			ST B ₂	B212	10-126
STORE CPU ID	STIDP	S		P A SP			ST B ₂	B202	10-127
STORE CPU TIMER	STPT	S		P A SP			ST B ₂	B209	10-128
STORE FACILITY LIST	STFL	S	N3	P				B2B1	10-128
STORE FACILITY LIST EXTENDED	STFLE	S	C FL	A SP	G0		ST B ₂	B2B0	7-306
STORE FPC	STFPC	S		A	Db		ST B ₂	B29C	9-47
STORE HALFWORD	STH	RX-a		A			ST B ₂	40	7-307
STORE HALFWORD	STHY	RXY-a	LD	A			ST B ₂	E370	7-307
STORE HALFWORD HIGH (16)	STHH	RXY-a	HW	A			ST B ₂	E3C7	7-308
STORE HALFWORD RELATIVE LONG	STHRL	RIL-b	GE	A			ST	C47	7-307
STORE HIGH (32)	STFH	RXY-a	HW	A			ST B ₂	E3CB	7-308
STORE MULTIPLE (32)	STM	RS-a		A			ST B ₂	90	7-308
STORE MULTIPLE (32)	STMY	RSY-a	LD	A			ST B ₂	EB90	7-308
STORE MULTIPLE (64)	STMG	RSY-a	N	A			ST B ₂	EB24	7-308
STORE MULTIPLE HIGH	STMH	RSY-a	N	A			ST B ₂	EB26	7-309
STORE ON CONDITION (32)	STOC	RSY-b	CL	A			ST B ₂	EBF3	7-309
STORE ON CONDITION (64)	STOCG	RSY-b	CL	A			ST B ₂	EBE3	7-309
STORE PAIR TO QUADWORD	STPQ	RXY-a	N	A SP			ST B ₂	E38E	7-310
STORE PREFIX	STPX	S		P A SP			ST B ₂	B211	10-128

Figure B-1. Instructions Arranged by Name (Part 16 of 18)

Name	Mne- monic	Characteristics										Op- code	Page
STORE REAL ADDRESS	STRAG	SSE	N	P	A ¹	SP		ST	B ₁	BP	E502	10-129	
STORE RELATIVE LONG (32)	STRL	RIL-b	GE	A	SP			ST			C4F	7-301	
STORE RELATIVE LONG (64)	STGRL	RIL-b	GE	A	SP			ST			C4B	7-301	
STORE REVERSED (16)	STRVH	RXY-a	N3	A				ST		B ₂	E33F	7-310	
STORE REVERSED (32)	STRV	RXY-a	N3	A				ST		B ₂	E33E	7-310	
STORE REVERSED (64)	STRVG	RXY-a	N	A				ST		B ₂	E32F	7-311	
STORE SUBCHANNEL	STSCH	S	C	P	A	SP	OP ¢ GS	ST		B ₂	B234	14-18	
STORE SYSTEM INFORMATION	STSI	S	C	P	A	SP	GM	ST		B ₂	B27D	10-130	
STORE THEN AND SYSTEM MASK	STNSM	SI		P	A			ST	B ₁		AC	10-149	
STORE THEN OR SYSTEM MASK	STOSM	SI		P	A	SP		ST	B ₁		AD	10-149	
STORE USING REAL ADDRESS (32)	STURA	RRE		P	A ¹	SP		SU			B246	10-150	
STORE USING REAL ADDRESS (64)	STURG	RRE	N	P	A ¹	SP		SU			B925	10-150	
SUBTRACT (32)	S	RX-a	C	A			IF			B ₂	5B	7-311	
SUBTRACT (32)	SR	RR	C				IF				1B	7-311	
SUBTRACT (32)	SRK	RRF-a	C	DO			IF				B9F9	7-311	
SUBTRACT (32)	SY	RXY-a	C	LD	A		IF			B ₂	E35B	7-311	
SUBTRACT (64)	SG	RXY-a	C	N	A		IF			B ₂	E309	7-311	
SUBTRACT (64)	SGR	RRE	C	N			IF				B909	7-311	
SUBTRACT (64)	SGRK	RRF-a	C	DO			IF				B9E9	7-311	
SUBTRACT (64←32)	SGF	RXY-a	C	N	A		IF			B ₂	E319	7-311	
SUBTRACT (64←32)	SGFR	RRE	C	N			IF				B919	7-311	
SUBTRACT (extended BFP)	SXBR	RRE	C		SP		Db Xi Xo Xu Xx				B34B	19-39	
SUBTRACT (extended DFP)	SXTR	RRF-a	C	TF	SP		Dt Xi Xo Xu Xx				B3DB	20-51	
SUBTRACT (extended DFP)	SXTRA	RRF-a	C	F	SP		Dt Xi Xo Xu Xx Xq				B3DB	20-51	
SUBTRACT (long BFP)	SDB	RXE	C	A			Db Xi Xo Xu Xx			B ₂	ED1B	19-39	
SUBTRACT (long BFP)	SDBR	RRE	C				Db Xi Xo Xu Xx				B31B	19-39	
SUBTRACT (long DFP)	SDTR	RRF-a	C	TF			Dt Xi Xo Xu Xx				B3D3	20-51	
SUBTRACT (long DFP)	SDTRA	RRF-a	C	F			Dt Xi Xo Xu Xx Xq				B3D3	20-51	
SUBTRACT (short BFP)	SEB	RXE	C	A			Db Xi Xo Xu Xx			B ₂	ED0B	19-39	
SUBTRACT (short BFP)	SEBR	RRE	C				Db Xi Xo Xu Xx				B30B	19-39	
SUBTRACT DECIMAL	SP	SS-b	C	A			Dd DF	ST	B ₁	B ₂	FB	8-13	
SUBTRACT HALFWORD	SH	RX-a	C	A			IF			B ₂	4B	7-312	
SUBTRACT HALFWORD	SHY	RXY-a	C	LD	A		IF			B ₂	E37B	7-312	
SUBTRACT HIGH (32)	SHHHR	RRF-a	C	HW			IF				B9C9	7-312	
SUBTRACT HIGH (32)	SHHLR	RRF-a	C	HW			IF				B9D9	7-312	
SUBTRACT LOGICAL (32)	SL	RX-a	C	A						B ₂	5F	7-313	
SUBTRACT LOGICAL (32)	SLR	RR	C								1F	7-313	
SUBTRACT LOGICAL (32)	SLRK	RRF-a	C	DO							B9FB	7-313	
SUBTRACT LOGICAL (32)	SLY	RXY-a	C	LD	A					B ₂	E35F	7-313	
SUBTRACT LOGICAL (64)	SLG	RXY-a	C	N	A					B ₂	E30B	7-313	
SUBTRACT LOGICAL (64)	SLGR	RRE	C	N							B90B	7-313	
SUBTRACT LOGICAL (64)	SLGRK	RRF-a	C	DO							B9EB	7-313	
SUBTRACT LOGICAL (64←32)	SLGF	RXY-a	C	N	A					B ₂	E31B	7-313	
SUBTRACT LOGICAL (64←32)	SLGFR	RRE	C	N							B91B	7-313	
SUBTRACT LOGICAL HIGH (32)	SLHHHR	RRF-a	C	HW							B9CB	7-314	
SUBTRACT LOGICAL HIGH (32)	SLHHLR	RRF-a	C	HW							B9DB	7-314	
SUBTRACT LOGICAL IMMEDIATE (32)	SLFI	RIL-a	C	EI							C25	7-313	
SUBTRACT LOGICAL IMMEDIATE (64←32)	SLGFI	RIL-a	C	EI							C24	7-313	
SUBTRACT LOGICAL WITH BORROW (32)	SLB	RXY-a	C	N3	A					B ₂	E399	7-315	
SUBTRACT LOGICAL WITH BORROW (32)	SLBR	RRE	C	N3							B999	7-314	
SUBTRACT LOGICAL WITH BORROW (64)	SLBG	RXY-a	C	N	A					B ₂	E389	7-315	
SUBTRACT LOGICAL WITH BORROW (64)	SLBGR	RRE	C	N							B989	7-314	
SUBTRACT NORMALIZED (extended HFP)	SXR	RR	C	SP			Da EU EO LS				37	18-24	
SUBTRACT NORMALIZED (long HFP)	SD	RX-a	C	A			Da EU EO LS			B ₂	6B	18-24	
SUBTRACT NORMALIZED (long HFP)	SDR	RR	C				Da EU EO LS				2B	18-24	

Figure B-1. Instructions Arranged by Name (Part 17 of 18)

Name	Mne- monic	Characteristics							Op- code	Page			
SUBTRACT NORMALIZED (short HFP)	SE	RX-a	C	A	Da	EU	EO	LS	B ₂	7B	18-24		
SUBTRACT NORMALIZED (short HFP)	SER	RR	C		Da	EU	EO	LS		3B	18-24		
SUBTRACT UNNORMALIZED (short HFP)	SU	RX-a	C	A	Da		EO	LS	B ₂	7F	18-24		
SUBTRACT UNNORMALIZED (short HFP)	SUR	RR	C		Da		EO	LS		3F	18-24		
SUBTRACT UNNORMALIZED (long HFP)	SW	RX-a	C	A	Da		EO	LS	B ₂	6F	18-24		
SUBTRACT UNNORMALIZED (long HFP)	SWR	RR	C		Da		EO	LS		2F	18-24		
SUPERVISOR CALL	SVC	I					¢		U ₁	0A	7-315		
TEST ACCESS	TAR	RRE	C	A ¹						B24C	10-150		
TEST ADDRESSING MODE	TAM	E	C	N3						010B	7-315		
TEST AND SET	TS	S	C	A			\$		ST	B ₂	93	7-316	
TEST BLOCK	TB	RRE	C	P A ¹	II		\$	G0			B22C	10-152	
TEST DATA CLASS (extended BFP)	TCXB	RXE	C		SP	Db					ED12	19-39	
TEST DATA CLASS (extended DFP)	TDCXT	RXE	C	TF		SP	Dt				ED58	20-51	
TEST DATA CLASS (long BFP)	TCDB	RXE	C			Db					ED11	19-39	
TEST DATA CLASS (long DFP)	TDCDT	RXE	C	TF		Dt					ED54	20-51	
TEST DATA CLASS (short BFP)	TCEB	RXE	C			Db					ED10	19-39	
TEST DATA CLASS (short DFP)	TDCET	RXE	C	TF		Dt					ED50	20-51	
TEST DATA GROUP (extended DFP)	TDGXT	RXE	C	TF	SP	Dt					ED59	20-52	
TEST DATA GROUP (long DFP)	TDGDT	RXE	C	TF		Dt					ED55	20-52	
TEST DATA GROUP (short DFP)	TDGET	RXE	C	TF		Dt					ED51	20-52	
TEST DECIMAL	TP	RSL	C	E2	A					B ₁ B ₂	EBC0	8-13	
TEST PENDING INTERRUPTION	TPI	S	C		P A ⁺ 1	SP		¢	ST	B ₂	B236	14-19	
TEST PROTECTION	TPROT	SSE	C		P A ¹					B ₁	E501	10-155	
TEST SUBCHANNEL	TSCH	S	C		P A	SP	OP	¢	GS	ST	B ₂	B235	14-20
TEST UNDER MASK	TM	SI	C		A					B ₁	91	7-316	
TEST UNDER MASK	TMY	SIY	C	LD	A					B ₁	EB51	7-316	
TEST UNDER MASK (high high)	TMHH	RI-a	C	N							A72	7-316	
TEST UNDER MASK (high low)	TMHL	RI-a	C	N							A73	7-316	
TEST UNDER MASK (low high)	TMLH	RI-a	C	N							A70	7-316	
TEST UNDER MASK (low low)	TMLL	RI-a	C	N							A71	7-316	
TEST UNDER MASK HIGH	TMH	RI-a	C								A70	7-316	
TEST UNDER MASK LOW	TML	RI-a	C								A71	7-316	
TRACE (32)	TRACE	RS-a			P A	SP	T	¢		B ₂	99	10-157	
TRACE (64)	TRACG	RSY-a		N	P A	SP	T	¢		B ₂	EB0F	10-157	
TRANSLATE	TR	SS-a			A				ST	B ₁ B ₂	DC	7-317	
TRANSLATE AND TEST	TRT	SS-a	C		A			GM		B ₁ B ₂	DD	7-318	
TRANSLATE AND TEST EXTENDED	TRTE	RRF-c	C	PE	A	SP	IC		ST	RM	B9BF	7-319	
TRANSLATE AND TEST REVERSE	TRTR	SS-a	C	E3	A			GM		B ₁ B ₂	D0	7-323	
TRANSLATE AND TEST REVERSE EXTENDED	TRTRE	RRF-c	C	PE	A	SP	IC		ST	RM	B9BD	7-319	
TRANSLATE EXTENDED	TRE	RRE	C		A	SP	IC		ST	R ₁ R ₂	B2A5	7-324	
TRANSLATE ONE TO ONE	TROO	RRF-c	C	E2	A	SP	IC		ST	RM R ₂	B993	7-327	
TRANSLATE ONE TO TWO	TROT	RRF-c	C	E2	A	SP	IC		ST	RM R ₂	B992	7-327	
TRANSLATE TWO TO ONE	TRTO	RRF-c	C	E2	A	SP	IC		ST	RM R ₂	B991	7-327	
TRANSLATE TWO TO TWO	TRTT	RRF-c	C	E2	A	SP	IC		ST	RM R ₂	B990	7-327	
TRAP	TRAP2	E			A		SO	T	B	ST	01FF	10-159	
TRAP	TRAP4	S			A		SO	T	B	ST	B2FF	10-159	
UNPACK	UNPK	SS-b			A				ST	B ₁ B ₂	F3	7-331	
UNPACK ASCII	UNPKA	SS-a	C	E2	A	SP			ST	B ₁ B ₂	EA	7-332	
UNPACK UNICODE	UNPKU	SS-a	C	E2	A	SP			ST	B ₁ B ₂	E2	7-333	
UPDATE TREE	UPT	E	C		A	SP	II		GM	I4	ST	0102	7-334
ZERO AND ADD	ZAP	SS-b	C		A		Dd	DF	ST	B ₁ B ₂	F8	8-13	

Figure B-1. Instructions Arranged by Name (Part 18 of 18)

Instructions Arranged by Mnemonic

Mnemonic	Name	Characteristics						Op-code	Page
	DIAGNOSE	DM	P DM				MD	83	10-19
A	ADD (32)	RX-a C	A	IF			B ₂	5A	7-25
AD	ADD NORMALIZED (long HFP)	RX-a C	A	Da EU EO LS			B ₂	6A	18-8
ADB	ADD (long BFP)	RXE C	A	Db Xi Xo Xu Xx			B ₂	ED1A	19-11
ADBR	ADD (long BFP)	RRE C		Db Xi Xo Xu Xx				B31A	19-11
ADR	ADD NORMALIZED (long HFP)	RR C		Da EU EO LS				2A	18-8
ADTR	ADD (long DFP)	RRF-a C TF		Dt Xi Xo Xu Xx				B3D2	20-19
ADTRA	ADD (long DFP)	RRF-a C F		Dt Xi Xo Xu Xx Xq				B3D2	20-19
AE	ADD NORMALIZED (short HFP)	RX-a C	A	Da EU EO LS			B ₂	7A	18-8
AEB	ADD (short BFP)	RXE C	A	Db Xi Xo Xu Xx			B ₂	ED0A	19-11
AEBR	ADD (short BFP)	RRE C		Db Xi Xo Xu Xx				B30A	19-11
AER	ADD NORMALIZED (short HFP)	RR C		Da EU EO LS				3A	18-8
AFI	ADD IMMEDIATE (32)	RIL-a C EI		IF				C29	7-25
AG	ADD (64)	RXY-a C N	A	IF			B ₂	E308	7-25
AGF	ADD (64←32)	RXY-a C N	A	IF			B ₂	E318	7-25
AGFI	ADD IMMEDIATE (64←32)	RIL-a C EI		IF				C28	7-25
AGFR	ADD (64←32)	RRE C N		IF				B918	7-25
AGHI	ADD HALFWORD IMMEDIATE (64)	RI-a C N		IF				A7B	7-27
AGHIK	ADD IMMEDIATE (64←16)	RIE-d C DO		IF				ECD9	7-25
AGR	ADD (64)	RRE C N		IF				B908	7-25
AGRK	ADD (64)	RRF-a C DO		IF				B9E8	7-25
AGSI	ADD IMMEDIATE (64←8)	SIY C GE	A	IF		ST	B ₁	EB7A	7-25
AH	ADD HALFWORD	RX-a C	A	IF			B ₂	4A	7-27
AHHHR	ADD HIGH (32)	RRF-a C HW		IF				B9C8	7-27
AHHLR	ADD HIGH (32)	RRF-a C HW		IF				B9D8	7-27
AHI	ADD HALFWORD IMMEDIATE (32)	RI-a C		IF				A7A	7-27
AHIK	ADD IMMEDIATE (32←16)	RIE-d C DO		IF				ECD8	7-25
AHY	ADD HALFWORD	RXY-a C LD	A	IF			B ₂	E37A	7-27
AIH	ADD IMMEDIATE HIGH (32)	RIL-a C HW		IF				CC8	7-25
AL	ADD LOGICAL (32)	RX-a C	A				B ₂	5E	7-28
ALC	ADD LOGICAL WITH CARRY (32)	RXY-a C N3	A				B ₂	E398	7-29
ALCG	ADD LOGICAL WITH CARRY (64)	RXY-a C N	A				B ₂	E388	7-29
ALCGR	ADD LOGICAL WITH CARRY (64)	RRE C N						B988	7-29
ALCR	ADD LOGICAL WITH CARRY (32)	RRE C N3						B998	7-29
ALFI	ADD LOGICAL IMMEDIATE (32)	RIL-a C EI						C2B	7-28
ALG	ADD LOGICAL (64)	RXY-a C N	A				B ₂	E30A	7-28
ALGF	ADD LOGICAL (64←32)	RXY-a C N	A				B ₂	E31A	7-28
ALGFI	ADD LOGICAL IMMEDIATE (64←32)	RIL-a C EI						C2A	7-28
ALGFR	ADD LOGICAL (64←32)	RRE C N						B91A	7-28
ALGHSIK	ADD LOGICAL WITH SIGNED IMMEDIATE (64←16)	RIE-d C DO						ECDB	7-30
ALGR	ADD LOGICAL (64)	RRE C N						B90A	7-28
ALGRK	ADD LOGICAL (64)	RRF-a C DO						B9EA	7-28
ALGSI	ADD LOGICAL WITH SIGNED IMMEDIATE (64←8)	SIY C GE	A			ST	B ₁	EB7E	7-30
ALHHHR	ADD LOGICAL HIGH (32)	RRF-a C HW						B9CA	7-29
ALHHLR	ADD LOGICAL HIGH (32)	RRF-a C HW						B9DA	7-29
ALHSIK	ADD LOGICAL WITH SIGNED IMMEDIATE (32←16)	RIE-d C DO						ECDA	7-30
ALR	ADD LOGICAL (32)	RR C						1E	7-28
ALRK	ADD LOGICAL (32)	RRF-a C DO						B9FA	7-28
ALSI	ADD LOGICAL WITH SIGNED IMMEDIATE (32←8)	SIY C GE	A			ST	B ₁	EB6E	7-30
ALSIH	ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	RIL-a C HW						CCA	7-31
ALSIHN	ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	RIL-a HW						CCB	7-31

Figure B-2. Instructions Arranged by Mnemonic (Part 1 of 18)

Mnemonic	Name	Characteristics						Op-code	Page
ALY	ADD LOGICAL (32)	RXY-a C LD	A				B ₂	E35E	7-28
AP	ADD DECIMAL	SS-b C	A	Dd DF		ST	B ₁ B ₂	FA	8-6
AR	ADD (32)	RR C		IF				1A	7-24
ARK	ADD (32)	RRF-a C DO		IF				B9F8	7-25
ASI	ADD IMMEDIATE (32←8)	SIY C GE	A	IF		ST	B ₁	EB6A	7-25
AU	ADD UNNORMALIZED (short HFP)	RX-a C	A	Da EO LS			B ₂	7E	18-9
AUR	ADD UNNORMALIZED (short HFP)	RR C		Da EO LS				3E	18-9
AW	ADD UNNORMALIZED (long HFP)	RX-a C	A	Da EO LS			B ₂	6E	18-9
AWR	ADD UNNORMALIZED (long HFP)	RR C		Da EO LS				2E	18-9
AXBR	ADD (extended BFP)	RRE C	SP	Db Xi Xo Xu Xx				B34A	19-11
AXR	ADD NORMALIZED (extended HFP)	RR C	SP	Da EU EO LS				36	18-8
AXTR	ADD (extended DFP)	RRF-a C TF	SP	Dt Xi Xo Xu Xx				B3DA	20-19
AXTRA	ADD (extended DFP)	RRF-a C F	SP	Dt Xi Xo Xu Xx Xq				B3DA	20-19
AY	ADD (32)	RXY-a C LD	A	IF			B ₂	E35A	7-25
BAKR	BRANCH AND STACK	RRE	A ¹	Z ⁵ T	B ST			B240	10-10
BAL	BRANCH AND LINK	RX-a			B			45	7-33
BALR	BRANCH AND LINK	RR		T	B			05	7-33
BAS	BRANCH AND SAVE	RX-a			B			4D	7-34
BASR	BRANCH AND SAVE	RR		T	B			0D	7-34
BASSM	BRANCH AND SAVE AND SET MODE	RR		T	B			0C	7-35
BC	BRANCH ON CONDITION	RX-b			B			47	7-36
BCR	BRANCH ON CONDITION	RR		ϕ ¹	B			07	7-36
BCT	BRANCH ON COUNT (32)	RX-a			B			46	7-37
BCTG	BRANCH ON COUNT (64)	RXY-a N			B			E346	7-37
BCTGR	BRANCH ON COUNT (64)	RRE N			B			B946	7-37
BCTR	BRANCH ON COUNT (32)	RR			B			06	7-37
BRAS	BRANCH RELATIVE AND SAVE	Rl-b			B			A75	7-39
BRASL	BRANCH RELATIVE AND SAVE LONG	RIL-b N3			B			C05	7-39
BRC	BRANCH RELATIVE ON CONDITION	Rl-c			B			A74	7-40
BRCL	BRANCH RELATIVE ON CONDITION LONG	RIL-c N3			B			C04	7-40
BRCT	BRANCH RELATIVE ON COUNT (32)	Rl-b			B			A76	7-41
BRCTG	BRANCH RELATIVE ON COUNT (64)	Rl-b N			B			A77	7-41
BRCTH	BRANCH RELATIVE ON COUNT HIGH (32)	RIL-b HW			B			CC6	7-41
BRXH	BRANCH RELATIVE ON INDEX HIGH (32)	RSI			B			84	7-41
BRXHG	BRANCH RELATIVE ON INDEX HIGH (64)	RIE-e N			B			EC44	7-41
BRXLE	BRANCH RELATIVE ON INDEX LOW OR EQ. (32)	RSI			B			85	7-41
BRXLG	BRANCH RELATIVE ON INDEX LOW OR EQ. (64)	RIE-e N			B			EC45	7-42
BSA	BRANCH AND SET AUTHORITY	RRE	Q A ¹	SO T	B			B25A	10-6
BSG	BRANCH IN SUBSPACE GROUP	RRE	A ¹	SO T	B	R ₂		B258	10-13
BSM	BRANCH AND SET MODE	RR		T	B			0B	7-36
BXH	BRANCH ON INDEX HIGH (32)	RS-a			B			86	7-38
BXHG	BRANCH ON INDEX HIGH (64)	RSY-a N			B			EB44	7-38
BXLE	BRANCH ON INDEX LOW OR EQUAL (32)	RS-a			B			87	7-38
BXLEG	BRANCH ON INDEX LOW OR EQUAL (64)	RSY-a N			B			EB45	7-38
C	COMPARE (32)	RX-a C	A				B ₂	59	7-112
CD	COMPARE (long HFP)	RX-a C	A	Da			B ₂	69	18-10
CDB	COMPARE (long BFP)	RXE C	A	Db Xi			B ₂	ED19	19-16
CDBR	COMPARE (long BFP)	RRE C		Db Xi				B319	19-16
CDFBR	CONVERT FROM FIXED (32 to long BFP)	RRE		Db				B395	19-18
CDFBRA	CONVERT FROM FIXED (32 to long BFP)	RRF-e F	SP	Db				B395	19-18
CDFR	CONVERT FROM FIXED (32 to long HFP)	RRE		Da				B3B5	18-10
CDFTR	CONVERT FROM FIXED (32 to long DFP)	RRE F		Dt				B951	20-24
CDGBR	CONVERT FROM FIXED (64 to long BFP)	RRE N		Db Xx				B3A5	19-18
CDGBRA	CONVERT FROM FIXED (64 to long BFP)	RRF-e F	SP	Db Xx				B3A5	19-18
CDGR	CONVERT FROM FIXED (64 to long HFP)	RRE N		Da				B3C5	18-10

Figure B-2. Instructions Arranged by Mnemonic (Part 2 of 18)

Mne- monic	Name	Characteristics						Op- code	Page
CDGTR	CONVERT FROM FIXED (64 to long DFP)	RRE	TF		Dt	Xx		B3F1	20-24
CDGTRA	CONVERT FROM FIXED (64 to long DFP)	RRF-e	F		Dt	Xx Xq		B3F1	20-24
CDLFBR	CONVERT FROM LOGICAL (32 to long BFP)	RRF-e	F	SP	Db			B391	19-20
CDLFTR	CONVERT FROM LOGICAL (32 to long DFP)	RRF-e	F		Dt			B953	20-25
CDLGBR	CONVERT FROM LOGICAL (64 to long BFP)	RRF-e	F	SP	Db	Xx		B3A1	19-20
CDLGTR	CONVERT FROM LOGICAL (64 to long DFP)	RRF-e	F		Dt	Xx Xq		B952	20-25
CDR	COMPARE (long HFP)	RR	C		Da			29	18-10
CDS	COMPARE DOUBLE AND SWAP (32)	RS-a	C	A SP	\$		ST	B ₂	BB
CDSG	COMPARE DOUBLE AND SWAP (64)	RSY-a	C N	A SP	\$		ST	B ₂	EB3E
CDSTR	CONVERT FROM SIGNED PACKED (64 to long DFP)	RRE	TF		Dt Dd			B3F3	20-26
CDSY	COMPARE DOUBLE AND SWAP (32)	RSY-a	C LD	A SP	\$		ST	B ₂	EB31
CDTR	COMPARE (long DFP)	RRE	C TF		Dt Xi			B3E4	20-21
CDUTR	CONVERT FROM UNSIGNED PACKED (64 to long DFP)	RRE	TF		Dt Dd			B3F2	20-27
CE	COMPARE (short HFP)	RX-a	C	A	Da			B ₂	79
CEB	COMPARE (short BFP)	RXE	C	A	Db Xi			B ₂	ED09
CEBR	COMPARE (short BFP)	RRE	C		Db Xi			B309	19-16
CEDTR	COMPARE BIASED EXPONENT (long DFP)	RRE	C TF		Dt			B3F4	20-23
CEFBR	CONVERT FROM FIXED (32 to short BFP)	RRE			Db	Xx		B394	19-18
CEFBRA	CONVERT FROM FIXED (32 to short BFP)	RRF-e	F	SP	Db	Xx		B394	19-18
CEFR	CONVERT FROM FIXED (32 to short HFP)	RRE			Da			B3B4	18-10
CEGBR	CONVERT FROM FIXED (64 to short BFP)	RRE	N		Db	Xx		B3A4	19-18
CEGBRA	CONVERT FROM FIXED (64 to short BFP)	RRF-e	F	SP	Db	Xx		B3A4	19-18
CEGR	CONVERT FROM FIXED (64 to short HFP)	RRE	N		Da			B3C4	18-10
CELFBR	CONVERT FROM LOGICAL (32 to short BFP)	RRF-e	F	SP	Db	Xx		B390	19-20
CELGBR	CONVERT FROM LOGICAL (64 to short BFP)	RRF-e	F	SP	Db	Xx		B3A0	19-20
CER	COMPARE (short HFP)	RR	C		Da			39	18-10
CEXTR	COMPARE BIASED EXPONENT (extended DFP)	RRE	C TF	SP	Dt			B3FC	20-23
CFC	COMPARE AND FORM CODEWORD	S	C	A SP	II	GM I1		B21A	7-115
CFDBR	CONVERT TO FIXED (long BFP to 32)	RRF-e	C	SP	Db Xi	Xx		B399	19-21
CFDBRA	CONVERT TO FIXED (long BFP to 32)	RRF-e	C F	SP	Db Xi	Xx		B399	19-21
CFDR	CONVERT TO FIXED (long HFP to 32)	RRF-e	C	SP	Da			B3B9	18-11
CFDTR	CONVERT TO FIXED (long DFP to 32)	RRF-e	C F		Dt Xi	Xx		B941	20-27
CFEBR	CONVERT TO FIXED (short BFP to 32)	RRF-e	C	SP	Db Xi	Xx		B398	19-21
CFEBRA	CONVERT TO FIXED (short BFP to 32)	RRF-e	C F	SP	Db Xi	Xx		B398	19-21
CFER	CONVERT TO FIXED (short HFP to 32)	RRF-e	C	SP	Da			B3B8	18-11
CFI	COMPARE IMMEDIATE (32)	RIL-a	C EI					C2D	7-112
CFXBR	CONVERT TO FIXED (extended BFP to 32)	RRF-e	C	SP	Db Xi	Xx		B39A	19-21
CFXBRA	CONVERT TO FIXED (extended BFP to 32)	RRF-e	C F	SP	Db Xi	Xx		B39A	19-21
CFXR	CONVERT TO FIXED (extended HFP to 32)	RRF-e	C	SP	Da			B3BA	18-11
CFXTR	CONVERT TO FIXED (extended DFP to 32)	RRF-e	C F	SP	Dt Xi	Xx		B949	20-27
CG	COMPARE (64)	RXY-a	C N	A				B ₂	E320
CGDBR	CONVERT TO FIXED (long BFP to 64)	RRF-e	C N	SP	Db Xi	Xx		B3A9	19-21
CGDBRA	CONVERT TO FIXED (long BFP to 64)	RRF-e	C F	SP	Db Xi	Xx		B3A9	19-21
CGDR	CONVERT TO FIXED (long HFP to 64)	RRF-e	C N	SP	Da			B3C9	18-11
CGDTR	CONVERT TO FIXED (long DFP to 64)	RRF-e	C TF		Dt Xi	Xx		B3E1	20-27
CGDTRA	CONVERT TO FIXED (long DFP to 64)	RRF-e	C F		Dt Xi	Xx		B3E1	20-27
CGEBR	CONVERT TO FIXED (short BFP to 64)	RRF-e	C N	SP	Db Xi	Xx		B3A8	19-21
CGEBRA	CONVERT TO FIXED (short BFP to 64)	RRF-e	C F	SP	Db Xi	Xx		B3A8	19-21
CGER	CONVERT TO FIXED (short HFP to 64)	RRF-e	C N	SP	Da			B3C8	18-11
CGF	COMPARE (64←32)	RXY-a	C N	A				B ₂	E330
CGFI	COMPARE IMMEDIATE (64←32)	RIL-a	C EI					C2C	7-112
CGFR	COMPARE (64←32)	RRE	C N					B930	7-112
CGFRL	COMPARE RELATIVE LONG (64←32)	RIL-b	C GE	A SP				C6C	7-112
CGH	COMPARE HALFWORD (64←16)	RXY-a	C GE	A				B ₂	E334

Figure B-2. Instructions Arranged by Mnemonic (Part 3 of 18)

Mnemonic	Name	Characteristics						Op-code	Page
CGHI	COMPARE HALFWORD IMMEDIATE (64←16)	RI-a C N						A7F	7-128
CGHRL	COMPARE HALFWORD RELATIVE LONG (64←16)	RIL-b C GE	A					C64	7-128
CGHSI	COMPARE HALFWORD IMMEDIATE (64←16)	SIL C GE	A				B ₁	E558	7-128
CGIB	COMPARE IMMEDIATE AND BRANCH (64←8)	RIS GE				B		ECFC	7-113
CGIJ	COMPARE IMMEDIATE AND BRANCH RELATIVE (64←8)	RIE-c GE				B		EC7C	7-114
CGIT	COMPARE IMMEDIATE AND TRAP (64←16)	RIE-a GE			Dc			EC70	7-127
CGR	COMPARE (64)	RRE C N						B920	7-112
CGRB	COMPARE AND BRANCH (64)	RRS GE				B		ECE4	7-113
CGRJ	COMPARE AND BRANCH RELATIVE (64)	RIE-b GE				B		EC64	7-113
CGRL	COMPARE RELATIVE LONG (64)	RIL-b C GE	A SP					C68	7-112
CGRT	COMPARE AND TRAP (64)	RRF-c GE			Dc			B960	7-127
CGXBR	CONVERT TO FIXED (extended BFP to 64)	RRF-e C N	SP	Db Xi	Xx			B3AA	19-21
CGXBRA	CONVERT TO FIXED (extended BFP to 64)	RRF-e C F	SP	Db Xi	Xx			B3AA	19-21
CGXR	CONVERT TO FIXED (extended HFP to 64)	RRF-e C N	SP	Da				B3CA	18-11
CGXTR	CONVERT TO FIXED (extended DFP to 64)	RRF-e C TF	SP	Dt Xi	Xx			B3E9	20-27
CGXTRA	CONVERT TO FIXED (extended DFP to 64)	RRF-e C F	SP	Dt Xi	Xx			B3E9	20-27
CH	COMPARE HALFWORD (32←16)	RX-a C	A				B ₂	49	7-128
CHF	COMPARE HIGH (32)	RXY-a C HW	A				B ₂	E3CD	7-129
CHHR	COMPARE HIGH (32)	RRE C HW						B9CD	7-129
CHHSI	COMPARE HALFWORD IMMEDIATE (16)	SIL C GE	A				B ₁	E554	7-128
CHI	COMPARE HALFWORD IMMEDIATE (32←16)	RI-a C						A7E	7-128
CHLR	COMPARE HIGH (32)	RRE C HW						B9DD	7-129
CHRL	COMPARE HALFWORD RELATIVE LONG (32←16)	RIL-b C GE	A					C65	7-128
CHSI	COMPARE HALFWORD IMMEDIATE (32←16)	SIL C GE	A				B ₁	E55C	7-128
CHY	COMPARE HALFWORD (32←16)	RXY-a C LD	A				B ₂	E379	7-128
CIB	COMPARE IMMEDIATE AND BRANCH (32←8)	RIS GE				B		ECFE	7-113
CIH	COMPARE IMMEDIATE HIGH (32)	RIE-a C HW						CCD	7-129
CIJ	COMPARE IMMEDIATE AND BRANCH RELATIVE (32←8)	RIE-c GE				B		EC7E	7-114
CIT	COMPARE IMMEDIATE AND TRAP (32←16)	RIE-a GE			Dc			EC72	7-127
CKSM	CHECKSUM	RRE C	A SP	IC			R ₂	B241	7-43
CL	COMPARE LOGICAL (32)	RX-a C	A				B ₂	55	7-130
CLC	COMPARE LOGICAL (character)	SS-a C	A				B ₁ B ₂	D5	7-130
CLCL	COMPARE LOGICAL LONG	RR C	A SP	II			R ₁ R ₂	0F	7-135
CLCLE	COMPARE LOGICAL LONG EXTENDED	RS-a C	A SP	IC			R ₁ R ₃	A9	7-138
CLCLU	COMPARE LOGICAL LONG UNICODE	RSY-a C E2	A SP	IC			R ₁ R ₂	EB8F	7-140
CLFDBR	CONVERT TO LOGICAL (long BFP to 32)	RRF-e C F	SP	Db Xi	Xx			B39D	19-23
CLFDTR	CONVERT TO LOGICAL (long DFP to 32)	RRF-e C F		Dt Xi	Xx			B943	20-29
CLFEBR	CONVERT TO LOGICAL (short BFP to 32)	RRF-e C F	SP	Db Xi	Xx			B39C	19-23
CLFHSI	COMPARE LOGICAL IMMEDIATE (32←16)	SIL C GE	A				B ₁	E55D	7-130
CLFI	COMPARE LOGICAL IMMEDIATE (32)	RIL-a C EI						C2F	7-130
CLFIT	COMPARE LOGICAL IMMEDIATE AND TRAP (32←16)	RIE-a GE			Dc			EC73	7-133
CLFXBR	CONVERT TO LOGICAL (extended BFP to 32)	RRF-e C F	SP	Db Xi	Xx			B39E	19-23
CLFXTR	CONVERT TO LOGICAL (extended DFP to 32)	RRF-e C F	SP	Dt Xi	Xx			B94B	20-29
CLG	COMPARE LOGICAL (64)	RXY-a C N	A				B ₂	E321	7-130
CLGDBR	CONVERT TO LOGICAL (long BFP to 64)	RRF-e C F	SP	Db Xi	Xx			B3AD	19-23
CLGDTR	CONVERT TO LOGICAL (long DFP to 64)	RRF-e C F		Dt Xi	Xx			B942	20-29
CLGEBR	CONVERT TO LOGICAL (short BFP to 64)	RRF-e C F	SP	Db Xi	Xx			B3AC	19-23
CLGF	COMPARE LOGICAL (64←32)	RXY-a C N	A				B ₂	E331	7-130
CLGFI	COMPARE LOGICAL IMMEDIATE (64←32)	RIL-a C EI						C2E	7-130
CLGFR	COMPARE LOGICAL (64←32)	RRE C N						B931	7-130
CLGFRL	COMPARE LOGICAL RELATIVE LONG (64←32)	RIL-b C GE	A SP					C6E	7-131
CLGHRL	COMPARE LOGICAL RELATIVE LONG (64←16)	RIL-b C GE	A					C66	7-131

Figure B-2. Instructions Arranged by Mnemonic (Part 4 of 18)

Mnemonic	Name	Characteristics						Op-code	Page
CLGHSI	COMPARE LOGICAL IMMEDIATE (64←16)	SIL C GE	A				B ₁	E559	7-130
CLGIB	COMPARE LOGICAL IMMEDIATE AND BRANCH (64←8)	RIS GE				B		ECFD	7-132
CLGIJ	COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (64←8)	RIE-c GE				B		EC7D	7-132
CLGIT	COMPARE LOGICAL IMMEDIATE AND TRAP (64←16)	RIE-a GE			Dc			EC71	7-133
CLGR	COMPARE LOGICAL (64)	RRE C N						B921	7-130
CLGRB	COMPARE LOGICAL AND BRANCH (64)	RRS GE				B		ECE5	7-132
CLGRJ	COMPARE LOGICAL AND BRANCH RELATIVE (64)	RIE-b GE				B		EC65	7-132
CLGRL	COMPARE LOGICAL RELATIVE LONG (64)	RIL-b C GE	A SP					C6A	7-131
CLGRT	COMPARE LOGICAL AND TRAP (64)	RRF-c GE			Dc			B961	7-133
CLGXBR	CONVERT TO LOGICAL (extended BFP to 64)	RRF-e C F	SP	Db Xi	Xx			B3AE	19-23
CLGXTR	CONVERT TO LOGICAL (extended DFP to 64)	RRF-e C F	SP	Dt Xi	Xx			B94A	20-29
CLHF	COMPARE LOGICAL HIGH (32)	RXY-a C HW	A				B ₂	E3CF	7-135
CLHHR	COMPARE LOGICAL HIGH (32)	RRE C HW						B9CF	7-135
CLHHSI	COMPARE LOGICAL IMMEDIATE (16)	SIL C GE	A				B ₁	E555	7-130
CLHLR	COMPARE LOGICAL HIGH (32)	RRE C HW						B9DF	7-135
CLHRL	COMPARE LOGICAL RELATIVE LONG (32←16)	RIL-b C GE	A					C67	7-131
CLI	COMPARE LOGICAL (immediate)	SI C	A				B ₁	95	7-130
CLIB	COMPARE LOGICAL IMMEDIATE AND BRANCH (32←8)	RIS GE				B		ECFF	7-132
CLIH	COMPARE LOGICAL IMMEDIATE HIGH (32)	RIL-a C HW						CCF	7-135
CLIJ	COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (32←8)	RIE-c GE				B		EC7F	7-132
CLIJ	COMPARE LOGICAL (immediate)	SIY C LD	A				B ₁	EB55	7-130
CLM	COMPARE LOGICAL CHAR. UNDER MASK (low)	RS-b C	A				B ₂	BD	7-134
CLMH	COMPARE LOGICAL CHAR. UNDER MASK (high)	RSY-b C N	A				B ₂	EB20	7-134
CLMY	COMPARE LOGICAL CHAR. UNDER MASK (low)	RSY-b C LD	A				B ₂	EB21	7-134
CLR	COMPARE LOGICAL (32)	RR C						15	7-130
CLRB	COMPARE LOGICAL AND BRANCH (32)	RRS GE				B		ECF7	7-132
CLRJ	COMPARE LOGICAL AND BRANCH RELATIVE (32)	RIE-b GE				B		EC77	7-132
CLRL	COMPARE LOGICAL RELATIVE LONG (32)	RIL-b C GE	A SP					C6F	7-130
CLRT	COMPARE LOGICAL AND TRAP (32)	RRF-c GE			Dc			B973	7-133
CLST	COMPARE LOGICAL STRING	RRE C	A SP	IC	G0		R ₁ R ₂	B25D	7-143
CLY	COMPARE LOGICAL (32)	RXY-a C LD	A				B ₂	E355	7-130
CMPSC	COMPRESSION CALL	RRE C	A SP	II Dd	GM	ST	R ₁ R ₂	B263	7-147
CP	COMPARE DECIMAL	SS-b C	A	Dd			B ₁ B ₂	F9	8-6
CPSDR	COPY SIGN (long)	RRF-b FS			Da			B372	9-29
CPYA	COPY ACCESS	RRE					U ₁ U ₂	B24D	7-203
CR	COMPARE (32)	RR C						19	7-112
CRB	COMPARE AND BRANCH (32)	RRS GE				B		ECF6	7-113
CRJ	COMPARE AND BRANCH RELATIVE (32)	RIE-b GE				B		EC76	7-113
CRL	COMPARE RELATIVE LONG (32)	RIL-b C GE	A SP					C6D	7-112
CRT	COMPARE AND TRAP (32)	RRF-c GE			Dc			B972	7-127
CS	COMPARE AND SWAP (32)	RS-a C	A SP		\$	ST	B ₂	BA	7-122
CSCH	CLEAR SUBCHANNEL	S C	P	OP	¢ GS			B230	14-5
CSDTR	CONVERT TO SIGNED PACKED (long DFP to 64)	RRF-d TF		Dt				B3E3	20-31
CSG	COMPARE AND SWAP (64)	RSY-a C N	A SP		\$	ST	B ₂	EB30	7-122
CSP	COMPARE AND SWAP AND PURGE	RRE C	P A ¹ SP		\$	ST	R ₂	B250	10-17
CSPG	COMPARE AND SWAP AND PURGE	RRE C DE	P A ¹ SP		\$	ST	R ₂	B98A	10-17
CSST	COMPARE AND SWAP AND STORE	SSF C CS	A SP		\$ GM	ST	B ₁ B ₂	C82	7-124
CSXTR	CONVERT TO SIGNED PACKED (extended DFP to 128)	RRF-d TF	SP	Dt				B3EB	20-31
CSY	COMPARE AND SWAP (32)	RSY-a C LD	A SP		\$	ST	B ₂	EB14	7-122
CU12	CONVERT UTF-8 TO UTF-16	RRF-c C	A SP	IC		ST	R ₁ R ₂	B2A7	7-195

Figure B-2. Instructions Arranged by Mnemonic (Part 5 of 18)

Mnemonic	Name	Characteristics										Op-code	Page
CU14	CONVERT UTF-8 TO UTF-32	RRF-c	C	E3	A	SP	IC		ST	R ₁	R ₂	B9B0	7-199
CU21	CONVERT UTF-16 TO UTF-8	RRF-c	C		A	SP	IC		ST	R ₁	R ₂	B2A6	7-186
CU24	CONVERT UTF-16 TO UTF-32	RRF-c	C	E3	A	SP	IC		ST	R ₁	R ₂	B9B1	7-183
CU41	CONVERT UTF-32 TO UTF-8	RRE	C	E3	A	SP	IC		ST	R ₁	R ₂	B9B2	7-192
CU42	CONVERT UTF-32 TO UTF-16	RRE	C	E3	A	SP	IC		ST	R ₁	R ₂	B9B3	7-189
CUDTR	CONVERT TO UNSIGNED PACKED (long DFP to 64)	RRE		TF			Dt					B3E2	20-32
CUSE	COMPARE UNTIL SUBSTRING EQUAL	RRE	C		A	SP	II	GM		R ₁	R ₂	B257	7-144
CUTFU	CONVERT UTF-8 TO UNICODE	RRF-c	C		A	SP	IC		ST	R ₁	R ₂	B2A7	7-195
CUUTF	CONVERT UNICODE TO UTF-8	RRF-c	C		A	SP	IC		ST	R ₁	R ₂	B2A6	7-186
CUXTR	CONVERT TO UNSIGNED PACKED (extended DFP to 128)	RRE		TF		SP	Dt					B3EA	20-32
CVB	CONVERT TO BINARY (32)	RX-a			A		Dd	IK			B ₂	4F	7-181
CVBG	CONVERT TO BINARY (64)	RXY-a		N	A		Dd	IK			B ₂	E30E	7-181
CVBY	CONVERT TO BINARY (32)	RXY-a		LD	A		Dd	IK			B ₂	E306	7-181
CVD	CONVERT TO DECIMAL (32)	RX-a			A				ST		B ₂	4E	7-182
CVDG	CONVERT TO DECIMAL (64)	RXY-a		N	A				ST		B ₂	E32E	7-182
CVDY	CONVERT TO DECIMAL (32)	RXY-a		LD	A				ST		B ₂	E326	7-182
CXBR	COMPARE (extended BFP)	RRE	C			SP	Db	Xi				B349	19-16
CXFBR	CONVERT FROM FIXED (32 to extended BFP)	RRE				SP	Db					B396	19-18
CXFBRA	CONVERT FROM FIXED (32 to extended BFP)	RRF-e		F		SP	Db					B396	19-18
CXFR	CONVERT FROM FIXED (32 to extended HFP)	RRE				SP	Da					B3B6	18-10
CXFTR	CONVERT FROM FIXED (32 to extended DFP)	RRE		F		SP	Dt					B959	20-24
CXGBR	CONVERT FROM FIXED (64 to extended BFP)	RRE		N		SP	Db					B3A6	19-18
CXGBRA	CONVERT FROM FIXED (64 to extended BFP)	RRF-e		F		SP	Db					B3A6	19-18
CXGR	CONVERT FROM FIXED (64 to extended HFP)	RRE		N		SP	Da					B3C6	18-10
CXGTR	CONVERT FROM FIXED (64 to extended DFP)	RRE		TF		SP	Dt					B3F9	20-24
CXGTRA	CONVERT FROM FIXED (64 to extended DFP)	RRF-e		F		SP	Dt					B3F9	20-24
CXLFB	CONVERT FROM LOGICAL (32 to extended BFP)	RRF-e		F		SP	Db					B392	19-20
CXLFBTR	CONVERT FROM LOGICAL (32 to extended DFP)	RRF-e		F		SP	Dt					B95B	20-25
CXLGBR	CONVERT FROM LOGICAL (64 to extended BFP)	RRF-e		F		SP	Db					B3A2	19-20
CXLGTR	CONVERT FROM LOGICAL (64 to extended DFP)	RRF-e		F		SP	Dt					B95A	20-25
CXR	COMPARE (extended HFP)	RRE	C			SP	Da					B369	18-10
CXSTR	CONVERT FROM SIGNED PACKED (128 to extended DFP)	RRE		TF		SP	Dt	Dd				B3FB	20-26
CXTR	COMPARE (extended DFP)	RRE	C	TF		SP	Dt	Xi				B3EC	20-21
CXUTR	CONVERT FROM UNSIGNED PACKED (128 to ext. DFP)	RRE		TF		SP	Dt	Dd				B3FA	20-27
CY	COMPARE (32)	RXY-a	C	LD	A						B ₂	E359	7-112
D	DIVIDE (32←64)	RX-a			A	SP		IK			B ₂	5D	7-203
DD	DIVIDE (long HFP)	RX-a			A		Da	EU EO FK			B ₂	6D	18-12
DDB	DIVIDE (long BFP)	RXE			A		Db	Xi Xz Xo Xu Xx			B ₂	ED1D	19-26
DDBR	DIVIDE (long BFP)	RRE					Db	Xi Xz Xo Xu Xx				B31D	19-26
DDR	DIVIDE (long HFP)	RR					Da	EU EO FK				2D	18-11
DDTR	DIVIDE (long DFP)	RRF-a		TF			Dt	Xi Xz Xo Xu Xx				B3D1	20-32
DDTRA	DIVIDE (long DFP)	RRF-a		F			Dt	Xi Xz Xo Xu Xx Xq				B3D1	20-32
DE	DIVIDE (short HFP)	RX-a			A		Da	EU EO FK			B ₂	7D	18-12
DEB	DIVIDE (short BFP)	RXE			A		Db	Xi Xz Xo Xu Xx			B ₂	ED0D	19-26
DEBR	DIVIDE (short BFP)	RRE					Db	Xi Xz Xo Xu Xx				B30D	19-26
DER	DIVIDE (short HFP)	RR					Da	EU EO FK				3D	18-11
DIDBR	DIVIDE TO INTEGER (long BFP)	RRF-b	C			SP	Db	Xi Xu Xx				B35B	19-27
DIEBR	DIVIDE TO INTEGER (short BFP)	RRF-b	C			SP	Db	Xi Xu Xx				B353	19-27
DL	DIVIDE LOGICAL (32←64)	RXY-a		N3	A	SP		IK			B ₂	E397	7-204
DLG	DIVIDE LOGICAL (64←128)	RXY-a		N	A	SP		IK			B ₂	E387	7-204
DLGR	DIVIDE LOGICAL (64←128)	RRE		N		SP		IK				B987	7-204
DLR	DIVIDE LOGICAL (32←64)	RRE		N3		SP		IK				B997	7-204

Figure B-2. Instructions Arranged by Mnemonic (Part 6 of 18)

Mne- monic	Name	Characteristics										Op- code	Page
DP	DIVIDE DECIMAL	SS-b		A	SP	Dd	DK		ST	B ₁	B ₂	FD	8-7
DR	DIVIDE (32←64)	RR			SP		IK					1D	7-203
DSG	DIVIDE SINGLE (64)	RXY-a	N	A	SP		IK				B ₂	E30D	7-205
DSGF	DIVIDE SINGLE (64←32)	RXY-a	N	A	SP		IK				B ₂	E31D	7-205
DSGFR	DIVIDE SINGLE (64←32)	RRE	N		SP		IK					B91D	7-204
DSGR	DIVIDE SINGLE (64)	RRE	N		SP		IK					B90D	7-204
DXBR	DIVIDE (extended BFP)	RRE			SP	Db	Xi Xz Xo Xu Xx					B34D	19-26
DXR	DIVIDE (extended HFP)	RRE			SP	Da	EU EO FK					B22D	18-12
DXTR	DIVIDE (extended DFP)	RRF-a	TF		SP	Dt	Xi Xz Xo Xu Xx					B3D9	20-32
DXTRA	DIVIDE (extended DFP)	RRF-a	F		SP	Dt	Xi Xz Xo Xu Xx Xq					B3D9	20-32
EAR	EXTRACT ACCESS	RRE									U ₂	B24F	7-208
ECAG	EXTRACT CACHE ATTRIBUTE	RSY-a	GE									EB4C	7-208
ECTG	EXTRACT CPU TIME	SSF	ET	A			GM		R ₃	B ₁	B ₂	C81	7-210
ED	EDIT	SS-a	C	A		Dd			ST	B ₁	B ₂	DE	8-7
EDMK	EDIT AND MARK	SS-a	C	A		Dd	G1		ST	B ₁	B ₂	DF	8-10
EEDTR	EXTRACT BIASED EXPONENT (long DFP to 64)	RRE	TF			Dt						B3E5	20-34
EEXTR	EXTRACT BIASED EXPONENT (extended DFP to 64)	RRE	TF		SP	Dt						B3ED	20-34
EFPC	EXTRACT FPC	RRE				Db						B38C	9-30
EPAIR	EXTRACT PRIMARY ASN AND INSTANCE	RRE	RA	Q		SO						B99A	10-20
EPAR	EXTRACT PRIMARY ASN	RRE		Q		SO						B226	10-20
EPSW	EXTRACT PSW	RRE	N3									B98D	7-210
EREG	EXTRACT STACKED REGISTERS (32)	RRE		A ¹		SE				U ₁	U ₂	B249	10-21
EREGG	EXTRACT STACKED REGISTERS (64)	RRE	N	A ¹		SE				U ₁	U ₂	B90E	10-21
ESAIR	EXTRACT SECONDARY ASN AND INSTANCE	RRE	RA	Q		SO						B99B	10-21
ESAR	EXTRACT SECONDARY ASN	RRE		Q		SO						B227	10-21
ESDTR	EXTRACT SIGNIFICANCE (long DFP)	RRE	TF			Dt						B3E7	20-35
ESEA	EXTRACT AND SET EXTENDED AUTHORITY	RRE	N	P								B99D	10-20
ESTA	EXTRACT STACKED STATE	RRE	C	A ¹	SP	SE						B24A	10-23
ESXTR	EXTRACT SIGNIFICANCE (extended DFP)	RRE	TF		SP	Dt						B3EF	20-35
EX	EXECUTE	RX-a		AI	SP		EX					44	7-207
EXRL	EXECUTE RELATIVE LONG	RIL-b	XX	AI			EX					C60	7-207
FIDBR	LOAD FP INTEGER (long BFP)	RRF-e			SP	Db	Xi Xx					B35F	19-31
FIDBRA	LOAD FP INTEGER (long BFP)	RRF-e	F		SP	Db	Xi Xx					B35F	19-31
FIDR	LOAD FP INTEGER (long HFP)	RRE				Da						B37F	18-14
FIDTR	LOAD FP INTEGER (long DFP)	RRF-e	TF			Dt	Xi Xx Xq					B3D7	20-37
FIEBR	LOAD FP INTEGER (short BFP)	RRF-e			SP	Db	Xi Xx					B357	19-31
FIEBRA	LOAD FP INTEGER (short BFP)	RRF-e	F		SP	Db	Xi Xx					B357	19-31
FIER	LOAD FP INTEGER (short HFP)	RRE				Da						B377	18-14
FIXBR	LOAD FP INTEGER (extended BFP)	RRF-e			SP	Db	Xi Xx					B347	19-31
FIXBRA	LOAD FP INTEGER (extended BFP)	RRF-e	F		SP	Db	Xi Xx					B347	19-31
FIXR	LOAD FP INTEGER (extended HFP)	RRE			SP	Da						B367	18-14
FIXTR	LOAD FP INTEGER (extended DFP)	RRF-e	TF		SP	Dt	Xi Xx Xq					B3DF	20-37
FLOGR	FIND LEFTMOST ONE	RRE	C	EI	SP							B983	7-211
HDR	HALVE (long HFP)	RR				Da	EU					24	18-13
HER	HALVE (short HFP)	RR				Da	EU					34	18-13
HSCH	HALT SUBCHANNEL	S	C	P		OP	¢ GS					B231	14-6
IAC	INSERT ADDRESS SPACE CONTROL	RRE	C	Q		SO						B224	10-25
IC	INSERT CHARACTER	RX-a		A							B ₂	43	7-211
ICM	INSERT CHARACTERS UNDER MASK (low)	RS-b	C	A							B ₂	BF	7-212
ICMH	INSERT CHARACTERS UNDER MASK (high)	RSY-b	C	N	A						B ₂	EB80	7-212
ICMY	INSERT CHARACTERS UNDER MASK (low)	RSY-b	C	LD	A						B ₂	EB81	7-212
ICY	INSERT CHARACTER	RXY-a	LD	A							B ₂	E373	7-211
IDTE	INVALIDATE DAT TABLE ENTRY	RRF-b	DE	P A ¹	SP		\$					B98E	10-28
IEDTR	INSERT BIASED EXPONENT (64 to long DFP)	RRF-b	TF			Dt						B3F6	20-35
IEXTR	INSERT BIASED EXPONENT (64 to extended DFP)	RRF-b	TF		SP	Dt						B3FE	20-35

Figure B-2. Instructions Arranged by Mnemonic (Part 7 of 18)

Mnemonic	Name	Characteristics						Op-code	Page
IIHF	INSERT IMMEDIATE (high)	RIL-a	EI					C08	7-212
IIHH	INSERT IMMEDIATE (high high)	RI-a	N					A50	7-212
IIHL	INSERT IMMEDIATE (high low)	RI-a	N					A51	7-213
IILF	INSERT IMMEDIATE (low)	RIL-a	EI					C09	7-213
IILH	INSERT IMMEDIATE (low high)	RI-a	N					A52	7-213
IILL	INSERT IMMEDIATE (low low)	RI-a	N					A53	7-213
IPK	INSERT PSW KEY	S		Q		G2		B20B	10-26
IPM	INSERT PROGRAM MASK	RRE						B222	7-213
IPTE	INVALIDATE PAGE TABLE ENTRY	RRF-a		P A ¹ SP		\$		B221	10-32
ISKE	INSERT STORAGE KEY EXTENDED	RRE		P A ¹				B229	10-26
IVSK	INSERT VIRTUAL STORAGE KEY	RRE		Q A ¹	SO		R ₂	B223	10-27
KDB	COMPARE AND SIGNAL (long BFP)	RXE C		A	Db Xi		B ₂	ED18	19-17
KDBR	COMPARE AND SIGNAL (long BFP)	RRE C			Db Xi			B318	19-17
KDTR	COMPARE AND SIGNAL (long DFP)	RRE C TF			Dt Xi			B3E0	20-22
KEB	COMPARE AND SIGNAL (short BFP)	RXE C		A	Db Xi		B ₂	ED08	19-17
KEBR	COMPARE AND SIGNAL (short BFP)	RRE C			Db Xi			B308	19-17
KIMD	COMPUTE INTERMEDIATE MESSAGE DIGEST	RRE C MS		A SP	IC GM I1	ST	R ₂	B93E	7-159
KLMD	COMPUTE LAST MESSAGE DIGEST	RRE C MS		A SP	IC GM I1	ST	R ₂	B93F	7-159
KM	CIPHER MESSAGE	RRE C MS		A SP	IC GM I1	ST	R ₁ R ₂	B92E	7-46
KMAC	COMPUTE MESSAGE AUTHENTICATION CODE	RRE C MS		A SP	IC GM I1	ST	R ₂	B91E	7-171
KMC	CIPHER MESSAGE WITH CHAINING	RRE C MS		A SP	IC GM I1	ST	R ₁ R ₂	B92F	7-46
KMCTR	CIPHER MESSAGE WITH COUNTER	RRF-b C M4		A SP	IC GM I1	ST	R ₁ , R ₂ , R ₃	B92D	7-86
KMF	CIPHER MESSAGE WITH CFB	RRE C M4		A SP	IC GM I1	ST	R ₁ R ₂	B92A	7-71
KMO	CIPHER MESSAGE WITH OFB	RRE C M4		A SP	IC GM I1	ST	R ₁ R ₂	B92B	7-98
KXBR	COMPARE AND SIGNAL (extended BFP)	RRE C		SP	Db Xi			B348	19-17
KXTR	COMPARE AND SIGNAL (extended DFP)	RRE C TF		SP	Dt Xi			B3E8	20-22
L	LOAD (32)	RX-a		A			B ₂	58	7-213
LA	LOAD ADDRESS	RX-a						41	7-215
LAA	LOAD AND ADD (32)	RSY-a C IA		A SP	IF	ST	B ₂	EBF8	7-217
LAAG	LOAD AND ADD (64)	RSY-a C IA		A SP	IF	ST	B ₂	EBE8	7-217
LAAL	LOAD AND ADD LOGICAL (32)	RSY-a C IA		A SP		ST	B ₂	EBFA	7-217
LAALG	LOAD AND ADD LOGICAL (64)	RSY-a C IA		A SP		ST	B ₂	EBEA	7-217
LAE	LOAD ADDRESS EXTENDED	RX-a					U ₁ BP	51	7-215
LAEY	LOAD ADDRESS EXTENDED	RXY-a GE					U ₁ BP	E375	7-215
LAM	LOAD ACCESS MULTIPLE	RS-a		A SP			UB	9A	7-214
LAMY	LOAD ACCESS MULTIPLE	RSY-a LD		A SP			UB	EB9A	7-214
LAN	LOAD AND AND (32)	RSY-a C IA		A SP		ST	B ₂	EBF4	7-218
LANG	LOAD AND AND (64)	RSY-a C IA		A SP		ST	B ₂	EBE4	7-218
LAO	LOAD AND OR (32)	RSY-a C IA		A SP		ST	B ₂	EBF6	7-219
LAOG	LOAD AND OR (64)	RSY-a C IA		A SP		ST	B ₂	EBE6	7-219
LARL	LOAD ADDRESS RELATIVE LONG	RIL-b N3						C00	7-216
LASP	LOAD ADDRESS SPACE PARAMETERS	SSE C		P A ¹ SP	SO		B ₁	E500	10-34
LAX	LOAD AND EXCLUSIVE OR (32)	RSY-a C IA		A SP		ST	B ₂	EBF7	7-218
LAXG	LOAD AND EXCLUSIVE OR (64)	RSY-a C IA		A SP		ST	B ₂	EBE7	7-218
LAY	LOAD ADDRESS	RXY-a LD						E371	7-215
LB	LOAD BYTE (32)	RXY-a LD		A				E376	7-220
LBH	LOAD BYTE HIGH (32←8)	RXY-a HW		A			B ₂	E3C0	7-221
LBR	LOAD BYTE (32)	RRE EI						B926	7-220
LCDBR	LOAD COMPLEMENT (long BFP)	RRE C			Db			B313	19-30
LCDFR	LOAD COMPLEMENT (long)	RRE FS			Da			B373	9-30
LCDR	LOAD COMPLEMENT (long HFP)	RR C			Da			23	18-14
LCEBR	LOAD COMPLEMENT (short BFP)	RRE C			Db			B303	19-30
LCER	LOAD COMPLEMENT (short HFP)	RR C			Da			33	18-14
LCGFR	LOAD COMPLEMENT (64←32)	RRE C N						B913	7-221
LCGR	LOAD COMPLEMENT (64)	RRE C N			IF			B903	7-221

Figure B-2. Instructions Arranged by Mnemonic (Part 8 of 18)

Mne- monic	Name	Characteristics						Op- code	Page
LCT	LOAD COMPLEMENT (32)	RR C		IF				B ₂ 13	7-221
LCTL	LOAD CONTROL (32)	RS-a	P A SP					B ₂ B7	10-44
LCTLG	LOAD CONTROL (64)	RSY-a N	P A SP					B ₂ EB2F	10-44
LCXBR	LOAD COMPLEMENT (extended BFP)	RRE C	SP	Db				B343	19-30
LCXR	LOAD COMPLEMENT (extended HFP)	RRE C	SP	Da				B363	18-14
LD	LOAD (long)	RX-a	A	Da				B ₂ 68	9-30
LDE	LOAD LENGTHENED (short to long HFP)	RXE	A	Da				B ₂ ED24	18-15
LDEB	LOAD LENGTHENED (short to long BFP)	RXE	A	Db Xi				B ₂ ED04	19-33
LDEBR	LOAD LENGTHENED (short to long BFP)	RRE		Db Xi				B304	19-32
LDER	LOAD LENGTHENED (short to long HFP)	RRE		Da				B324	18-15
LDETR	LOAD LENGTHENED (short to long DFP)	RRF-d TF		Dt Xi				B3D4	20-40
LDGR	LOAD FPR FROM GR (64 to long)	RRE FG		Da				B3C1	9-33
LDR	LOAD (long)	RR		Da				28	9-30
LDXBR	LOAD ROUNDED (extended to long BFP)	RRE	SP	Db Xi Xo Xu Xx				B345	19-34
LDXBRA	LOAD ROUNDED (extended to long BFP)	RRF-e F	SP	Db Xi Xo Xu Xx				B345	19-34
LDXR	LOAD ROUNDED (extended to long HFP)	RR	SP	Da EO				25	18-16
LDXTR	LOAD ROUNDED (extended to long DFP)	RRF-e TF	SP	Dt Xi Xo Xu Xx Xq				B3DD	20-41
LDY	LOAD (long)	RXY-a LD	A	Da				B ₂ ED65	9-30
LE	LOAD (short)	RX-a	A	Da				B ₂ 78	9-30
LEDBR	LOAD ROUNDED (long to short BFP)	RRE		Db Xi Xo Xu Xx				B344	19-34
LEDBRA	LOAD ROUNDED (long to short BFP)	RRF-e F	SP	Db Xi Xo Xu Xx				B344	19-34
LEDR	LOAD ROUNDED (long to short HFP)	RR		Da EO				35	18-16
LEDTR	LOAD ROUNDED (long to short DFP)	RRF-e TF		Dt Xi Xo Xu Xx Xq				B3D5	20-41
LER	LOAD (short)	RR		Da				38	9-30
LEXBR	LOAD ROUNDED (extended to short BFP)	RRE	SP	Db Xi Xo Xu Xx				B346	19-34
LEXBRA	LOAD ROUNDED (extended to short BFP)	RRF-e F	SP	Db Xi Xo Xu Xx				B346	19-34
LEXR	LOAD ROUNDED (extended to short HFP)	RRE	SP	Da EO				B366	18-16
LEY	LOAD (short)	RXY-a LD	A	Da				B ₂ ED64	9-30
LFAS	LOAD FPC AND SIGNAL	S XF	A SP	Dt Xg				B ₂ B2BD	9-31
LFH	LOAD HIGH (32)	RXY-a HW	A					B ₂ E3CA	7-223
LFPC	LOAD FPC	S	A SP	Db				B ₂ B29D	9-31
LG	LOAD (64)	RXY-a N	A					B ₂ E304	7-213
LGB	LOAD BYTE (64)	RXY-a LD	A					E377	7-220
LGBR	LOAD BYTE (64)	RRE EI						B906	7-220
LGDR	LOAD GR FROM FPR (long to 64)	RRE FG		Da				B3CD	9-33
LGF	LOAD (64 ← 32)	RXY-a N	A					B ₂ E314	7-213
LGFI	LOAD IMMEDIATE (64 ← 32)	RIL-a EI						C01	7-214
LGFR	LOAD (64 ← 32)	RRE N						B914	7-213
LGFRLL	LOAD RELATIVE LONG (64 ← 32)	RIL-b GE	A SP					C4C	7-214
LGH	LOAD HALFWORD (64)	RXY-a N	A					B ₂ E315	7-222
LGHI	LOAD HALFWORD IMMEDIATE (64)	RI-a N						A79	7-222
LGHR	LOAD HALFWORD (64)	RRE EI						B907	7-221
LGHRL	LOAD HALFWORD RELATIVE LONG (64 ← 16)	RIL-b GE	A					C44	7-222
LGR	LOAD (64)	RRE N						B904	7-213
LGRLL	LOAD RELATIVE LONG (64)	RIL-b GE	A SP					C48	7-214
LH	LOAD HALFWORD (32)	RX-a	A					B ₂ 48	7-221
LHH	LOAD HALFWORD HIGH (32 ← 16)	RXY-a HW	A					B ₂ E3C4	7-222
LHI	LOAD HALFWORD IMMEDIATE (32)	RI-a						A78	7-222
LHR	LOAD HALFWORD (32)	RRE EI						B927	7-221
LHRL	LOAD HALFWORD RELATIVE LONG (32 ← 16)	RIL-b GE	A					C45	7-222
LHY	LOAD HALFWORD (32)	RXY-a LD	A					B ₂ E378	7-221
LLC	LOAD LOGICAL CHARACTER (32)	RXY-a EI	A					B ₂ E394	7-224
LLCH	LOAD LOGICAL CHARACTER HIGH (32 ← 8)	RXY-a HW	A					B ₂ E3C2	7-224
LLCR	LOAD LOGICAL CHARACTER (32)	RRE EI						B994	7-224
LLGC	LOAD LOGICAL CHARACTER (64)	RXY-a N	A					B ₂ E390	7-224

Figure B-2. Instructions Arranged by Mnemonic (Part 9 of 18)

Mnemonic	Name	Characteristics						Op-code	Page
LLGCR	LOAD LOGICAL CHARACTER (64)	RRE	EI					B984	7-223
LLGF	LOAD LOGICAL (64←32)	RXY-a	N	A			B ₂	E316	7-223
LLGFR	LOAD LOGICAL (64←32)	RRE	N					B916	7-223
LLGFRL	LOAD LOGICAL RELATIVE LONG (64←32)	RIL-b	GE	A	SP			C4E	7-223
LLGH	LOAD LOGICAL HALFWORD (64)	RXY-a	N	A			B ₂	E391	7-224
LLGHR	LOAD LOGICAL HALFWORD (64)	RRE	EI					B985	7-224
LLGHRL	LOAD LOGICAL HALFWORD RELATIVE LONG (64←16)	RIL-b	GE	A				C46	7-224
LLGT	LOAD LOGICAL THIRTY ONE BITS	RXY-a	N	A			B ₂	E317	7-226
LLGTR	LOAD LOGICAL THIRTY ONE BITS	RRE	N					B917	7-226
LLH	LOAD LOGICAL HALFWORD (32)	RXY-a	EI	A			B ₂	E395	7-224
LLHH	LOAD LOGICAL HALFWORD HIGH (32←16)	RXY-a	HW	A			B ₂	E3C6	7-225
LLHR	LOAD LOGICAL HALFWORD (32)	RRE	EI					B995	7-224
LLHRL	LOAD LOGICAL HALFWORD RELATIVE LONG (32←16)	RIL-b	GE	A				C42	7-224
LLIHF	LOAD LOGICAL IMMEDIATE (high)	RIL-a	EI					C0E	7-225
LLIHH	LOAD LOGICAL IMMEDIATE (high high)	RI-a	N					A5C	7-225
LLIHL	LOAD LOGICAL IMMEDIATE (high low)	RI-a	N					A5D	7-225
LLILF	LOAD LOGICAL IMMEDIATE (low)	RIL-a	EI					C0F	7-225
LLILH	LOAD LOGICAL IMMEDIATE (low high)	RI-a	N					A5E	7-225
LLILL	LOAD LOGICAL IMMEDIATE (low low)	RI-a	N					A5F	7-225
LM	LOAD MULTIPLE (32)	RS-a		A			B ₂	98	7-226
LMD	LOAD MULTIPLE DISJOINT	SS-e	N	A			B ₂ B ₄	EF	7-227
LMG	LOAD MULTIPLE (64)	RSY-a	N	A			B ₂	EB04	7-226
LMH	LOAD MULTIPLE HIGH	RSY-a	N	A			B ₂	EB96	7-227
LMY	LOAD MULTIPLE (32)	RSY-a	LD	A			B ₂	EB98	7-226
LNDBR	LOAD NEGATIVE (long BFP)	RRE	C			Db		B311	19-33
LNDFR	LOAD NEGATIVE (long)	RRE	FS			Da		B371	9-33
LNDR	LOAD NEGATIVE (long HFP)	RR	C			Da		21	18-15
LNEBR	LOAD NEGATIVE (short BFP)	RRE	C			Db		B301	19-33
LNER	LOAD NEGATIVE (short HFP)	RR	C			Da		31	18-15
LNGFR	LOAD NEGATIVE (64←32)	RRE	C	N				B911	7-227
LNGR	LOAD NEGATIVE (64)	RRE	C	N				B901	7-227
LNR	LOAD NEGATIVE (32)	RR	C					11	7-227
LNxBR	LOAD NEGATIVE (extended BFP)	RRE	C		SP	Db		B341	19-33
LNxR	LOAD NEGATIVE (extended HFP)	RRE	C		SP	Da		B361	18-15
LOC	LOAD ON CONDITION (32)	RSY-b	CL	A			B ₂	EBF2	7-228
LOCg	LOAD ON CONDITION (64)	RSY-b	CL	A			B ₂	EBE2	7-228
LOCGR	LOAD ON CONDITION (64)	RRF-c	CL	A				B9E2	7-228
LOCR	LOAD ON CONDITION (32)	RRF-c	CL	A				B9F2	7-228
LPD	LOAD PAIR DISJOINT (32)	SSF	C	IA	A	SP	B ₁ B ₂	C84	7-229
LPDBR	LOAD POSITIVE (long BFP)	RRE	C			Db		B310	19-33
LPDFR	LOAD POSITIVE (long)	RRE	FS			Da		B370	9-34
LPDG	LOAD PAIR DISJOINT (64)	SSF	C	IA	A	SP	B ₁ B ₂	C85	7-229
LPDR	LOAD POSITIVE (long HFP)	RR	C			Da		20	18-16
LPEBR	LOAD POSITIVE (short BFP)	RRE	C			Db		B300	19-33
LPER	LOAD POSITIVE (short HFP)	RR	C			Da		30	18-16
LPGFR	LOAD POSITIVE (64←32)	RRE	C	N				B910	7-230
LPGR	LOAD POSITIVE (64)	RRE	C	N		IF		B900	7-230
LPQ	LOAD PAIR FROM QUADWORD	RXY-a	N	A	SP		B ₂	E38F	7-230
LPR	LOAD POSITIVE (32)	RR	C			IF		10	7-230
LPSW	LOAD PSW	S	L	P	A	SP	B ₂	82	10-47
LPSWE	LOAD PSW EXTENDED	S	L	N	P	A	B ₂	B2B2	10-48
LPTEA	LOAD PAGE TABLE ENTRY ADDRESS	RRF-b	C	D2	P	A ¹ SP	R ₂	B9AA	10-44
LPXBR	LOAD POSITIVE (extended BFP)	RRE	C			Db		B340	19-33

Figure B-2. Instructions Arranged by Mnemonic (Part 10 of 18)

Mne- monic	Name	Characteristics						Op- code	Page
LPXR	LOAD POSITIVE (extended HFP)	RRE C		SP	Da			B360	18-16
LR	LOAD (32)	RR						18	7-213
LRA	LOAD REAL ADDRESS (32)	RX-a C	P A ¹		SO		BP	B1	10-49
LRAG	LOAD REAL ADDRESS (64)	RXY-a C N	P A ¹				BP	E303	10-49
LRAY	LOAD REAL ADDRESS (32)	RXY-a C LD	P A ¹		SO		BP	E313	10-49
LRDR	LOAD ROUNDED (extended to long HFP)	RR		SP	Da EO			25	18-16
LRER	LOAD ROUNDED (long to short HFP)	RR			Da EO			35	18-16
LRL	LOAD RELATIVE LONG (32)	RIL-b GE	A SP					C4D	7-214
LRV	LOAD REVERSED (32)	RXY-a N3	A				B ₂	E31E	7-231
LRVG	LOAD REVERSED (64)	RXY-a N	A				B ₂	E30F	7-231
LRVGR	LOAD REVERSED (64)	RRE N						B90F	7-231
LRVH	LOAD REVERSED (16)	RXY-a N3	A				B ₂	E31F	7-231
LRVR	LOAD REVERSED (32)	RRE N3						B91F	7-230
LT	LOAD AND TEST (32)	RXY-a C EI	A				B ₂	E312	7-220
LTDBR	LOAD AND TEST (long BFP)	RRE C			Db Xi			B312	19-30
LTDR	LOAD AND TEST (long HFP)	RR C			Da			22	18-13
LTDR	LOAD AND TEST (long DFP)	RRE C TF			Dt Xi			B3D6	20-36
LTEBR	LOAD AND TEST (short BFP)	RRE C			Db Xi			B302	19-30
LTER	LOAD AND TEST (short HFP)	RR C			Da			32	18-13
LTG	LOAD AND TEST (64)	RXY-a C EI	A				B ₂	E302	7-220
LTGF	LOAD AND TEST (64 ← 32)	RXY-a C GE	A				B ₂	E332	7-220
LTGFR	LOAD AND TEST (64 ← 32)	RRE C N						B912	7-220
LTGR	LOAD AND TEST (64)	RRE C N						B902	7-219
LTR	LOAD AND TEST (32)	RR C						12	7-219
LTXBR	LOAD AND TEST (extended BFP)	RRE C		SP	Db Xi			B342	19-30
LTXR	LOAD AND TEST (extended HFP)	RRE C		SP	Da			B362	18-13
LTXTR	LOAD AND TEST (extended DFP)	RRE C TF		SP	Dt Xi			B3DE	20-36
LURA	LOAD USING REAL ADDRESS (32)	RRE	P A ¹	SP				B24B	10-53
LURAG	LOAD USING REAL ADDRESS (64)	RRE N	P A ¹	SP				B905	10-53
LXD	LOAD LENGTHENED (long to extended HFP)	RXE	A	SP	Da		B ₂	ED25	18-15
LXDB	LOAD LENGTHENED (long to extended BFP)	RXE	A	SP	Db Xi		B ₂	ED05	19-33
LXDBR	LOAD LENGTHENED (long to extended BFP)	RRE		SP	Db Xi			B305	19-32
LXDR	LOAD LENGTHENED (long to extended HFP)	RRE		SP	Da			B325	18-15
LXDTR	LOAD LENGTHENED (long to extended DFP)	RRF-d TF		SP	Dt Xi			B3DC	20-40
LXE	LOAD LENGTHENED (short to extended HFP)	RXE	A	SP	Da		B ₂	ED26	18-15
LXEB	LOAD LENGTHENED (short to extended BFP)	RXE	A	SP	Db Xi		B ₂	ED06	19-33
LXEBr	LOAD LENGTHENED (short to extended BFP)	RRE		SP	Db Xi			B306	19-32
LXER	LOAD LENGTHENED (short to extended HFP)	RRE		SP	Da			B326	18-15
LXR	LOAD (extended)	RRE		SP	Da			B365	9-30
LY	LOAD (32)	RXY-a LD	A				B ₂	E358	7-213
LZDR	LOAD ZERO (long)	RRE			Da			B375	9-34
LZER	LOAD ZERO (short)	RRE			Da			B374	9-34
LZXR	LOAD ZERO (extended)	RRE		SP	Da			B376	9-34
M	MULTIPLY (64 ← 32)	RX-a	A	SP			B ₂	5C	7-247
MAD	MULTIPLY AND ADD (long HFP)	RXF HM	A		Da EU EO		B ₂	ED3E	18-19
MADB	MULTIPLY AND ADD (long BFP)	RXF	A		Db Xi Xo Xu Xx		B ₂	ED1E	19-37
MADBR	MULTIPLY AND ADD (long BFP)	RRD			Db Xi Xo Xu Xx			B31E	19-37
MADR	MULTIPLY AND ADD (long HFP)	RRD HM			Da EU EO			B33E	18-18
MAE	MULTIPLY AND ADD (short HFP)	RXF HM	A		Da EU EO		B ₂	ED2E	18-19
MAEB	MULTIPLY AND ADD (short BFP)	RXF	A		Db Xi Xo Xu Xx		B ₂	ED0E	19-37
MAEBR	MULTIPLY AND ADD (short BFP)	RRD			Db Xi Xo Xu Xx			B30E	19-37
MAER	MULTIPLY AND ADD (short HFP)	RRD HM			Da EU EO			B32E	18-18
MAY	MULTIPLY & ADD UNNORMALIZED (long to ext. HFP)	RXF UE	A		Da		B ₂	ED3A	18-20
MAYH	MULTIPLY AND ADD UNNRM. (long to ext. high HFP)	RXF UE	A		Da		B ₂	ED3C	18-20
MAYHR	MULTIPLY AND ADD UNNRM. (long to ext. high HFP)	RRD UE			Da			B33C	18-20

Figure B-2. Instructions Arranged by Mnemonic (Part 11 of 18)

Mnemonic	Name	Characteristics						Op-code	Page
MAYL	MULTIPLY AND ADD UNNRM. (long to ext. low HFP)	RXF	UE	A	Da		B ₂	ED38	18-20
MAYLR	MULTIPLY AND ADD UNNRM. (long to ext. low HFP)	RRD	UE		Da			B338	18-20
MAYR	MULTIPLY & ADD UNNORMALIZED (long to ext. HFP)	RRD	UE		Da			B33A	18-20
MC	MONITOR CALL	SI		SP	ME			AF	7-231
MD	MULTIPLY (long HFP)	RX-a		A	Da EU EO		B ₂	6C	18-17
MDB	MULTIPLY (long BFP)	RXE		A	Db Xi Xo Xu Xx		B ₂	ED1C	19-35
MDBR	MULTIPLY (long BFP)	RRE			Db Xi Xo Xu Xx			B31C	19-35
MDE	MULTIPLY (short to long HFP)	RX-a		A	Da EU EO		B ₂	7C	18-17
MDEB	MULTIPLY (short to long BFP)	RXE		A	Db Xi		B ₂	ED0C	19-35
MDEBR	MULTIPLY (short to long BFP)	RRE			Db Xi			B30C	19-35
MDER	MULTIPLY (short to long HFP)	RR			Da EU EO			3C	18-17
MDR	MULTIPLY (long HFP)	RR			Da EU EO			2C	18-17
MDTR	MULTIPLY (long DFP)	RRF-a	TF		Dt Xi Xo Xu Xx			B3D0	20-43
MDTRA	MULTIPLY (long DFP)	RRF-a	F		Dt Xi Xo Xu Xx Xq			B3D0	20-43
ME	MULTIPLY (short to long HFP)	RX-a		A	Da EU EO		B ₂	7C	18-17
MEE	MULTIPLY (short HFP)	RXE		A	Da EU EO		B ₂	ED37	18-17
MEEB	MULTIPLY (short BFP)	RXE		A	Db Xi Xo Xu Xx		B ₂	ED17	19-35
MEEBR	MULTIPLY (short BFP)	RRE			Db Xi Xo Xu Xx			B317	19-35
MEER	MULTIPLY (short HFP)	RRE			Da EU EO			B337	18-17
MER	MULTIPLY (short to long HFP)	RR			Da EU EO			3C	18-17
MFY	MULTIPLY (64←32)	RXY-a	GE	A SP			B ₂	E35C	7-247
MGHI	MULTIPLY HALFWORD IMMEDIATE (64)	RI-a	N					A7D	7-248
MH	MULTIPLY HALFWORD (32)	RX-a		A			B ₂	4C	7-248
MHI	MULTIPLY HALFWORD IMMEDIATE (32)	RI-a						A7C	7-248
MHY	MULTIPLY HALFWORD (32)	RXY-a	GE	A			B ₂	E37C	7-248
ML	MULTIPLY LOGICAL (64←32)	RXY-a	N3	A SP			B ₂	E396	7-249
MLG	MULTIPLY LOGICAL (128←64)	RXY-a	N	A SP			B ₂	E386	7-249
MLGR	MULTIPLY LOGICAL (128←64)	RRE	N	SP				B986	7-249
MLR	MULTIPLY LOGICAL (64←32)	RRE	N3	SP				B996	7-248
MP	MULTIPLY DECIMAL	SS-b		A SP	Dd	ST	B ₁ B ₂	FC	8-11
MR	MULTIPLY (64←32)	RR		SP				1C	7-247
MS	MULTIPLY SINGLE (32)	RX-a		A			B ₂	71	7-249
MSCH	MODIFY SUBCHANNEL	S	C	P A SP	OP ¢ GS		B ₂	B232	14-8
MSD	MULTIPLY AND SUBTRACT (long HFP)	RXF	HM	A	Da EU EO		B ₂	ED3F	18-19
MSDB	MULTIPLY AND SUBTRACT (long BFP)	RXF		A	Db Xi Xo Xu Xx		B ₂	ED1F	19-37
MSDBR	MULTIPLY AND SUBTRACT (long BFP)	RRD			Db Xi Xo Xu Xx			B31F	19-37
MSDR	MULTIPLY AND SUBTRACT (long HFP)	RRD	HM		Da EU EO			B33F	18-19
MSE	MULTIPLY AND SUBTRACT (short HFP)	RXF	HM	A	Da EU EO		B ₂	ED2F	18-19
MSEB	MULTIPLY AND SUBTRACT (short BFP)	RXF		A	Db Xi Xo Xu Xx		B ₂	ED0F	19-37
MSEBR	MULTIPLY AND SUBTRACT (short BFP)	RRD			Db Xi Xo Xu Xx			B30F	19-37
MSER	MULTIPLY AND SUBTRACT (short HFP)	RRD	HM		Da EU EO			B32F	18-19
MSFI	MULTIPLY SINGLE IMMEDIATE (32)	RIL-a	GE					C21	7-250
MSG	MULTIPLY SINGLE (64)	RXY-a	N	A			B ₂	E30C	7-249
MSGF	MULTIPLY SINGLE (64←32)	RXY-a	N	A			B ₂	E31C	7-249
MSGFI	MULTIPLY SINGLE IMMEDIATE (64←32)	RIL-a	GE					C20	7-250
MSGFR	MULTIPLY SINGLE (64←32)	RRE	N					B91C	7-249
MSGR	MULTIPLY SINGLE (64)	RRE	N					B90C	7-249
MSR	MULTIPLY SINGLE (32)	RRE						B252	7-249
MSTA	MODIFY STACKED STATE	RRE		A ¹ SP	SE	ST		B247	10-53
MSY	MULTIPLY SINGLE (32)	RXY-a	LD	A			B ₂	E351	7-249
MVC	MOVE (character)	SS-a		A		ST	B ₁ B ₂	D2	7-232
MVCDK	MOVE WITH DESTINATION KEY	SSe		Q A	GM	ST	B ₁ B ₂	E50F	10-58
MVCIN	MOVE INVERSE	SS-a		A		ST	B ₁ B ₂	E8	7-233
MVCK	MOVE WITH KEY	SS-d	C	Q A		ST	B ₁ B ₂	D9	10-59
MVCL	MOVE LONG	RR	C	A SP	II	ST	R ₁ R ₂	0E	7-234

Figure B-2. Instructions Arranged by Mnemonic (Part 12 of 18)

Mne- monic	Name	Characteristics								Op- code	Page
MVCLE	MOVE LONG EXTENDED	RS-a C	A SP	IC	ST	R ₁	R ₃	A8	7-238		
MVCLU	MOVE LONG UNICODE	RSY-a C E2	A SP	IC	ST	R ₁	R ₃	EB8E	7-241		
MVCOS	MOVE WITH OPTIONAL SPECIFICATIONS	SSF C MO	Q A	SO G0	ST	B†	B†	C80	10-60		
MVCP	MOVE TO PRIMARY	SS-d C	Q A	SO ¢	ST			DA	10-56		
MVCS	MOVE TO SECONDARY	SS-d C	Q A	SO ¢	ST			DB	10-56		
MVCSK	MOVE WITH SOURCE KEY	SSE	Q A	GM	ST	B ₁	B ₂	E50E	10-63		
MVGHI	MOVE (64←16)	SIL GE	A		ST	B ₁		E548	7-233		
MVHHI	MOVE (16←16)	SIL GE	A		ST	B ₁		E544	7-232		
MVHI	MOVE (32←16)	SIL GE	A		ST	B ₁		E54C	7-233		
MVI	MOVE (immediate)	SI	A		ST	B ₁		92	7-233		
MVIY	MOVE (immediate)	SIY LD	A		ST	B ₁		EB52	7-233		
MVN	MOVE NUMERICS	SS-a	A		ST	B ₁	B ₂	D1	7-244		
MVO	MOVE WITH OFFSET	SS-b	A		ST	B ₁	B ₂	F1	7-246		
MVPG	MOVE PAGE	RRE C	Q A SP	G0	ST	R ₁	R ₂	B254	10-54		
MVST	MOVE STRING	RRE C	A SP	IC G0	ST	R ₁	R ₂	B255	7-245		
MVZ	MOVE ZONES	SS-a	A		ST	B ₁	B ₂	D3	7-246		
MXBR	MULTIPLY (extended BFP)	RRE	SP	Db Xi Xo Xu Xx				B34C	19-35		
MXD	MULTIPLY (long to extended HFP)	RX-a	A SP	Da EU EO			B ₂	67	18-17		
MXDB	MULTIPLY (long to extended BFP)	RXE	A SP	Db Xi			B ₂	ED07	19-35		
MXDBR	MULTIPLY (long to extended BFP)	RRE	SP	Db Xi				B307	19-35		
MXDR	MULTIPLY (long to extended HFP)	RR	SP	Da EU EO				27	18-17		
MXR	MULTIPLY (extended HFP)	RR	SP	Da EU EO				26	18-17		
MXTR	MULTIPLY (extended DFP)	RRF-a TF	SP	Dt Xi Xo Xu Xx				B3D8	20-43		
MXTRA	MULTIPLY (extended DFP)	RRF-a F	SP	Dt Xi Xo Xu Xx Xq				B3D8	20-43		
MY	MULTIPLY UNNORMALIZED (long to ext. HFP)	RXF UE	A SP	Da			B ₂	ED3B	18-21		
MYH	MULTIPLY UNNORM. (long to ext. high HFP)	RXF UE	A	Da			B ₂	ED3D	18-21		
MYHR	MULTIPLY UNNORM. (long to ext. high HFP)	RRD UE		Da				B33D	18-21		
MYL	MULTIPLY UNNORM. (long to ext. low HFP)	RXF UE	A	Da			B ₂	ED39	18-21		
MYLR	MULTIPLY UNNORM. (long to ext. low HFP)	RRD UE		Da				B339	18-21		
MYR	MULTIPLY UNNORMALIZED (long to ext. HFP)	RRD UE	SP	Da				B33B	18-21		
N	AND (32)	RX-a C	A				B ₂	54	7-31		
NC	AND (character)	SS-a C	A		ST	B ₁	B ₂	D4	7-32		
NG	AND (64)	RXY-a C N	A				B ₂	E380	7-32		
NGR	AND (64)	RRE C N						B980	7-31		
NGRK	AND (64)	RRF-a C DO						B9E4	7-31		
NI	AND (immediate)	SI C	A		ST	B ₁		94	7-32		
NIHF	AND IMMEDIATE (high)	RIL-a C EI						C0A	7-32		
NIHH	AND IMMEDIATE (high high)	RI-a C N						A54	7-32		
NIHL	AND IMMEDIATE (high low)	RI-a C N						A55	7-33		
NILF	AND IMMEDIATE (low)	RIL-a C EI						C0B	7-33		
NILH	AND IMMEDIATE (low high)	RI-a C N						A56	7-33		
NILL	AND IMMEDIATE (low low)	RI-a C N						A57	7-33		
NIY	AND (immediate)	SIY C LD	A		ST	B ₁		EB54	7-32		
NR	AND (32)	RR C						14	7-31		
NRK	AND (32)	RRF-a C DO						B9F4	7-31		
NY	AND (32)	RXY-a C LD	A				B ₂	E354	7-32		
O	OR (32)	RX-a C	A				B ₂	56	7-250		
OC	OR (character)	SS-a C	A		ST	B ₁	B ₂	D6	7-251		
OG	OR (64)	RXY-a C N	A				B ₂	E381	7-250		
OGR	OR (64)	RRE C N						B981	7-250		
OGRK	OR (64)	RRF-a C DO						B9E6	7-250		
OI	OR (immediate)	SI C	A		ST	B ₁		96	7-251		
OIHF	OR IMMEDIATE (high)	RIL-a C EI						C0C	7-251		
OIHH	OR IMMEDIATE (high high)	RI-a C N						A58	7-251		
OIHL	OR IMMEDIATE (high low)	RI-a C N						A59	7-251		

Figure B-2. Instructions Arranged by Mnemonic (Part 13 of 18)

Mnemonic	Name	Characteristics						Op-code	Page
OILF	OR IMMEDIATE (low)	RIL-a C EI						C0D	7-251
OILH	OR IMMEDIATE (low high)	RI-a C N						A5A	7-252
OILL	OR IMMEDIATE (low low)	RI-a C N						A5B	7-252
OIY	OR (immediate)	SIY C LD	A			ST	B ₁	EB56	7-251
OR	OR (32)	RR C						16	7-250
ORK	OR (32)	RRF-a C DO						B9F6	7-250
OY	OR (32)	RXY-a C LD	A				B ₂	E356	7-250
PACK	PACK	SS-b	A			ST	B ₁ B ₂	F2	7-252
PALB	PURGE ALB	RRE	P		\$			B248	10-107
PC	PROGRAM CALL	S	Q A ¹	Z ¹ T	¢ GM	B ST		B218	10-81
PCC	PERFORM CRYPTOGRAPHIC COMPUTATION	RRE C M4	A SP	IC	GM I1	ST		B92C	7-255
PCKMO	PERFORM CRYPTOGRAPHIC KEY MGMT. OPERATIONS	RRE M3	P A SP		GM	ST		B928	10-66
PFD	PREFETCH DATA	RXY-b GE					B ₂	E336	7-285
PFDR	PREFETCH DATA RELATIVE LONG	RIL-c GE						C62	7-285
PFMF	PERFORM FRAME MANAGEMENT FUNCTION	RRE ED	P A ¹ SP	II	¢ ³			B9AF	10-70
PFPO	PERFORM FLOATING-POINT OPERATION	E PF	SP	Da Xi	Xo Xu Xx Xq			010A	9-34
PGIN	PAGE IN	RRE C ES	P A ¹		¢			B22E	10-64
PGOUT	PAGE OUT	RRE C ES	P A ¹		¢			B22F	10-65
PKA	PACK ASCII	SS-f E2	A SP			ST	B ₁ B ₂	E9	7-253
PKU	PACK UNICODE	SS-f E2	A SP			ST	B ₁ B ₂	E1	7-254
PLO	PERFORM LOCKED OPERATION	SS-e C	A SP		\$ GM	ST	FC	EE	7-270
POPCNT	POPULATION COUNT	RRE C PK						B9E1	7-284
PR	PROGRAM RETURN	E L	Q A ¹ SP	Z ⁴ T	¢ ²	B ST		0101	10-94
PT	PROGRAM TRANSFER	RRE	Q A ¹ SP	Z ² T	¢	B		B228	10-98
PTF	PERFORM TOPOLOGY FUNCTION	RRE C CT	P SP					B9A2	10-80
PTFF	PERFORM TIMING FACILITY FUNCTION	E C TS	Q A SP		GM	ST		0104	10-73
PTI	PROGRAM TRANSFER WITH INSTANCE	RRE RA	Q A ¹ SP	Z ⁶ T	¢	B		B99E	10-98
PTLB	PURGE TLB	S	P		\$			B20D	10-107
QADTR	QUANTIZE (long DFP)	RRF-b TF		Dt Xi	Xx Xq			B3F5	20-45
QAXTR	QUANTIZE (extended DFP)	RRF-b TF	SP	Dt Xi	Xx Xq			B3FD	20-45
RCHP	RESET CHANNEL PATH	S C	P					B23B	14-9
RISBG	ROTATE THEN INSERT SELECTED BITS	RIE-f C GE						EC55	7-288
RISBHG	ROTATE THEN INSERT SELECTED BITS HIGH	RIE-f HW						EC5D	7-290
RISBLG	ROTATE THEN INSERT SELECTED BITS LOW	RIE-f HW						EC51	7-290
RLL	ROTATE LEFT SINGLE LOGICAL (32)	RSY-a N3						EB1D	7-286
RLLG	ROTATE LEFT SINGLE LOGICAL (64)	RSY-a N						EB1C	7-286
RNSBG	ROTATE THEN AND SELECTED BITS	RIE-f C GE						EC54	7-287
ROSBG	ROTATE THEN OR SELECTED BITS	RIE-f C GE						EC56	7-287
RP	RESUME PROGRAM	S L	Q A SP	WE T		B	B ₂	B277	10-108
RRBE	RESET REFERENCE BIT EXTENDED	RRE C	P A ¹					B22A	10-107
RRBM	RESET REFERENCE BITS MULTIPLE	RRE RB	P A ¹					B9AE	10-108
RRDTR	REROUND (long DFP)	RRF-b TF		Dt Xi	Xx Xq			B3F7	20-47
RRXTR	REROUND (extended DFP)	RRF-b TF	SP	Dt Xi	Xx Xq			B3FF	20-47
RSCH	RESUME SUBCHANNEL	S C	P	OP	¢ GS			B238	14-10
RXSBG	ROTATE THEN EXCLUSIVE OR SELECTED BITS	RIE-f C GE						EC57	7-287
S	SUBTRACT (32)	RX-a C	A		IF		B ₂	5B	7-311
SAC	SET ADDRESS SPACE CONTROL	S	Q SP	SW	¢			B219	10-110
SACF	SET ADDRESS SPACE CONTROL FAST	S	Q SP	SW				B279	10-111
SAL	SET ADDRESS LIMIT	S	P	OP	¢ G1			B237	14-12
SAM24	SET ADDRESSING MODE (24)	E N3	SP	T				010C	7-295
SAM31	SET ADDRESSING MODE (31)	E N3	SP	T				010D	7-295
SAM64	SET ADDRESSING MODE (64)	E N		T				010E	7-295
SAR	SET ACCESS	RRE					U ₁	B24E	7-295
SCHM	SET CHANNEL MONITOR	S	P	OP	¢ GM			B23C	14-13

Figure B-2. Instructions Arranged by Mnemonic (Part 14 of 18)

Mne- monic	Name	Characteristics						Op- code	Page
SCK	SET CLOCK	S C	P A SP				B ₂	B204	10-112
SCKC	SET CLOCK COMPARATOR	S	P A SP				B ₂	B206	10-113
SCKPF	SET CLOCK PROGRAMMABLE FIELD	E	P A SP			G0		0107	10-113
SD	SUBTRACT NORMALIZED (long HFP)	RX-a C	A	Da EU EO LS			B ₂	6B	18-24
SDB	SUBTRACT (long BFP)	RXE C	A	Db Xi Xo Xu Xx			B ₂	ED1B	19-39
SDBR	SUBTRACT (long BFP)	RRE C		Db Xi Xo Xu Xx				B31B	19-39
SDR	SUBTRACT NORMALIZED (long HFP)	RR C		Da EU EO LS				2B	18-24
SDTR	SUBTRACT (long DFP)	RRF-a C TF		Dt Xi Xo Xu Xx				B3D3	20-51
SDTRA	SUBTRACT (long DFP)	RRF-a C F		Dt Xi Xo Xu Xx Xq				B3D3	20-51
SE	SUBTRACT NORMALIZED (short HFP)	RX-a C	A	Da EU EO LS			B ₂	7B	18-24
SEB	SUBTRACT (short BFP)	RXE C	A	Db Xi Xo Xu Xx			B ₂	ED0B	19-39
SEBR	SUBTRACT (short BFP)	RRE C		Db Xi Xo Xu Xx				B30B	19-39
SER	SUBTRACT NORMALIZED (short HFP)	RR C		Da EU EO LS				3B	18-24
SFASR	SET FPC AND SIGNAL	RRE XF	SP	Dt Xg				B385	9-46
SFPC	SET FPC	RRE	SP	Db				B384	9-46
SG	SUBTRACT (64)	RXY-a C N	A	IF			B ₂	E309	7-311
SGF	SUBTRACT (64←32)	RXY-a C N	A	IF			B ₂	E319	7-311
SGFR	SUBTRACT (64←32)	RRE C N		IF				B919	7-311
SGR	SUBTRACT (64)	RRE C N		IF				B909	7-311
SGRK	SUBTRACT (64)	RRF-a C DO		IF				B9E9	7-311
SH	SUBTRACT HALFWORD	RX-a C	A	IF			B ₂	4B	7-312
SHHHR	SUBTRACT HIGH (32)	RRF-a C HW		IF				B9C9	7-312
SHHLR	SUBTRACT HIGH (32)	RRF-a C HW		IF				B9D9	7-312
SHY	SUBTRACT HALFWORD	RXY-a C LD	A	IF			B ₂	E37B	7-312
SIGP	SIGNAL PROCESSOR	RS-a C	P	\$				AE	10-124
SL	SUBTRACT LOGICAL (32)	RX-a C	A				B ₂	5F	7-313
SLA	SHIFT LEFT SINGLE (32)	RS-a C		IF				8B	7-297
SLAG	SHIFT LEFT SINGLE (64)	RSY-a C N		IF				EB0B	7-297
SLAK	SHIFT LEFT SINGLE (32)	RSY-a C DO		IF				EBDD	7-297
SLB	SUBTRACT LOGICAL WITH BORROW (32)	RXY-a C N3	A				B ₂	E399	7-315
SLBG	SUBTRACT LOGICAL WITH BORROW (64)	RXY-a C N	A				B ₂	E389	7-315
SLBGR	SUBTRACT LOGICAL WITH BORROW (64)	RRE C N						B989	7-314
SLBR	SUBTRACT LOGICAL WITH BORROW (32)	RRE C N3						B999	7-314
SLDA	SHIFT LEFT DOUBLE	RS-a C	SP	IF				8F	7-296
SLDL	SHIFT LEFT DOUBLE LOGICAL	RS-a	SP					8D	7-297
SLDT	SHIFT SIGNIFICAND LEFT (long DFP)	RXF TF		Dt				ED40	20-49
SLFI	SUBTRACT LOGICAL IMMEDIATE (32)	RIL-a C EI						C25	7-313
SLG	SUBTRACT LOGICAL (64)	RXY-a C N	A				B ₂	E30B	7-313
SLGF	SUBTRACT LOGICAL (64←32)	RXY-a C N	A				B ₂	E31B	7-313
SLGFI	SUBTRACT LOGICAL IMMEDIATE (64←32)	RIL-a C EI						C24	7-313
SLGFR	SUBTRACT LOGICAL (64←32)	RRE C N						B91B	7-313
SLGR	SUBTRACT LOGICAL (64)	RRE C N						B90B	7-313
SLGRK	SUBTRACT LOGICAL (64)	RRF-a C DO						B9EB	7-313
SLHHHR	SUBTRACT LOGICAL HIGH (32)	RRF-a C HW						B9CB	7-314
SLHHLR	SUBTRACT LOGICAL HIGH (32)	RRF-a C HW						B9DB	7-314
SLL	SHIFT LEFT SINGLE LOGICAL (32)	RS-a						89	7-298
SLLG	SHIFT LEFT SINGLE LOGICAL (64)	RSY-a N						EB0D	7-298
SLLK	SHIFT LEFT SINGLE LOGICAL (32)	RSY-a DO						EBDF	7-298
SLR	SUBTRACT LOGICAL (32)	RR C						1F	7-313
SLRK	SUBTRACT LOGICAL (32)	RRF-a C DO						B9FB	7-313
SLXT	SHIFT SIGNIFICAND LEFT (extended DFP)	RXF TF	SP	Dt				ED48	20-49
SLY	SUBTRACT LOGICAL (32)	RXY-a C LD	A				B ₂	E35F	7-313
SP	SUBTRACT DECIMAL	SS-b C	A	Dd DF		ST	B ₁ B ₂	FB	8-13
SPKA	SET PSW KEY FROM ADDRESS	S	Q					B20A	10-115
SPM	SET PROGRAM MASK	RR L						04	7-296

Figure B-2. Instructions Arranged by Mnemonic (Part 15 of 18)

Mne- monic	Name	Characteristics						Op- code	Page		
SPT	SET CPU TIMER	S	P	A	SP			B ₂	B208	10-114	
SPX	SET PREFIX	S	P	A	SP	\$		B ₂	B210	10-114	
SQD	SQUARE ROOT (long HFP)	RXE		A		Da	SQ	B ₂	ED35	18-23	
SQDB	SQUARE ROOT (long BFP)	RXE		A		Db	Xi Xx	B ₂	ED15	19-38	
SQDBR	SQUARE ROOT (long BFP)	RRE				Db	Xi Xx		B315	19-38	
SQDR	SQUARE ROOT (long HFP)	RRE				Da	SQ		B244	18-23	
SQE	SQUARE ROOT (short HFP)	RXE		A		Da	SQ	B ₂	ED34	18-23	
SQEB	SQUARE ROOT (short BFP)	RXE		A		Db	Xi Xx	B ₂	ED14	19-38	
SQEBR	SQUARE ROOT (short BFP)	RRE				Db	Xi Xx		B314	19-38	
SQER	SQUARE ROOT (short HFP)	RRE				Da	SQ		B245	18-23	
SQXBR	SQUARE ROOT (extended BFP)	RRE			SP	Db	Xi Xx		B316	19-38	
SQXR	SQUARE ROOT (extended HFP)	RRE			SP	Da	SQ		B336	18-23	
SR	SUBTRACT (32)	RR	C				IF		1B	7-311	
SRA	SHIFT RIGHT SINGLE (32)	RS-a	C						8A	7-300	
SRAG	SHIFT RIGHT SINGLE (64)	RSY-a	C	N					EB0A	7-300	
SRAK	SHIFT RIGHT SINGLE (32)	RSY-a	C	DO					EBDC	7-300	
SRDA	SHIFT RIGHT DOUBLE	RS-a	C			SP			8E	7-299	
SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS-a				SP			8C	7-299	
SRDT	SHIFT SIGNIFICAND RIGHT (long DFP)	RXF		TF		Dt			ED41	20-50	
SRK	SUBTRACT (32)	RRF-a	C	DO			IF		B9F9	7-311	
SRL	SHIFT RIGHT SINGLE LOGICAL (32)	RS-a							88	7-301	
SRLG	SHIFT RIGHT SINGLE LOGICAL (64)	RSY-a		N					EB0C	7-301	
SRLK	SHIFT RIGHT SINGLE LOGICAL (32)	RSY-a		DO					EBDE	7-301	
SRNM	SET BFP ROUNDING MODE (2 bit)	S				Db			B299	9-45	
SRNMB	SET BFP ROUNDING MODE (3 bit)	S		F		SP	Db		B2B8	9-45	
SRNMT	SET DFP ROUNDING MODE	S		TR			Dt		B2B9	9-45	
SRP	SHIFT AND ROUND DECIMAL	SS-c	C		A		Dd DF	ST	B ₁ B ₂	F0 8-12	
SRST	SEARCH STRING	RRE	C		A	SP	IC		G0	B25E 7-291	
SRSTU	SEARCH STRING UNICODE	RRE	C	E3	A	SP	IC		G0	B9BE 7-293	
SRXT	SHIFT SIGNIFICAND RIGHT (extended DFP)	RXF		TF		SP	Dt			ED49 20-50	
SSAIR	SET SECONDARY ASN WITH INSTANCE	RRE		RA	A ¹		Z ⁷ T ¢			B99F 10-115	
SSAR	SET SECONDARY ASN	RRE			A ¹		Z ³ T ¢			B225 10-115	
SSCH	START SUBCHANNEL	S		C	P	A	SP		OP ¢ GS	B ₂ B233 14-15	
SSKE	SET STORAGE KEY EXTENDED	RRF-c	C		P	A ¹			II ¢	B22B 10-121	
SSM	SET SYSTEM MASK	S			P	A	SP		SO		B ₂ 80 10-124
ST	STORE (32)	RX-a				A		ST		B ₂ 50 7-301	
STAM	STORE ACCESS MULTIPLE	RS-a				A	SP	ST		UB 9B 7-302	
STAMY	STORE ACCESS MULTIPLE	RSY-a		LD		A	SP	ST		UB EB9B 7-302	
STAP	STORE CPU ADDRESS	S			P	A	SP	ST		B ₂ B212 10-126	
STC	STORE CHARACTER	RX-a				A		ST		B ₂ 42 7-302	
STCH	STORE CHARACTER HIGH (8)	RXY-a		HW		A		ST		B ₂ E3C3 7-303	
STCK	STORE CLOCK	S		C		A		ST		B ₂ B205 7-304	
STCKC	STORE CLOCK COMPARATOR	S			P	A	SP	ST		B ₂ B207 10-126	
STCKE	STORE CLOCK EXTENDED	S		C		A		ST		B ₂ B278 7-305	
STCKF	STORE CLOCK FAST	S		C	SC	A		ST		B ₂ B27C 7-304	
STCM	STORE CHARACTERS UNDER MASK (low)	RS-b				A		ST		B ₂ BE 7-303	
STCMH	STORE CHARACTERS UNDER MASK (high)	RSY-b		N		A		ST		B ₂ EB2C 7-303	
STCMY	STORE CHARACTERS UNDER MASK (low)	RSY-b		LD		A		ST		B ₂ EB2D 7-303	
STCPS	STORE CHANNEL PATH STATUS	S			P	A	SP		¢	ST B ₂ B23A 14-17	
STCRW	STORE CHANNEL REPORT WORD	S		C		P	A	SP	¢	ST B ₂ B239 14-17	
STCTG	STORE CONTROL (64)	RSY-a		N		P	A	SP		ST B ₂ EB25 10-126	
STCTL	STORE CONTROL (32)	RS-a				P	A	SP		ST B ₂ B6 10-126	
STCY	STORE CHARACTER	RXY-a		LD		A		ST		B ₂ E372 7-302	
STD	STORE (long)	RX-a				A		Da		ST B ₂ 60 9-47	
STDY	STORE (long)	RXY-a		LD		A		Da		ST B ₂ ED67 9-47	

Figure B-2. Instructions Arranged by Mnemonic (Part 16 of 18)

Mne- monic	Name	Characteristics						Op- code	Page
STE	STORE (short)	RX-a		A	Da	ST	B ₂	70	9-47
STEY	STORE (short)	RXY-a	LD	A	Da	ST	B ₂	ED66	9-47
STFH	STORE HIGH (32)	RXY-a	HW	A		ST	B ₂	E3CB	7-308
STFL	STORE FACILITY LIST	S	N3	P				B2B1	10-128
STFLE	STORE FACILITY LIST EXTENDED	S	C FL	A	SP	ST	B ₂	B2B0	7-306
STFPC	STORE FPC	S		A		ST	B ₂	B29C	9-47
STG	STORE (64)	RXY-a	N	A		ST	B ₂	E324	7-301
STGRL	STORE RELATIVE LONG (64)	RIL-b	GE	A	SP	ST		C4B	7-301
STH	STORE HALFWORD	RX-a		A		ST	B ₂	40	7-307
STHH	STORE HALFWORD HIGH (16)	RXY-a	HW	A		ST	B ₂	E3C7	7-308
STHRL	STORE HALFWORD RELATIVE LONG	RIL-b	GE	A		ST		C47	7-307
STHY	STORE HALFWORD	RXY-a	LD	A		ST	B ₂	E370	7-307
STIDP	STORE CPU ID	S		P	A SP	ST	B ₂	B202	10-127
STM	STORE MULTIPLE (32)	RS-a		A		ST	B ₂	90	7-308
STMG	STORE MULTIPLE (64)	RSY-a	N	A		ST	B ₂	EB24	7-308
STMH	STORE MULTIPLE HIGH	RSY-a	N	A		ST	B ₂	EB26	7-309
STMY	STORE MULTIPLE (32)	RSY-a	LD	A		ST	B ₂	EB90	7-308
STNSM	STORE THEN AND SYSTEM MASK	SI		P	A	ST	B ₁	AC	10-149
STOC	STORE ON CONDITION (32)	RSY-b	CL	A		ST	B ₂	EBF3	7-309
STOCG	STORE ON CONDITION (64)	RSY-b	CL	A		ST	B ₂	EBE3	7-309
STOSM	STORE THEN OR SYSTEM MASK	SI		P	A SP	ST	B ₁	AD	10-149
STPQ	STORE PAIR TO QUADWORD	RXY-a	N	A	SP	ST	B ₂	E38E	7-310
STPT	STORE CPU TIMER	S		P	A SP	ST	B ₂	B209	10-128
STPX	STORE PREFIX	S		P	A SP	ST	B ₂	B211	10-128
STRAG	STORE REAL ADDRESS	SSE	N	P	A ¹ SP	ST	B ₁	BP	E502
STRL	STORE RELATIVE LONG (32)	RIL-b	GE	A	SP	ST		C4F	7-301
STRV	STORE REVERSED (32)	RXY-a	N3	A		ST	B ₂	E33E	7-310
STRVG	STORE REVERSED (64)	RXY-a	N	A		ST	B ₂	E32F	7-311
STRVH	STORE REVERSED (16)	RXY-a	N3	A		ST	B ₂	E33F	7-310
STSCH	STORE SUBCHANNEL	S	C	P	A SP	OP	¢	GS	B234
STSI	STORE SYSTEM INFORMATION	S	C	P	A SP			GM	B27D
STURA	STORE USING REAL ADDRESS (32)	RRE		P	A ¹ SP	SU			B246
STURG	STORE USING REAL ADDRESS (64)	RRE	N	P	A ¹ SP	SU			B925
STY	STORE (32)	RXY-a	LD	A		ST	B ₂	E350	7-301
SVC	SUPERVISOR CALL	I						0A	7-315
SU	SUBTRACT UNNORMALIZED (short HFP)	RX-a	C	A		Da	EO	LS	B ₂
SUR	SUBTRACT UNNORMALIZED (short HFP)	RR	C			Da	EO	LS	
SW	SUBTRACT UNNORMALIZED (long HFP)	RX-a	C	A		Da	EO	LS	B ₂
SWR	SUBTRACT UNNORMALIZED (long HFP)	RR	C			Da	EO	LS	
SXBR	SUBTRACT (extended BFP)	RRE	C		SP	Db	Xi	Xo Xu Xx	
SXR	SUBTRACT NORMALIZED (extended HFP)	RR	C		SP	Da	EU	EO LS	
SXTR	SUBTRACT (extended DFP)	RRF-a	C TF		SP	Dt	Xi	Xo Xu Xx	
SXTRA	SUBTRACT (extended DFP)	RRF-a	C F		SP	Dt	Xi	Xo Xu Xx Xq	
SY	SUBTRACT (32)	RXY-a	C LD	A		IF			B ₂
TAM	TEST ADDRESSING MODE	E	C N3						
TAR	TEST ACCESS	RRE	C	A ¹					U ₁
TB	TEST BLOCK	RRE	C	P	A ¹	II	\$	G0	
TBDR	CONVERT HFP TO BFP (long)	RRF-e	C		SP	Da			
TBEDR	CONVERT HFP TO BFP (long to short)	RRF-e	C		SP	Da			
TCDB	TEST DATA CLASS (long BFP)	RXE	C			Db			
TCEB	TEST DATA CLASS (short BFP)	RXE	C			Db			
TCXB	TEST DATA CLASS (extended BFP)	RXE	C		SP	Db			
TDCDT	TEST DATA CLASS (long DFP)	RXE	C TF			Dt			
TDCET	TEST DATA CLASS (short DFP)	RXE	C TF			Dt			
TDCXT	TEST DATA CLASS (extended DFP)	RXE	C TF		SP	Dt			

Figure B-2. Instructions Arranged by Mnemonic (Part 17 of 18)

Mnemonic	Name	Characteristics						Op-code	Page
TDGDT	TEST DATA GROUP (long DFP)	RXE C TF		Dt				ED55	20-52
TDGET	TEST DATA GROUP (short DFP)	RXE C TF		Dt				ED51	20-52
TDGXT	TEST DATA GROUP (extended DFP)	RXE C TF	SP	Dt				ED59	20-52
THDER	CONVERT BFP TO HFP (short to long)	RRE C		Da				B358	9-27
THDR	CONVERT BFP TO HFP (long)	RRE C		Da				B359	9-27
TM	TEST UNDER MASK	SI C	A			B ₁		91	7-316
TMH	TEST UNDER MASK HIGH	RI-a C						A70	7-316
TMHH	TEST UNDER MASK (high high)	RI-a C N						A72	7-316
TMHL	TEST UNDER MASK (high low)	RI-a C N						A73	7-316
TML	TEST UNDER MASK LOW	RI-a C						A71	7-316
TMLH	TEST UNDER MASK (low high)	RI-a C N						A70	7-316
TMLL	TEST UNDER MASK (low low)	RI-a C N						A71	7-316
TMY	TEST UNDER MASK	SIY C LD	A			B ₁		EB51	7-316
TP	TEST DECIMAL	RSL C E2	A			B ₁ B ₂		EBC0	8-13
TPI	TEST PENDING INTERRUPTION	S C	P A ⁺ SP	¢		ST	B ₂	B236	14-19
TPROT	TEST PROTECTION	SSE C	P A ¹			ST	B ₁	E501	10-155
TR	TRANSLATE	SS-a	A			ST	B ₁ B ₂	DC	7-317
TRACE	TRACE (32)	RS-a	P A SP	T ¢			B ₂	99	10-157
TRACG	TRACE (64)	RSY-a N	P A SP	T ¢			B ₂	EB0F	10-157
TRAP2	TRAP	E	A	SO T		B ST		01FF	10-159
TRAP4	TRAP	S	A	SO T		B ST		B2FF	10-159
TRE	TRANSLATE EXTENDED	RRE C	A SP	IC		ST	R ₁ R ₂	B2A5	7-324
TROO	TRANSLATE ONE TO ONE	RRF-c C E2	A SP	IC GM		ST	RM R ₂	B993	7-327
TROT	TRANSLATE ONE TO TWO	RRF-c C E2	A SP	IC GM		ST	RM R ₂	B992	7-327
TRT	TRANSLATE AND TEST	SS-a C	A	GM		ST	B ₁ B ₂	DD	7-318
TRTE	TRANSLATE AND TEST EXTENDED	RRF-c C PE	A SP	IC		ST	RM	B9BF	7-319
TRTO	TRANSLATE TWO TO ONE	RRF-c C E2	A SP	IC GM		ST	RM R ₂	B991	7-327
TRTR	TRANSLATE AND TEST REVERSE	SS-a C E3	A	GM		ST	B ₁ B ₂	D0	7-323
TRTRE	TRANSLATE AND TEST REVERSE EXTENDED	RRF-c C PE	A SP	IC		ST	RM	B9BD	7-319
TRTT	TRANSLATE TWO TO TWO	RRF-c C E2	A SP	IC GM		ST	RM R ₂	B990	7-327
TS	TEST AND SET	S C	A	\$		ST	B ₂	93	7-316
TSCH	TEST SUBCHANNEL	S C	P A SP	OP ¢ GS		ST	B ₂	B235	14-20
UNPK	UNPACK	SS-b	A			ST	B ₁ B ₂	F3	7-331
UNPKA	UNPACK ASCII	SS-a C E2	A SP			ST	B ₁ B ₂	EA	7-332
UNPKU	UNPACK UNICODE	SS-a C E2	A SP			ST	B ₁ B ₂	E2	7-333
UPT	UPDATE TREE	E C	A SP	II GM I4		ST		0102	7-334
X	EXCLUSIVE OR (32)	RX-a C	A				B ₂	57	7-205
XC	EXCLUSIVE OR (character)	SS-a C	A			ST	B ₁ B ₂	D7	7-206
XG	EXCLUSIVE OR (64)	RXY-a C N	A				B ₂	E382	7-205
XGR	EXCLUSIVE OR (64)	RRE C N						B982	7-205
XGRK	EXCLUSIVE OR (64)	RRF-a C DO						B9E7	7-205
XI	EXCLUSIVE OR (immediate)	SI C	A			ST	B ₁	97	7-205
XIHF	EXCLUSIVE OR IMMEDIATE (high)	RIL-a C EI						C06	7-206
XILF	EXCLUSIVE OR IMMEDIATE (low)	RIL-a C EI						C07	7-206
XIY	EXCLUSIVE OR (immediate)	SIY C LD	A			ST	B ₁	EB57	7-205
XR	EXCLUSIVE OR (32)	RR C						17	7-205
XRK	EXCLUSIVE OR (32)	RRF-a C DO						B9F7	7-205
XSCH	CANCEL SUBCHANNEL	S C	P	OP ¢ GS				B276	14-4
XY	EXCLUSIVE OR (32)	RXY-a C LD	A				B ₂	E357	7-205
ZAP	ZERO AND ADD	SS-b C	A	Dd DF		ST	B ₁ B ₂	F8	8-13

Figure B-2. Instructions Arranged by Mnemonic (Part 18 of 18)

Instructions Arranged by Operation Code

Op-code	Name	Mne-monic	Characteristics								Page
0101	PROGRAM RETURN	PR	E	L	Q A ¹ SP	Z ⁴ T c ²	B ST				10-94
0102	UPDATE TREE	UPT	E	C	A SP	II GM I4	ST				7-334
0104	PERFORM TIMING FACILITY FUNCTION	PTFF	E	C TS	Q A SP	GM	ST				10-73
0107	SET CLOCK PROGRAMMABLE FIELD	SCKPF	E		P SP	G0					10-113
010A	PERFORM FLOATING-POINT OPERATION	PFPO	E	PF	SP	Da Xi Xo Xu Xx Xq					9-34
010B	TEST ADDRESSING MODE	TAM	E	C N3							7-315
010C	SET ADDRESSING MODE (24)	SAM24	E	N3	SP	T					7-295
010D	SET ADDRESSING MODE (31)	SAM31	E	N3	SP	T					7-295
010E	SET ADDRESSING MODE (64)	SAM64	E	N		T					7-295
01FF	TRAP	TRAP2	E		A	SO T	B ST				10-159
04	SET PROGRAM MASK	SPM	RR	L							7-296
05	BRANCH AND LINK	BALR	RR			T	B				7-33
06	BRANCH ON COUNT (32)	BCTR	RR				B				7-37
07	BRANCH ON CONDITION	BCR	RR			c ¹	B				7-36
0A	SUPERVISOR CALL	SVC	I			c					7-315
0B	BRANCH AND SET MODE	BSM	RR			T	B				7-36
0C	BRANCH AND SAVE AND SET MODE	BASSM	RR			T	B				7-35
0D	BRANCH AND SAVE	BASR	RR			T	B				7-34
0E	MOVE LONG	MVCL	RR	C	A SP	II	ST	R ₁	R ₂		7-234
0F	COMPARE LOGICAL LONG	CLCL	RR	C	A SP	II		R ₁	R ₂		7-135
10	LOAD POSITIVE (32)	LPR	RR	C		IF					7-230
11	LOAD NEGATIVE (32)	LNR	RR	C							7-227
12	LOAD AND TEST (32)	LTR	RR	C							7-219
13	LOAD COMPLEMENT (32)	LCR	RR	C		IF					7-221
14	AND (32)	NR	RR	C							7-31
15	COMPARE LOGICAL (32)	CLR	RR	C							7-130
16	OR (32)	OR	RR	C							7-250
17	EXCLUSIVE OR (32)	XR	RR	C							7-205
18	LOAD (32)	LR	RR								7-213
19	COMPARE (32)	CR	RR	C							7-112
1A	ADD (32)	AR	RR	C		IF					7-24
1B	SUBTRACT (32)	SR	RR	C		IF					7-311
1C	MULTIPLY (64←32)	MR	RR		SP						7-247
1D	DIVIDE (32←64)	DR	RR		SP	IK					7-203
1E	ADD LOGICAL (32)	ALR	RR	C							7-28
1F	SUBTRACT LOGICAL (32)	SLR	RR	C							7-313
20	LOAD POSITIVE (long HFP)	LPDR	RR	C		Da					18-16
21	LOAD NEGATIVE (long HFP)	LNDR	RR	C		Da					18-15
22	LOAD AND TEST (long HFP)	LTDR	RR	C		Da					18-13
23	LOAD COMPLEMENT (long HFP)	LCDR	RR	C		Da					18-14
24	HALVE (long HFP)	HDR	RR			Da EU					18-13
25	LOAD ROUNDED (extended to long HFP)	LDXR	RR		SP	Da EO					18-16
25	LOAD ROUNDED (extended to long HFP)	LRDR	RR		SP	Da EO					18-16
26	MULTIPLY (extended HFP)	MXR	RR		SP	Da EU EO					18-17
27	MULTIPLY (long to extended HFP)	MXDR	RR		SP	Da EU EO					18-17
28	LOAD (long)	LDR	RR			Da					9-30
29	COMPARE (long HFP)	CDR	RR	C		Da					18-10
2A	ADD NORMALIZED (long HFP)	ADR	RR	C		Da EU EO LS					18-8
2B	SUBTRACT NORMALIZED (long HFP)	SDR	RR	C		Da EU EO LS					18-24
2C	MULTIPLY (long HFP)	MDR	RR			Da EU EO					18-17
2D	DIVIDE (long HFP)	DDR	RR			Da EU EO FK					18-11

Figure B-3. Instructions Arranged by Operation Code (Part 1 of 18)

Op-code	Name	Mne-monic	Characteristics							Page	
2E	ADD UNNORMALIZED (long HFP)	AWR	RR	C		Da	EO	LS			18-9
2F	SUBTRACT UNNORMALIZED (long HFP)	SWR	RR	C		Da	EO	LS			18-24
30	LOAD POSITIVE (short HFP)	LPER	RR	C		Da					18-16
31	LOAD NEGATIVE (short HFP)	LNER	RR	C		Da					18-15
32	LOAD AND TEST (short HFP)	LTER	RR	C		Da					18-13
33	LOAD COMPLEMENT (short HFP)	LCER	RR	C		Da					18-14
34	HALVE (short HFP)	HER	RR			Da	EU				18-13
35	LOAD ROUNDED (long to short HFP)	LEDR	RR			Da	EO				18-16
35	LOAD ROUNDED (long to short HFP)	LRER	RR			Da	EO				18-16
36	ADD NORMALIZED (extended HFP)	AXR	RR	C	SP	Da	EU	EO	LS		18-8
37	SUBTRACT NORMALIZED (extended HFP)	SXR	RR	C	SP	Da	EU	EO	LS		18-24
38	LOAD (short)	LER	RR			Da					9-30
39	COMPARE (short HFP)	CER	RR	C		Da					18-10
3A	ADD NORMALIZED (short HFP)	AER	RR	C		Da	EU	EO	LS		18-8
3B	SUBTRACT NORMALIZED (short HFP)	SER	RR	C		Da	EU	EO	LS		18-24
3C	MULTIPLY (short to long HFP)	MDER	RR			Da	EU	EO			18-17
3C	MULTIPLY (short to long HFP)	MER	RR			Da	EU	EO			18-17
3D	DIVIDE (short HFP)	DER	RR			Da	EU	EO	FK		18-11
3E	ADD UNNORMALIZED (short HFP)	AUR	RR	C		Da	EO	LS			18-9
3F	SUBTRACT UNNORMALIZED (short HFP)	SUR	RR	C		Da	EO	LS			18-24
40	STORE HALFWORD	STH	RX-a		A				ST	B ₂	7-307
41	LOAD ADDRESS	LA	RX-a								7-215
42	STORE CHARACTER	STC	RX-a		A				ST	B ₂	7-302
43	INSERT CHARACTER	IC	RX-a		A					B ₂	7-211
44	EXECUTE	EX	RX-a		AI SP		EX				7-207
45	BRANCH AND LINK	BAL	RX-a						B		7-33
46	BRANCH ON COUNT (32)	BCT	RX-a						B		7-37
47	BRANCH ON CONDITION	BC	RX-b						B		7-36
48	LOAD HALFWORD (32)	LH	RX-a		A					B ₂	7-221
49	COMPARE HALFWORD (32←16)	CH	RX-a	C	A					B ₂	7-128
4A	ADD HALFWORD	AH	RX-a	C	A		IF			B ₂	7-27
4B	SUBTRACT HALFWORD	SH	RX-a	C	A		IF			B ₂	7-312
4C	MULTIPLY HALFWORD (32)	MH	RX-a		A					B ₂	7-248
4D	BRANCH AND SAVE	BAS	RX-a						B		7-34
4E	CONVERT TO DECIMAL (32)	CVD	RX-a		A				ST	B ₂	7-182
4F	CONVERT TO BINARY (32)	CVB	RX-a		A		Dd	IK		B ₂	7-181
50	STORE (32)	ST	RX-a		A				ST	B ₂	7-301
51	LOAD ADDRESS EXTENDED	LAE	RX-a							U ₁	BP
54	AND (32)	N	RX-a	C	A					B ₂	7-31
55	COMPARE LOGICAL (32)	CL	RX-a	C	A					B ₂	7-130
56	OR (32)	O	RX-a	C	A					B ₂	7-250
57	EXCLUSIVE OR (32)	X	RX-a	C	A					B ₂	7-205
58	LOAD (32)	L	RX-a		A					B ₂	7-213
59	COMPARE (32)	C	RX-a	C	A					B ₂	7-112
5A	ADD (32)	A	RX-a	C	A		IF			B ₂	7-25
5B	SUBTRACT (32)	S	RX-a	C	A		IF			B ₂	7-311
5C	MULTIPLY (64←32)	M	RX-a		A SP					B ₂	7-247
5D	DIVIDE (32←64)	D	RX-a		A SP			IK		B ₂	7-203
5E	ADD LOGICAL (32)	AL	RX-a	C	A					B ₂	7-28
5F	SUBTRACT LOGICAL (32)	SL	RX-a	C	A					B ₂	7-313
60	STORE (long)	STD	RX-a		A		Da		ST	B ₂	9-47
67	MULTIPLY (long to extended HFP)	MXD	RX-a		A SP	Da	EU	EO		B ₂	18-17
68	LOAD (long)	LD	RX-a		A	Da				B ₂	9-30
69	COMPARE (long HFP)	CD	RX-a	C	A	Da				B ₂	18-10
6A	ADD NORMALIZED (long HFP)	AD	RX-a	C	A	Da	EU	EO	LS	B ₂	18-8

Figure B-3. Instructions Arranged by Operation Code (Part 2 of 18)

Op-code	Name	Mne-monic	Characteristics						Page			
6B	SUBTRACT NORMALIZED (long HFP)	SD	RX-a	C	A	Da	EU	EO	LS	B ₂	18-24	
6C	MULTIPLY (long HFP)	MD	RX-a		A	Da	EU	EO		B ₂	18-17	
6D	DIVIDE (long HFP)	DD	RX-a		A	Da	EU	EO	FK	B ₂	18-12	
6E	ADD UNNORMALIZED (long HFP)	AW	RX-a	C	A	Da		EO	LS	B ₂	18-9	
6F	SUBTRACT UNNORMALIZED (long HFP)	SW	RX-a	C	A	Da		EO	LS	B ₂	18-24	
70	STORE (short)	STE	RX-a		A	Da				ST	B ₂	9-47
71	MULTIPLY SINGLE (32)	MS	RX-a		A						B ₂	7-249
78	LOAD (short)	LE	RX-a		A	Da					B ₂	9-30
79	COMPARE (short HFP)	CE	RX-a	C	A	Da					B ₂	18-10
7A	ADD NORMALIZED (short HFP)	AE	RX-a	C	A	Da	EU	EO	LS		B ₂	18-8
7B	SUBTRACT NORMALIZED (short HFP)	SE	RX-a	C	A	Da	EU	EO	LS		B ₂	18-24
7C	MULTIPLY (short to long HFP)	MDE	RX-a		A	Da	EU	EO			B ₂	18-17
7C	MULTIPLY (short to long HFP)	ME	RX-a		A	Da	EU	EO			B ₂	18-17
7D	DIVIDE (short HFP)	DE	RX-a		A	Da	EU	EO	FK		B ₂	18-12
7E	ADD UNNORMALIZED (short HFP)	AU	RX-a	C	A	Da		EO	LS		B ₂	18-9
7F	SUBTRACT UNNORMALIZED (short HFP)	SU	RX-a	C	A	Da		EO	LS		B ₂	18-24
80	SET SYSTEM MASK	SSM	S		P	A	SP	SO			B ₂	10-124
82	LOAD PSW	LPSW	S	L	P	A	SP		¢		B ₂	10-47
83	DIAGNOSE			DM	P	DM					MD	10-19
84	BRANCH RELATIVE ON INDEX HIGH (32)	BRXH	RSI							B		7-41
85	BRANCH RELATIVE ON INDEX LOW OR EQ. (32)	BRXLE	RSI							B		7-41
86	BRANCH ON INDEX HIGH (32)	BXH	RS-a							B		7-38
87	BRANCH ON INDEX LOW OR EQUAL (32)	BXLE	RS-a							B		7-38
88	SHIFT RIGHT SINGLE LOGICAL (32)	SRL	RS-a									7-301
89	SHIFT LEFT SINGLE LOGICAL (32)	SLL	RS-a									7-298
8A	SHIFT RIGHT SINGLE (32)	SRA	RS-a	C								7-300
8B	SHIFT LEFT SINGLE (32)	SLA	RS-a	C				IF				7-297
8C	SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS-a				SP					7-299
8D	SHIFT LEFT DOUBLE LOGICAL	SLDL	RS-a				SP					7-297
8E	SHIFT RIGHT DOUBLE	SRDA	RS-a	C			SP					7-299
8F	SHIFT LEFT DOUBLE	SLDA	RS-a	C			SP		IF			7-296
90	STORE MULTIPLE (32)	STM	RS-a		A					ST	B ₂	7-308
91	TEST UNDER MASK	TM	SI	C	A						B ₁	7-316
92	MOVE (immediate)	MVI	SI		A					ST	B ₁	7-233
93	TEST AND SET	TS	S	C	A			\$		ST	B ₂	7-316
94	AND (immediate)	NI	SI	C	A					ST	B ₁	7-32
95	COMPARE LOGICAL (immediate)	CLI	SI	C	A						B ₁	7-130
96	OR (immediate)	OI	SI	C	A					ST	B ₁	7-251
97	EXCLUSIVE OR (immediate)	XI	SI	C	A					ST	B ₁	7-205
98	LOAD MULTIPLE (32)	LM	RS-a		A						B ₂	7-226
99	TRACE (32)	TRACE	RS-a		P	A	SP	T	¢		B ₂	10-157
9A	LOAD ACCESS MULTIPLE	LAM	RS-a		A	SP					UB	7-214
9B	STORE ACCESS MULTIPLE	STAM	RS-a		A	SP				ST	UB	7-302
A50	INSERT IMMEDIATE (high high)	IIHH	RI-a	N								7-212
A51	INSERT IMMEDIATE (high low)	IIHL	RI-a	N								7-213
A52	INSERT IMMEDIATE (low high)	IILH	RI-a	N								7-213
A53	INSERT IMMEDIATE (low low)	IILL	RI-a	N								7-213
A54	AND IMMEDIATE (high high)	NIHH	RI-a	C	N							7-32
A55	AND IMMEDIATE (high low)	NIHL	RI-a	C	N							7-33
A56	AND IMMEDIATE (low high)	NILH	RI-a	C	N							7-33
A57	AND IMMEDIATE (low low)	NILL	RI-a	C	N							7-33
A58	OR IMMEDIATE (high high)	OIHH	RI-a	C	N							7-251
A59	OR IMMEDIATE (high low)	OIHL	RI-a	C	N							7-251
A5A	OR IMMEDIATE (low high)	OILH	RI-a	C	N							7-252
A5B	OR IMMEDIATE (low low)	OILL	RI-a	C	N							7-252

Figure B-3. Instructions Arranged by Operation Code (Part 3 of 18)

Op-code	Name	Mne-monic	Characteristics						Page
A5C	LOAD LOGICAL IMMEDIATE (high high)	LLIHH	RI-a	N					7-225
A5D	LOAD LOGICAL IMMEDIATE (high low)	LLIHL	RI-a	N					7-225
A5E	LOAD LOGICAL IMMEDIATE (low high)	LLILH	RI-a	N					7-225
A5F	LOAD LOGICAL IMMEDIATE (low low)	LLILL	RI-a	N					7-225
A70	TEST UNDER MASK (low high)	TMLH	RI-a	C N					7-316
A70	TEST UNDER MASK HIGH	TMH	RI-a	C					7-316
A71	TEST UNDER MASK (low low)	TMLL	RI-a	C N					7-316
A71	TEST UNDER MASK LOW	TML	RI-a	C					7-316
A72	TEST UNDER MASK (high high)	TMHH	RI-a	C N					7-316
A73	TEST UNDER MASK (high low)	TMHL	RI-a	C N					7-316
A74	BRANCH RELATIVE ON CONDITION	BRC	RI-c				B		7-40
A75	BRANCH RELATIVE AND SAVE	BRAS	RI-b				B		7-39
A76	BRANCH RELATIVE ON COUNT (32)	BRCT	RI-b				B		7-41
A77	BRANCH RELATIVE ON COUNT (64)	BRCTG	RI-b	N			B		7-41
A78	LOAD HALFWORD IMMEDIATE (32)	LHI	RI-a						7-222
A79	LOAD HALFWORD IMMEDIATE (64)	LGHI	RI-a	N					7-222
A7A	ADD HALFWORD IMMEDIATE (32)	AHI	RI-a	C		IF			7-27
A7B	ADD HALFWORD IMMEDIATE (64)	AGHI	RI-a	C N		IF			7-27
A7C	MULTIPLY HALFWORD IMMEDIATE (32)	MHI	RI-a						7-248
A7D	MULTIPLY HALFWORD IMMEDIATE (64)	MGHI	RI-a	N					7-248
A7E	COMPARE HALFWORD IMMEDIATE (32←16)	CHI	RI-a	C					7-128
A7F	COMPARE HALFWORD IMMEDIATE (64←16)	CGHI	RI-a	C N					7-128
A8	MOVE LONG EXTENDED	MVCLE	RS-a	C	A	SP IC	ST	R ₁ R ₃	7-238
A9	COMPARE LOGICAL LONG EXTENDED	CLCLE	RS-a	C	A	SP IC		R ₁ R ₃	7-138
AC	STORE THEN AND SYSTEM MASK	STNSM	SI		P A		ST	B ₁	10-149
AD	STORE THEN OR SYSTEM MASK	STOSM	SI		P A SP		ST	B ₁	10-149
AE	SIGNAL PROCESSOR	SIGP	RS-a	C	P	\$			10-124
AF	MONITOR CALL	MC	SI			ME			7-231
B1	LOAD REAL ADDRESS (32)	LRA	RX-a	C	P A ¹	SO		BP	10-49
B202	STORE CPU ID	STIDP	S		P A SP		ST	B ₂	10-127
B204	SET CLOCK	SCK	S	C	P A SP			B ₂	10-112
B205	STORE CLOCK	STCK	S	C	A	\$	ST	B ₂	7-304
B206	SET CLOCK COMPARATOR	SCKC	S		P A SP			B ₂	10-113
B207	STORE CLOCK COMPARATOR	STCKC	S		P A SP		ST	B ₂	10-126
B208	SET CPU TIMER	SPT	S		P A SP			B ₂	10-114
B209	STORE CPU TIMER	STPT	S		P A SP		ST	B ₂	10-128
B20A	SET PSW KEY FROM ADDRESS	SPKA	S		Q				10-115
B20B	INSERT PSW KEY	IPK	S		Q	G2			10-26
B20D	PURGE TLB	PTLB	S		P	\$			10-107
B210	SET PREFIX	SPX	S		P A SP	\$		B ₂	10-114
B211	STORE PREFIX	STPX	S		P A SP		ST	B ₂	10-128
B212	STORE CPU ADDRESS	STAP	S		P A SP		ST	B ₂	10-126
B218	PROGRAM CALL	PC	S		Q A ¹	Z ¹ T ¢ GM	B ST		10-81
B219	SET ADDRESS SPACE CONTROL	SAC	S		Q SP	SW ¢			10-110
B21A	COMPARE AND FORM CODEWORD	CFC	S	C	A SP	II GM I1			7-115
B221	INVALIDATE PAGE TABLE ENTRY	IPTE	RRF-a		P A ¹ SP	\$			10-32
B222	INSERT PROGRAM MASK	IPM	RRE						7-213
B223	INSERT VIRTUAL STORAGE KEY	IVSK	RRE		Q A ¹	SO		R ₂	10-27
B224	INSERT ADDRESS SPACE CONTROL	IAC	RRE	C	Q	SO			10-25
B225	SET SECONDARY ASN	SSAR	RRE		A ¹	Z ³ T ¢			10-115
B226	EXTRACT PRIMARY ASN	EPAR	RRE		Q	SO			10-20
B227	EXTRACT SECONDARY ASN	ESAR	RRE		Q	SO			10-21
B228	PROGRAM TRANSFER	PT	RRE		Q A ¹ SP	Z ² T ¢	B		10-98
B229	INSERT STORAGE KEY EXTENDED	ISKE	RRE		P A ¹				10-26
B22A	RESET REFERENCE BIT EXTENDED	RRBE	RRE	C	P A ¹				10-107

Figure B-3. Instructions Arranged by Operation Code (Part 4 of 18)

Op-code	Name	Mne-monic	Characteristics							Page
B22B	SET STORAGE KEY EXTENDED	SSKE	RRF-c C	P A ¹	II	¢				10-121
B22C	TEST BLOCK	TB	RRE C	P A ¹	II	\$ G0				10-152
B22D	DIVIDE (extended HFP)	DXR	RRE	SP	Da	EU EO FK				18-12
B22E	PAGE IN	PGIN	RRE C ES	P A ¹		¢				10-64
B22F	PAGE OUT	PGOUT	RRE C ES	P A ¹		¢				10-65
B230	CLEAR SUBCHANNEL	CSCH	S C	P	OP	¢ GS				14-5
B231	HALT SUBCHANNEL	HSCH	S C	P	OP	¢ GS				14-6
B232	MODIFY SUBCHANNEL	MSCH	S C	P A SP	OP	¢ GS		B ₂		14-8
B233	START SUBCHANNEL	SSCH	S C	P A SP	OP	¢ GS		B ₂		14-15
B234	STORE SUBCHANNEL	STSCH	S C	P A SP	OP	¢ GS	ST	B ₂		14-18
B235	TEST SUBCHANNEL	TSCH	S C	P A SP	OP	¢ GS	ST	B ₂		14-20
B236	TEST PENDING INTERRUPTION	TPI	S C	P A ⁺ SP		¢	ST	B ₂		14-19
B237	SET ADDRESS LIMIT	SAL	S	P	OP	¢ G1				14-12
B238	RESUME SUBCHANNEL	RSCH	S C	P	OP	¢ GS				14-10
B239	STORE CHANNEL REPORT WORD	STCRW	S C	P A SP		¢	ST	B ₂		14-17
B23A	STORE CHANNEL PATH STATUS	STCPS	S	P A SP		¢	ST	B ₂		14-17
B23B	RESET CHANNEL PATH	RCHP	S C	P						14-9
B23C	SET CHANNEL MONITOR	SCHM	S	P	OP	¢ GM				14-13
B240	BRANCH AND STACK	BAKR	RRE	A ¹	Z ⁵ T		B ST			10-10
B241	CHECKSUM	CKSM	RRE C	A SP	IC			R ₂		7-43
B244	SQUARE ROOT (long HFP)	SQDR	RRE		Da	SQ				18-23
B245	SQUARE ROOT (short HFP)	SQER	RRE		Da	SQ				18-23
B246	STORE USING REAL ADDRESS (32)	STURA	RRE	P A ¹ SP			SU			10-150
B247	MODIFY STACKED STATE	MSTA	RRE	A ¹ SP	SE		ST			10-53
B248	PURGE ALB	PALB	RRE	P		\$				10-107
B249	EXTRACT STACKED REGISTERS (32)	EREG	RRE	A ¹	SE			U ₁ U ₂		10-21
B24A	EXTRACT STACKED STATE	ESTA	RRE C	A ¹ SP	SE					10-23
B24B	LOAD USING REAL ADDRESS (32)	LURA	RRE	P A ¹ SP						10-53
B24C	TEST ACCESS	TAR	RRE C	A ¹				U ₁		10-150
B24D	COPY ACCESS	CPYA	RRE					U ₁ U ₂		7-203
B24E	SET ACCESS	SAR	RRE					U ₁		7-295
B24F	EXTRACT ACCESS	EAR	RRE					U ₂		7-208
B250	COMPARE AND SWAP AND PURGE	CSP	RRE C	P A ¹ SP		\$	ST	R ₂		10-17
B252	MULTIPLY SINGLE (32)	MSR	RRE							7-249
B254	MOVE PAGE	MVPG	RRE C	Q A SP		G0	ST	R ₁ R ₂		10-54
B255	MOVE STRING	MVST	RRE C	A SP	IC	G0	ST	R ₁ R ₂		7-245
B257	COMPARE UNTIL SUBSTRING EQUAL	CUSE	RRE C	A SP	II	GM		R ₁ R ₂		7-144
B258	BRANCH IN SUBSPACE GROUP	BSG	RRE	A ¹	SO T		B	R ₂		10-13
B25A	BRANCH AND SET AUTHORITY	BSA	RRE	Q A ¹	SO T		B			10-6
B25D	COMPARE LOGICAL STRING	CLST	RRE C	A SP	IC	G0		R ₁ R ₂		7-143
B25E	SEARCH STRING	SRST	RRE C	A SP	IC	G0		R ₂		7-291
B263	COMPRESSION CALL	CMPSC	RRE C	A SP	II Dd	GM	ST	R ₁ R ₂		7-147
B276	CANCEL SUBCHANNEL	XSCH	S C	P	OP	¢ GS				14-4
B277	RESUME PROGRAM	RP	S L	Q A SP	WE T		B	B ₂		10-108
B278	STORE CLOCK EXTENDED	STCKE	S C	A		\$	ST	B ₂		7-305
B279	SET ADDRESS SPACE CONTROL FAST	SACF	S	Q SP	SW					10-111
B27C	STORE CLOCK FAST	STCKF	S C SC	A			ST	B ₂		7-304
B27D	STORE SYSTEM INFORMATION	STSI	S C	P A SP		GM	ST	B ₂		10-130
B299	SET BFP ROUNDING MODE (2 bit)	SRNM	S		Db					9-45
B29C	STORE FPC	STFPC	S	A	Db		ST	B ₂		9-47
B29D	LOAD FPC	LFPC	S	A SP	Db			B ₂		9-31
B2A5	TRANSLATE EXTENDED	TRE	RRE C	A SP	IC		ST	R ₁ R ₂		7-324
B2A6	CONVERT UNICODE TO UTF-8	CUUTF	RRF-c C	A SP	IC		ST	R ₁ R ₂		7-186
B2A6	CONVERT UTF-16 TO UTF-8	CU21	RRF-c C	A SP	IC		ST	R ₁ R ₂		7-186
B2A7	CONVERT UTF-8 TO UNICODE	CUTFU	RRF-c C	A SP	IC		ST	R ₁ R ₂		7-195

Figure B-3. Instructions Arranged by Operation Code (Part 5 of 18)

Op-code	Name	Mne-monic	Characteristics										Page
B2A7	CONVERT UTF-8 TO UTF-16	CU12	RRF-c	C	A	SP	IC		ST	R ₁	R ₂	7-195	
B2B0	STORE FACILITY LIST EXTENDED	STFLE	S	C	FL	A	SP	G0	ST		B ₂	7-306	
B2B1	STORE FACILITY LIST	STFL	S		N3	P						10-128	
B2B2	LOAD PSW EXTENDED	LPSWE	S	L	N	P	A	SP	c		B ₂	10-48	
B2B8	SET BFP ROUNDING MODE (3 bit)	SRNMB	S		F		SP	Db				9-45	
B2B9	SET DFP ROUNDING MODE	SRNMT	S		TR			Dt				9-45	
B2BD	LOAD FPC AND SIGNAL	LFAS	S		XF	A	SP	Dt Xg			B ₂	9-31	
B2FF	TRAP	TRAP4	S			A		SO T	B	ST		10-159	
B300	LOAD POSITIVE (short BFP)	LPEBR	RRE	C				Db				19-33	
B301	LOAD NEGATIVE (short BFP)	LNEBR	RRE	C				Db				19-33	
B302	LOAD AND TEST (short BFP)	LTEBR	RRE	C				Db Xi				19-30	
B303	LOAD COMPLEMENT (short BFP)	LCEBR	RRE	C				Db				19-30	
B304	LOAD LENGTHENED (short to long BFP)	LDEBR	RRE					Db Xi				19-32	
B305	LOAD LENGTHENED (long to extended BFP)	LXDBR	RRE				SP	Db Xi				19-32	
B306	LOAD LENGTHENED (short to extended BFP)	LXEBR	RRE				SP	Db Xi				19-32	
B307	MULTIPLY (long to extended BFP)	MXDBR	RRE				SP	Db Xi				19-35	
B308	COMPARE AND SIGNAL (short BFP)	KEBR	RRE	C				Db Xi				19-17	
B309	COMPARE (short BFP)	CEBR	RRE	C				Db Xi				19-16	
B30A	ADD (short BFP)	AEBR	RRE	C				Db Xi X _o X _u X _x				19-11	
B30B	SUBTRACT (short BFP)	SEBR	RRE	C				Db Xi X _o X _u X _x				19-39	
B30C	MULTIPLY (short to long BFP)	MDEBR	RRE					Db Xi				19-35	
B30D	DIVIDE (short BFP)	DEBR	RRE					Db Xi X _z X _o X _u X _x				19-26	
B30E	MULTIPLY AND ADD (short BFP)	MAEBR	RRD					Db Xi X _o X _u X _x				19-37	
B30F	MULTIPLY AND SUBTRACT (short BFP)	MSEBR	RRD					Db Xi X _o X _u X _x				19-37	
B310	LOAD POSITIVE (long BFP)	LPDBR	RRE	C				Db				19-33	
B311	LOAD NEGATIVE (long BFP)	LNDBR	RRE	C				Db				19-33	
B312	LOAD AND TEST (long BFP)	LTDBR	RRE	C				Db Xi				19-30	
B313	LOAD COMPLEMENT (long BFP)	LCDBR	RRE	C				Db				19-30	
B314	SQUARE ROOT (short BFP)	SQEBR	RRE					Db Xi X _x				19-38	
B315	SQUARE ROOT (long BFP)	SQDBR	RRE					Db Xi X _x				19-38	
B316	SQUARE ROOT (extended BFP)	SQXBR	RRE				SP	Db Xi X _x				19-38	
B317	MULTIPLY (short BFP)	MEEBR	RRE					Db Xi X _o X _u X _x				19-35	
B318	COMPARE AND SIGNAL (long BFP)	KDBR	RRE	C				Db Xi				19-17	
B319	COMPARE (long BFP)	CDBR	RRE	C				Db Xi				19-16	
B31A	ADD (long BFP)	ADBR	RRE	C				Db Xi X _o X _u X _x				19-11	
B31B	SUBTRACT (long BFP)	SDBR	RRE	C				Db Xi X _o X _u X _x				19-39	
B31C	MULTIPLY (long BFP)	MDBR	RRE					Db Xi X _o X _u X _x				19-35	
B31D	DIVIDE (long BFP)	DDBR	RRE					Db Xi X _z X _o X _u X _x				19-26	
B31E	MULTIPLY AND ADD (long BFP)	MADBR	RRD					Db Xi X _o X _u X _x				19-37	
B31F	MULTIPLY AND SUBTRACT (long BFP)	MSDBR	RRD					Db Xi X _o X _u X _x				19-37	
B324	LOAD LENGTHENED (short to long HFP)	LDER	RRE					Da				18-15	
B325	LOAD LENGTHENED (long to extended HFP)	LXDR	RRE				SP	Da				18-15	
B326	LOAD LENGTHENED (short to extended HFP)	LXER	RRE				SP	Da				18-15	
B32E	MULTIPLY AND ADD (short HFP)	MAER	RRD	HM				Da EU EO				18-18	
B32F	MULTIPLY AND SUBTRACT (short HFP)	MSER	RRD	HM				Da EU EO				18-19	
B336	SQUARE ROOT (extended HFP)	SQXR	RRE				SP	Da SQ				18-23	
B337	MULTIPLY (short HFP)	MEER	RRE					Da EU EO				18-17	
B338	MULTIPLY AND ADD UNNRM. (long to ext. low HFP)	MAYLR	RRD	UE				Da				18-20	
B339	MULTIPLY UNNORM. (long to ext. low HFP)	MYLR	RRD	UE				Da				18-21	
B33A	MULTIPLY & ADD UNNORMALIZED (long to ext. HFP)	MAYR	RRD	UE			SP	Da				18-20	
B33B	MULTIPLY UNNORMALIZED (long to ext. HFP)	MYR	RRD	UE				Da				18-21	
B33C	MULTIPLY AND ADD UNNRM. (long to ext. high HFP)	MAYHR	RRD	UE				Da				18-20	
B33D	MULTIPLY UNNORM. (long to ext. high HFP)	MYHR	RRD	UE				Da				18-21	
B33E	MULTIPLY AND ADD (long HFP)	MADR	RRD	HM				Da EU EO				18-18	
B33F	MULTIPLY AND SUBTRACT (long HFP)	MSDR	RRD	HM				Da EU EO				18-19	

Figure B-3. Instructions Arranged by Operation Code (Part 6 of 18)

Op-code	Name	Mne-monic	Characteristics						Page
B340	LOAD POSITIVE (extended BFP)	LPXBR	RRE C	SP	Db				19-33
B341	LOAD NEGATIVE (extended BFP)	LNxBR	RRE C	SP	Db				19-33
B342	LOAD AND TEST (extended BFP)	LTXBR	RRE C	SP	Db Xi				19-30
B343	LOAD COMPLEMENT (extended BFP)	LCXBR	RRE C	SP	Db				19-30
B344	LOAD ROUNDED (long to short BFP)	LEDBR	RRE		Db Xi	Xo Xu Xx			19-34
B344	LOAD ROUNDED (long to short BFP)	LEDBRA	RRF-e F	SP	Db Xi	Xo Xu Xx			19-34
B345	LOAD ROUNDED (extended to long BFP)	LDXBR	RRE	SP	Db Xi	Xo Xu Xx			19-34
B345	LOAD ROUNDED (extended to long BFP)	LDXBRA	RRF-e F	SP	Db Xi	Xo Xu Xx			19-34
B346	LOAD ROUNDED (extended to short BFP)	LEXBR	RRE	SP	Db Xi	Xo Xu Xx			19-34
B346	LOAD ROUNDED (extended to short BFP)	LEXBRA	RRF-e F	SP	Db Xi	Xo Xu Xx			19-34
B347	LOAD FP INTEGER (extended BFP)	FIXBR	RRF-e	SP	Db Xi	Xx			19-31
B347	LOAD FP INTEGER (extended BFP)	FIXBRA	RRF-e F	SP	Db Xi	Xx			19-31
B348	COMPARE AND SIGNAL (extended BFP)	KXBR	RRE C	SP	Db Xi				19-17
B349	COMPARE (extended BFP)	CXBR	RRE C	SP	Db Xi				19-16
B34A	ADD (extended BFP)	AXBR	RRE C	SP	Db Xi	Xo Xu Xx			19-11
B34B	SUBTRACT (extended BFP)	SXBR	RRE C	SP	Db Xi	Xo Xu Xx			19-39
B34C	MULTIPLY (extended BFP)	MXBR	RRE	SP	Db Xi	Xo Xu Xx			19-35
B34D	DIVIDE (extended BFP)	DXBR	RRE	SP	Db Xi Xz	Xo Xu Xx			19-26
B350	CONVERT HFP TO BFP (long to short)	TBEDR	RRF-e C	SP	Da				9-28
B351	CONVERT HFP TO BFP (long)	TBDR	RRF-e C	SP	Da				9-28
B353	DIVIDE TO INTEGER (short BFP)	DIEBR	RRF-b C	SP	Db Xi	Xu Xx			19-27
B357	LOAD FP INTEGER (short BFP)	FIEBR	RRF-e	SP	Db Xi	Xx			19-31
B357	LOAD FP INTEGER (short BFP)	FIEBRA	RRF-e F	SP	Db Xi	Xx			19-31
B358	CONVERT BFP TO HFP (short to long)	THDER	RRE C		Da				9-27
B359	CONVERT BFP TO HFP (long)	THDR	RRE C		Da				9-27
B35B	DIVIDE TO INTEGER (long BFP)	DIDBR	RRF-b C	SP	Db Xi	Xu Xx			19-27
B35F	LOAD FP INTEGER (long BFP)	FIDBR	RRF-e	SP	Db Xi	Xx			19-31
B35F	LOAD FP INTEGER (long BFP)	FIDBRA	RRF-e F	SP	Db Xi	Xx			19-31
B360	LOAD POSITIVE (extended HFP)	LPXR	RRE C	SP	Da				18-16
B361	LOAD NEGATIVE (extended HFP)	LNXR	RRE C	SP	Da				18-15
B362	LOAD AND TEST (extended HFP)	LTXR	RRE C	SP	Da				18-13
B363	LOAD COMPLEMENT (extended HFP)	LCXR	RRE C	SP	Da				18-14
B365	LOAD (extended)	LXR	RRE	SP	Da				9-30
B366	LOAD ROUNDED (extended to short HFP)	LEXR	RRE	SP	Da EO				18-16
B367	LOAD FP INTEGER (extended HFP)	FIXR	RRE	SP	Da				18-14
B369	COMPARE (extended HFP)	CXR	RRE C	SP	Da				18-10
B370	LOAD POSITIVE (long)	LPDFR	RRE FS		Da				9-34
B371	LOAD NEGATIVE (long)	LNDFR	RRE FS		Da				9-33
B372	COPY SIGN (long)	CPSDR	RRF-b FS		Da				9-29
B373	LOAD COMPLEMENT (long)	LCDFR	RRE FS		Da				9-30
B374	LOAD ZERO (short)	LZER	RRE		Da				9-34
B375	LOAD ZERO (long)	LZDR	RRE		Da				9-34
B376	LOAD ZERO (extended)	LZXR	RRE	SP	Da				9-34
B377	LOAD FP INTEGER (short HFP)	FIER	RRE		Da				18-14
B37F	LOAD FP INTEGER (long HFP)	FIDR	RRE		Da				18-14
B384	SET FPC	SFPC	RRE	SP	Db				9-46
B385	SET FPC AND SIGNAL	SFASR	RRE XF	SP	Dt Xg				9-46
B38C	EXTRACT FPC	EFPC	RRE		Db				9-30
B390	CONVERT FROM LOGICAL (32 to short BFP)	CELFBR	RRF-e F	SP	Db	Xx			19-20
B391	CONVERT FROM LOGICAL (32 to long BFP)	CDLFBR	RRF-e F	SP	Db				19-20
B392	CONVERT FROM LOGICAL (32 to extended BFP)	CXLfBR	RRF-e F	SP	Db				19-20
B394	CONVERT FROM FIXED (32 to short BFP)	CEfBR	RRE		Db	Xx			19-18
B394	CONVERT FROM FIXED (32 to short BFP)	CEfBRA	RRF-e F	SP	Db	Xx			19-18
B395	CONVERT FROM FIXED (32 to long BFP)	CDfBR	RRE		Db				19-18
B395	CONVERT FROM FIXED (32 to long BFP)	CDfBRA	RRF-e F	SP	Db				19-18

Figure B-3. Instructions Arranged by Operation Code (Part 7 of 18)

Op-code	Name	Mne-monic	Characteristics							Page
B396	CONVERT FROM FIXED (32 to extended BFP)	CXFBR	RRE		SP	Db			19-18	
B396	CONVERT FROM FIXED (32 to extended BFP)	CXFBRA	RRF-e	F	SP	Db			19-18	
B398	CONVERT TO FIXED (short BFP to 32)	CFEBR	RRF-e	C	SP	Db	Xi	Xx	19-21	
B398	CONVERT TO FIXED (short BFP to 32)	CFEBRA	RRF-e	C	F	SP	Db	Xi	Xx	19-21
B399	CONVERT TO FIXED (long BFP to 32)	CFDBR	RRF-e	C	SP	Db	Xi	Xx	19-21	
B399	CONVERT TO FIXED (long BFP to 32)	CFDBRA	RRF-e	C	F	SP	Db	Xi	Xx	19-21
B39A	CONVERT TO FIXED (extended BFP to 32)	CFXBR	RRF-e	C	SP	Db	Xi	Xx	19-21	
B39A	CONVERT TO FIXED (extended BFP to 32)	CFXBRA	RRF-e	C	F	SP	Db	Xi	Xx	19-21
B39C	CONVERT TO LOGICAL (short BFP to 32)	CLFEBR	RRF-e	C	F	SP	Db	Xi	Xx	19-23
B39D	CONVERT TO LOGICAL (long BFP to 32)	CLFDBR	RRF-e	C	F	SP	Db	Xi	Xx	19-23
B39E	CONVERT TO LOGICAL (extended BFP to 32)	CLFXBR	RRF-e	C	F	SP	Db	Xi	Xx	19-23
B3A0	CONVERT FROM LOGICAL (64 to short BFP)	CELGBR	RRF-e	F	SP	Db		Xx	19-20	
B3A1	CONVERT FROM LOGICAL (64 to long BFP)	CDLGBR	RRF-e	F	SP	Db		Xx	19-20	
B3A2	CONVERT FROM LOGICAL (64 to extended BFP)	CXLGBR	RRF-e	F	SP	Db			19-20	
B3A4	CONVERT FROM FIXED (64 to short BFP)	CEGBR	RRE	N			Db	Xx	19-18	
B3A4	CONVERT FROM FIXED (64 to short BFP)	CEGBRA	RRF-e	F	SP	Db		Xx	19-18	
B3A5	CONVERT FROM FIXED (64 to long BFP)	CDGBR	RRE	N			Db	Xx	19-18	
B3A5	CONVERT FROM FIXED (64 to long BFP)	CDGBRA	RRF-e	F	SP	Db		Xx	19-18	
B3A6	CONVERT FROM FIXED (64 to extended BFP)	CXGBR	RRE	N	SP	Db			19-18	
B3A6	CONVERT FROM FIXED (64 to extended BFP)	CXGBRA	RRF-e	F	SP	Db			19-18	
B3A8	CONVERT TO FIXED (short BFP to 64)	CGBR	RRF-e	C	N	SP	Db	Xi	Xx	19-21
B3A8	CONVERT TO FIXED (short BFP to 64)	CGBRA	RRF-e	C	F	SP	Db	Xi	Xx	19-21
B3A9	CONVERT TO FIXED (long BFP to 64)	CGDBR	RRF-e	C	N	SP	Db	Xi	Xx	19-21
B3A9	CONVERT TO FIXED (long BFP to 64)	CGDBRA	RRF-e	C	F	SP	Db	Xi	Xx	19-21
B3AA	CONVERT TO FIXED (extended BFP to 64)	CGXBR	RRF-e	C	N	SP	Db	Xi	Xx	19-21
B3AA	CONVERT TO FIXED (extended BFP to 64)	CGXBRA	RRF-e	C	F	SP	Db	Xi	Xx	19-21
B3AC	CONVERT TO LOGICAL (short BFP to 64)	CLGEBR	RRF-e	C	F	SP	Db	Xi	Xx	19-23
B3AD	CONVERT TO LOGICAL (long BFP to 64)	CLGDBR	RRF-e	C	F	SP	Db	Xi	Xx	19-23
B3AE	CONVERT TO LOGICAL (extended BFP to 64)	CLGXBR	RRF-e	C	F	SP	Db	Xi	Xx	19-23
B3B4	CONVERT FROM FIXED (32 to short HFP)	CEFR	RRE				Da		18-10	
B3B5	CONVERT FROM FIXED (32 to long HFP)	CDFR	RRE				Da		18-10	
B3B6	CONVERT FROM FIXED (32 to extended HFP)	CXFR	RRE		SP		Da		18-10	
B3B8	CONVERT TO FIXED (short HFP to 32)	CFER	RRF-e	C	SP		Da		18-11	
B3B9	CONVERT TO FIXED (long HFP to 32)	CFDR	RRF-e	C	SP		Da		18-11	
B3BA	CONVERT TO FIXED (extended HFP to 32)	CFXR	RRF-e	C	SP		Da		18-11	
B3C1	LOAD FPR FROM GR (64 to long)	LDGR	RRE	FG			Da		9-33	
B3C4	CONVERT FROM FIXED (64 to short HFP)	CEGR	RRE	N			Da		18-10	
B3C5	CONVERT FROM FIXED (64 to long HFP)	CDGR	RRE	N			Da		18-10	
B3C6	CONVERT FROM FIXED (64 to extended HFP)	CXGR	RRE	N	SP		Da		18-10	
B3C8	CONVERT TO FIXED (short HFP to 64)	CGER	RRF-e	C	N	SP		Da	18-11	
B3C9	CONVERT TO FIXED (long HFP to 64)	CGDR	RRF-e	C	N	SP		Da	18-11	
B3CA	CONVERT TO FIXED (extended HFP to 64)	CGXR	RRF-e	C	N	SP		Da	18-11	
B3CD	LOAD GR FROM FPR (long to 64)	LGDR	RRE	FG			Da		9-33	
B3D0	MULTIPLY (long DFP)	MDTR	RRF-a	TF			Dt	Xi	Xo Xu Xx	20-43
B3D0	MULTIPLY (long DFP)	MDTRA	RRF-a	F			Dt	Xi	Xo Xu Xx Xq	20-43
B3D1	DIVIDE (long DFP)	DDTR	RRF-a	TF			Dt	Xi	Xz Xo Xu Xx	20-32
B3D1	DIVIDE (long DFP)	DDTRA	RRF-a	F			Dt	Xi	Xz Xo Xu Xx Xq	20-32
B3D2	ADD (long DFP)	ADTR	RRF-a	C	TF		Dt	Xi	Xo Xu Xx	20-19
B3D2	ADD (long DFP)	ADTRA	RRF-a	C	F		Dt	Xi	Xo Xu Xx Xq	20-19
B3D3	SUBTRACT (long DFP)	SDTR	RRF-a	C	TF		Dt	Xi	Xo Xu Xx	20-51
B3D3	SUBTRACT (long DFP)	SDTRA	RRF-a	C	F		Dt	Xi	Xo Xu Xx Xq	20-51
B3D4	LOAD LENGTHENED (short to long DFP)	LDETR	RRF-d	TF			Dt	Xi		20-40
B3D5	LOAD ROUNDED (long to short DFP)	LEDTR	RRF-e	TF			Dt	Xi	Xo Xu Xx Xq	20-41
B3D6	LOAD AND TEST (long DFP)	LTDR	RRE	C	TF		Dt	Xi		20-36
B3D7	LOAD FP INTEGER (long DFP)	FIDTR	RRF-e	TF			Dt	Xi	Xx Xq	20-37

Figure B-3. Instructions Arranged by Operation Code (Part 8 of 18)

Op-code	Name	Mne-monic	Characteristics										Page	
B3D8	MULTIPLY (extended DFP)	MXTR	RRF-a	TF	SP	Dt	Xi	Xo	Xu	Xx			20-43	
B3D8	MULTIPLY (extended DFP)	MXTRA	RRF-a	F	SP	Dt	Xi	Xo	Xu	Xx	Xq		20-43	
B3D9	DIVIDE (extended DFP)	DXTR	RRF-a	TF	SP	Dt	Xi	Xz	Xo	Xu	Xx		20-32	
B3D9	DIVIDE (extended DFP)	DXTRA	RRF-a	F	SP	Dt	Xi	Xz	Xo	Xu	Xx	Xq	20-32	
B3DA	ADD (extended DFP)	AXTR	RRF-a	C TF	SP	Dt	Xi	Xo	Xu	Xx			20-19	
B3DA	ADD (extended DFP)	AXTRA	RRF-a	C F	SP	Dt	Xi	Xo	Xu	Xx	Xq		20-19	
B3DB	SUBTRACT (extended DFP)	SXTR	RRF-a	C TF	SP	Dt	Xi	Xo	Xu	Xx			20-51	
B3DB	SUBTRACT (extended DFP)	SXTRA	RRF-a	C F	SP	Dt	Xi	Xo	Xu	Xx	Xq		20-51	
B3DC	LOAD LENGTHENED (long to extended DFP)	LXDTR	RRF-d	TF	SP	Dt	Xi						20-40	
B3DD	LOAD ROUNDED (extended to long DFP)	LDXTR	RRF-e	TF	SP	Dt	Xi	Xo	Xu	Xx	Xq		20-41	
B3DE	LOAD AND TEST (extended DFP)	LTXTR	RRE	C TF	SP	Dt	Xi						20-36	
B3DF	LOAD FP INTEGER (extended DFP)	FIXTR	RRF-e	TF	SP	Dt	Xi			Xx	Xq		20-37	
B3E0	COMPARE AND SIGNAL (long DFP)	KDTR	RRE	C TF		Dt	Xi						20-22	
B3E1	CONVERT TO FIXED (long DFP to 64)	CGDTR	RRF-e	C TF		Dt	Xi			Xx			20-27	
B3E1	CONVERT TO FIXED (long DFP to 64)	CGDTRA	RRF-e	C F		Dt	Xi			Xx			20-27	
B3E2	CONVERT TO UNSIGNED PACKED (long DFP to 64)	CUDTR	RRE	TF		Dt							20-32	
B3E3	CONVERT TO SIGNED PACKED (long DFP to 64)	CSDTR	RRF-d	TF		Dt							20-31	
B3E4	COMPARE (long DFP)	CDTR	RRE	C TF		Dt	Xi						20-21	
B3E5	EXTRACT BIASED EXPONENT (long DFP to 64)	EEDTR	RRE	TF		Dt							20-34	
B3E7	EXTRACT SIGNIFICANCE (long DFP)	ESDTR	RRE	TF		Dt							20-35	
B3E8	COMPARE AND SIGNAL (extended DFP)	KXTR	RRE	C TF	SP	Dt	Xi						20-22	
B3E9	CONVERT TO FIXED (extended DFP to 64)	CGXTR	RRF-e	C TF	SP	Dt	Xi			Xx			20-27	
B3E9	CONVERT TO FIXED (extended DFP to 64)	CGXTRA	RRF-e	C F	SP	Dt	Xi			Xx			20-27	
B3EA	CONVERT TO UNSIGNED PACKED (extended DFP to 128)	CUXTR	RRE	TF	SP	Dt							20-32	
B3EB	CONVERT TO SIGNED PACKED (extended DFP to 128)	CSXTR	RRF-d	TF	SP	Dt							20-31	
B3EC	COMPARE (extended DFP)	CXTR	RRE	C TF	SP	Dt	Xi						20-21	
B3ED	EXTRACT BIASED EXPONENT (extended DFP to 64)	EEXTR	RRE	TF	SP	Dt							20-34	
B3EF	EXTRACT SIGNIFICANCE (extended DFP)	ESXTR	RRE	TF	SP	Dt							20-35	
B3F1	CONVERT FROM FIXED (64 to long DFP)	CDGTR	RRE	TF		Dt				Xx			20-24	
B3F1	CONVERT FROM FIXED (64 to long DFP)	CDGTRA	RRF-e	F		Dt				Xx	Xq		20-24	
B3F2	CONVERT FROM UNSIGNED PACKED (64 to long DFP)	CDUTR	RRE	TF		Dt	Dd						20-27	
B3F3	CONVERT FROM SIGNED PACKED (64 to long DFP)	CDSTR	RRE	TF		Dt	Dd						20-26	
B3F4	COMPARE BIASED EXPONENT (long DFP)	CEDTR	RRE	C TF		Dt							20-23	
B3F5	QUANTIZE (long DFP)	QADTR	RRF-b	TF		Dt	Xi			Xx	Xq		20-45	
B3F6	INSERT BIASED EXPONENT (64 to long DFP)	IEDTR	RRF-b	TF		Dt							20-35	
B3F7	REROUND (long DFP)	RRDTR	RRF-b	TF		Dt	Xi			Xx	Xq		20-47	
B3F9	CONVERT FROM FIXED (64 to extended DFP)	CXGTR	RRE	TF	SP	Dt							20-24	
B3F9	CONVERT FROM FIXED (64 to extended DFP)	CXGTRA	RRF-e	F	SP	Dt							20-24	
B3FA	CONVERT FROM UNSIGNED PACKED (128 to ext. DFP)	CXUTR	RRE	TF	SP	Dt	Dd						20-27	
B3FB	CONVERT FROM SIGNED PACKED (128 to extended DFP)	CXSTR	RRE	TF	SP	Dt	Dd						20-26	
B3FC	COMPARE BIASED EXPONENT (extended DFP)	CEXTR	RRE	C TF	SP	Dt							20-23	
B3FD	QUANTIZE (extended DFP)	QAXTR	RRF-b	TF	SP	Dt	Xi			Xx	Xq		20-45	
B3FE	INSERT BIASED EXPONENT (64 to extended DFP)	IEXTR	RRF-b	TF	SP	Dt							20-35	
B3FF	REROUND (extended DFP)	RRXTR	RRF-b	TF	SP	Dt	Xi			Xx	Xq		20-47	
B6	STORE CONTROL (32)	STCTL	RS-a		P A SP							ST	B ₂	10-126
B7	LOAD CONTROL (32)	LCTL	RS-a		P A SP								B ₂	10-44
B900	LOAD POSITIVE (64)	LPGR	RRE	C N						IF				7-230
B901	LOAD NEGATIVE (64)	LNGR	RRE	C N										7-227
B902	LOAD AND TEST (64)	LTGR	RRE	C N										7-219
B903	LOAD COMPLEMENT (64)	LCGR	RRE	C N						IF				7-221
B904	LOAD (64)	LGR	RRE	N										7-213

Figure B-3. Instructions Arranged by Operation Code (Part 9 of 18)

Op-code	Name	Mne-monic	Characteristics							Page
B905	LOAD USING REAL ADDRESS (64)	LURAG	RRE	N	P A ¹ SP					10-53
B906	LOAD BYTE (64)	LGBR	RRE	EI						7-220
B907	LOAD HALFWORD (64)	LGHR	RRE	EI						7-221
B908	ADD (64)	AGR	RRE	C N		IF				7-25
B909	SUBTRACT (64)	SGR	RRE	C N		IF				7-311
B90A	ADD LOGICAL (64)	ALGR	RRE	C N						7-28
B90B	SUBTRACT LOGICAL (64)	SLGR	RRE	C N						7-313
B90C	MULTIPLY SINGLE (64)	MSGR	RRE	N						7-249
B90D	DIVIDE SINGLE (64)	DSGR	RRE	N	SP	IK				7-204
B90E	EXTRACT STACKED REGISTERS (64)	EREGG	RRE	N	A ¹	SE		U ₁ U ₂		10-21
B90F	LOAD REVERSED (64)	LRVGR	RRE	N						7-231
B910	LOAD POSITIVE (64←32)	LPGFR	RRE	C N						7-230
B911	LOAD NEGATIVE (64←32)	LNGFR	RRE	C N						7-227
B912	LOAD AND TEST (64←32)	LTGFR	RRE	C N						7-220
B913	LOAD COMPLEMENT (64←32)	LCGFR	RRE	C N						7-221
B914	LOAD (64←32)	LGFR	RRE	N						7-213
B916	LOAD LOGICAL (64←32)	LLGFR	RRE	N						7-223
B917	LOAD LOGICAL THIRTY ONE BITS	LLGTR	RRE	N						7-226
B918	ADD (64←32)	AGFR	RRE	C N		IF				7-25
B919	SUBTRACT (64←32)	SGFR	RRE	C N		IF				7-311
B91A	ADD LOGICAL (64←32)	ALGFR	RRE	C N						7-28
B91B	SUBTRACT LOGICAL (64←32)	SLGFR	RRE	C N						7-313
B91C	MULTIPLY SINGLE (64←32)	MSGFR	RRE	N						7-249
B91D	DIVIDE SINGLE (64←32)	DSGFR	RRE	N	SP	IK				7-204
B91E	COMPUTE MESSAGE AUTHENTICATION CODE	KMAC	RRE	C MS	A SP	IC	GM I1	ST	R ₂	7-171
B91F	LOAD REVERSED (32)	LRVR	RRE	N3						7-230
B920	COMPARE (64)	CGR	RRE	C N						7-112
B921	COMPARE LOGICAL (64)	CLGR	RRE	C N						7-130
B925	STORE USING REAL ADDRESS (64)	STURG	RRE	N	P A ¹ SP			SU		10-150
B926	LOAD BYTE (32)	LBR	RRE	EI						7-220
B927	LOAD HALFWORD (32)	LHR	RRE	EI						7-221
B928	PERFORM CRYPTOGRAPHIC KEY MGMT. OPERATIONS	PCKMO	RRE	M3	P A SP		GM	ST		10-66
B92A	CIPHER MESSAGE WITH CFB	KMF	RRE	C M4	A SP	IC	GM I1	ST	R ₁ R ₂	7-71
B92B	CIPHER MESSAGE WITH OFB	KMO	RRE	C M4	A SP	IC	GM I1	ST	R ₁ R ₂	7-98
B92C	PERFORM CRYPTOGRAPHIC COMPUTATION	PCC	RRE	C M4	A SP	IC	GM I1	ST		7-255
B92D	CIPHER MESSAGE WITH COUNTER	KMCTR	RRF-b	C M4	A SP	IC	GM I1	ST	R ₁ , R ₂ , R ₃	7-86
B92E	CIPHER MESSAGE	KM	RRE	C MS	A SP	IC	GM I1	ST	R ₁ R ₂	7-46
B92F	CIPHER MESSAGE WITH CHAINING	KMC	RRE	C MS	A SP	IC	GM I1	ST	R ₁ R ₂	7-46
B930	COMPARE (64←32)	CGFR	RRE	C N						7-112
B931	COMPARE LOGICAL (64←32)	CLGFR	RRE	C N						7-130
B93E	COMPUTE INTERMEDIATE MESSAGE DIGEST	KIMD	RRE	C MS	A SP	IC	GM I1	ST	R ₂	7-159
B93F	COMPUTE LAST MESSAGE DIGEST	KLMD	RRE	C MS	A SP	IC	GM I1	ST	R ₂	7-159
B941	CONVERT TO FIXED (long DFP to 32)	CFDTR	RRF-e	C F		Dt Xi	Xx			20-27
B942	CONVERT TO LOGICAL (long DFP to 64)	CLGDTR	RRF-e	C F		Dt Xi	Xx			20-29
B943	CONVERT TO LOGICAL (long DFP to 32)	CLFDTR	RRF-e	C F		Dt Xi	Xx			20-29
B946	BRANCH ON COUNT (64)	BCTGR	RRE	N				B		7-37
B949	CONVERT TO FIXED (extended DFP to 32)	CFXTR	RRF-e	C F	SP	Dt Xi	Xx			20-27
B94A	CONVERT TO LOGICAL (extended DFP to 64)	CLGXTR	RRF-e	C F	SP	Dt Xi	Xx			20-29
B94B	CONVERT TO LOGICAL (extended DFP to 32)	CLFXTR	RRF-e	C F	SP	Dt Xi	Xx			20-29
B951	CONVERT FROM FIXED (32 to long DFP)	CDFTR	RRE	F		Dt				20-24
B952	CONVERT FROM LOGICAL (64 to long DFP)	CDLGTR	RRF-e	F		Dt	Xx Xq			20-25
B953	CONVERT FROM LOGICAL (32 to long DFP)	CDLFTR	RRF-e	F		Dt				20-25
B959	CONVERT FROM FIXED (32 to extended DFP)	CXFTR	RRE	F	SP	Dt				20-24
B95A	CONVERT FROM LOGICAL (64 to extended DFP)	CXLGTR	RRF-e	F	SP	Dt				20-25

Figure B-3. Instructions Arranged by Operation Code (Part 10 of 18)

Op-code	Name	Mnemonic	Characteristics										Page
B95B	CONVERT FROM LOGICAL (32 to extended DFP)	CXLFT	RRF-e	F		SP	Dt						20-25
B960	COMPARE AND TRAP (64)	CGRT	RRF-c	GE			Dc						7-127
B961	COMPARE LOGICAL AND TRAP (64)	CLGRT	RRF-c	GE			Dc						7-133
B972	COMPARE AND TRAP (32)	CRT	RRF-c	GE			Dc						7-127
B973	COMPARE LOGICAL AND TRAP (32)	CLRT	RRF-c	GE			Dc						7-133
B980	AND (64)	NGR	RRE	C N									7-31
B981	OR (64)	OGR	RRE	C N									7-250
B982	EXCLUSIVE OR (64)	XGR	RRE	C N									7-205
B983	FIND LEFTMOST ONE	FLOGR	RRE	C EI		SP							7-211
B984	LOAD LOGICAL CHARACTER (64)	LLGCR	RRE	EI									7-223
B985	LOAD LOGICAL HALFWORD (64)	LLGHR	RRE	EI									7-224
B986	MULTIPLY LOGICAL (128←64)	MLGR	RRE	N		SP							7-249
B987	DIVIDE LOGICAL (64←128)	DLGR	RRE	N		SP	IK						7-204
B988	ADD LOGICAL WITH CARRY (64)	ALCGR	RRE	C N									7-29
B989	SUBTRACT LOGICAL WITH BORROW (64)	SLBGR	RRE	C N									7-314
B98A	COMPARE AND SWAP AND PURGE	CSPG	RRE	C DE	P A ¹	SP	\$		ST		R ₂		10-17
B98D	EXTRACT PSW	EPSW	RRE	N3									7-210
B98E	INVALIDATE DAT TABLE ENTRY	IDTE	RRF-b	DE	P A ¹	SP	\$						10-28
B990	TRANSLATE TWO TO TWO	TRTT	RRF-c	C E2	A	SP	IC	GM	ST		RM R ₂		7-327
B991	TRANSLATE TWO TO ONE	TRTO	RRF-c	C E2	A	SP	IC	GM	ST		RM R ₂		7-327
B992	TRANSLATE ONE TO TWO	TROT	RRF-c	C E2	A	SP	IC	GM	ST		RM R ₂		7-327
B993	TRANSLATE ONE TO ONE	TROO	RRF-c	C E2	A	SP	IC	GM	ST		RM R ₂		7-327
B994	LOAD LOGICAL CHARACTER (32)	LLCR	RRE	EI									7-224
B995	LOAD LOGICAL HALFWORD (32)	LLHR	RRE	EI									7-224
B996	MULTIPLY LOGICAL (64←32)	MLR	RRE	N3		SP							7-248
B997	DIVIDE LOGICAL (32←64)	DLR	RRE	N3		SP	IK						7-204
B998	ADD LOGICAL WITH CARRY (32)	ALCR	RRE	C N3									7-29
B999	SUBTRACT LOGICAL WITH BORROW (32)	SLBR	RRE	C N3									7-314
B99A	EXTRACT PRIMARY ASN AND INSTANCE	EPAIR	RRE	RA	Q		SO						10-20
B99B	EXTRACT SECONDARY ASN AND INSTANCE	ESAIR	RRE	RA	Q		SO						10-21
B99D	EXTRACT AND SET EXTENDED AUTHORITY	ESEA	RRE	N	P								10-20
B99E	PROGRAM TRANSFER WITH INSTANCE	PTI	RRE	RA	Q A ¹	SP	Z ⁶ T c		B				10-98
B99F	SET SECONDARY ASN WITH INSTANCE	SSAIR	RRE	RA	A ¹		Z ⁷ T c						10-115
B9A2	PERFORM TOPOLOGY FUNCTION	PTF	RRE	C CT	P	SP							10-80
B9AA	LOAD PAGE TABLE ENTRY ADDRESS	LPTEA	RRF-b	C D2	P A ¹	SP	SO				R ₂		10-44
B9AE	RESET REFERENCE BITS MULTIPLE	RRBM	RRE	RB	P A ¹								10-108
B9AF	PERFORM FRAME MANAGEMENT FUNCTION	PFMF	RRE	ED	P A ¹	SP	II	c ³					10-70
B9B0	CONVERT UTF-8 TO UTF-32	CU14	RRF-c	C E3	A	SP	IC		ST		R ₁ R ₂		7-199
B9B1	CONVERT UTF-16 TO UTF-32	CU24	RRF-c	C E3	A	SP	IC		ST		R ₁ R ₂		7-183
B9B2	CONVERT UTF-32 TO UTF-8	CU41	RRE	C E3	A	SP	IC		ST		R ₁ R ₂		7-192
B9B3	CONVERT UTF-32 TO UTF-16	CU42	RRE	C E3	A	SP	IC		ST		R ₁ R ₂		7-189
B9BD	TRANSLATE AND TEST REVERSE EXTENDED	TRTRE	RRF-c	C PE	A	SP	IC		ST		RM		7-319
B9BE	SEARCH STRING UNICODE	SRSTU	RRE	C E3	A	SP	IC	G0			R ₁ R ₂		7-293
B9BF	TRANSLATE AND TEST EXTENDED	TRTE	RRF-c	C PE	A	SP	IC		ST		RM		7-319
B9C8	ADD HIGH (32)	AHHHR	RRF-a	C HW			IF						7-27
B9C9	SUBTRACT HIGH (32)	SHHHR	RRF-a	C HW			IF						7-312
B9CA	ADD LOGICAL HIGH (32)	ALHHHR	RRF-a	C HW									7-29
B9CB	SUBTRACT LOGICAL HIGH (32)	SLHHHR	RRF-a	C HW									7-314
B9CD	COMPARE HIGH (32)	CHHR	RRE	C HW									7-129
B9CF	COMPARE LOGICAL HIGH (32)	CLHHR	RRE	C HW									7-135
B9D8	ADD HIGH (32)	AHHLR	RRF-a	C HW			IF						7-27
B9D9	SUBTRACT HIGH (32)	SHHLR	RRF-a	C HW			IF						7-312
B9DA	ADD LOGICAL HIGH (32)	ALHHLR	RRF-a	C HW									7-29
B9DB	SUBTRACT LOGICAL HIGH (32)	SLHHLR	RRF-a	C HW									7-314
B9DD	COMPARE HIGH (32)	CHLR	RRE	C HW									7-129

Figure B-3. Instructions Arranged by Operation Code (Part 11 of 18)

Op-code	Name	Mne-monic	Characteristics						Page
B9DF	COMPARE LOGICAL HIGH (32)	CLHLR	RRE C HW						7-135
B9E1	POPULATION COUNT	POPCNT	RRE C PK						7-284
B9E2	LOAD ON CONDITION (64)	LOCGR	RRF-c CL	A					7-228
B9E4	AND (64)	NGRK	RRF-a C DO						7-31
B9E6	OR (64)	OGRK	RRF-a C DO						7-250
B9E7	EXCLUSIVE OR (64)	XGRK	RRF-a C DO						7-205
B9E8	ADD (64)	AGRK	RRF-a C DO		IF				7-25
B9E9	SUBTRACT (64)	SGRK	RRF-a C DO		IF				7-311
B9EA	ADD LOGICAL (64)	ALGRK	RRF-a C DO						7-28
B9EB	SUBTRACT LOGICAL (64)	SLGRK	RRF-a C DO						7-313
B9F2	LOAD ON CONDITION (32)	LOCR	RRF-c CL	A					7-228
B9F4	AND (32)	NRK	RRF-a C DO						7-31
B9F6	OR (32)	ORK	RRF-a C DO						7-250
B9F7	EXCLUSIVE OR (32)	XRK	RRF-a C DO						7-205
B9F8	ADD (32)	ARK	RRF-a C DO		IF				7-25
B9F9	SUBTRACT (32)	SRK	RRF-a C DO		IF				7-311
B9FA	ADD LOGICAL (32)	ALRK	RRF-a C DO						7-28
B9FB	SUBTRACT LOGICAL (32)	SLRK	RRF-a C DO						7-313
BA	COMPARE AND SWAP (32)	CS	RS-a C	A SP	\$		ST	B ₂	7-122
BB	COMPARE DOUBLE AND SWAP (32)	CDS	RS-a C	A SP	\$		ST	B ₂	7-122
BD	COMPARE LOGICAL CHAR. UNDER MASK (low)	CLM	RS-b C	A				B ₂	7-134
BE	STORE CHARACTERS UNDER MASK (low)	STCM	RS-b	A			ST	B ₂	7-303
BF	INSERT CHARACTERS UNDER MASK (low)	ICM	RS-b C	A				B ₂	7-212
C00	LOAD ADDRESS RELATIVE LONG	LARL	RIL-b N3						7-216
C01	LOAD IMMEDIATE (64←32)	LGFI	RIL-a EI						7-214
C04	BRANCH RELATIVE ON CONDITION LONG	BRCL	RIL-c N3				B		7-40
C05	BRANCH RELATIVE AND SAVE LONG	BRASL	RIL-b N3				B		7-39
C06	EXCLUSIVE OR IMMEDIATE (high)	XIHF	RIL-a C EI						7-206
C07	EXCLUSIVE OR IMMEDIATE (low)	XILF	RIL-a C EI						7-206
C08	INSERT IMMEDIATE (high)	IHF	RIL-a EI						7-212
C09	INSERT IMMEDIATE (low)	IILF	RIL-a EI						7-213
C0A	AND IMMEDIATE (high)	NIHF	RIL-a C EI						7-32
C0B	AND IMMEDIATE (low)	NILF	RIL-a C EI						7-33
C0C	OR IMMEDIATE (high)	OIHF	RIL-a C EI						7-251
C0D	OR IMMEDIATE (low)	OILF	RIL-a C EI						7-251
C0E	LOAD LOGICAL IMMEDIATE (high)	LLIHF	RIL-a EI						7-225
C0F	LOAD LOGICAL IMMEDIATE (low)	LLILF	RIL-a EI						7-225
C20	MULTIPLY SINGLE IMMEDIATE (64←32)	MSGFI	RIL-a GE						7-250
C21	MULTIPLY SINGLE IMMEDIATE (32)	MSFI	RIL-a GE						7-250
C24	SUBTRACT LOGICAL IMMEDIATE (64←32)	SLGFI	RIL-a C EI						7-313
C25	SUBTRACT LOGICAL IMMEDIATE (32)	SLFI	RIL-a C EI						7-313
C28	ADD IMMEDIATE (64←32)	AGFI	RIL-a C EI		IF				7-25
C29	ADD IMMEDIATE (32)	AFI	RIL-a C EI		IF				7-25
C2A	ADD LOGICAL IMMEDIATE (64←32)	ALGFI	RIL-a C EI						7-28
C2B	ADD LOGICAL IMMEDIATE (32)	ALFI	RIL-a C EI						7-28
C2C	COMPARE IMMEDIATE (64←32)	CGFI	RIL-a C EI						7-112
C2D	COMPARE IMMEDIATE (32)	CFI	RIL-a C EI						7-112
C2E	COMPARE LOGICAL IMMEDIATE (64←32)	CLGFI	RIL-a C EI						7-130
C2F	COMPARE LOGICAL IMMEDIATE (32)	CLFI	RIL-a C EI						7-130
C42	LOAD LOGICAL HALFWORD RELATIVE LONG (32←16)	LLHRL	RIL-b GE	A					7-224
C44	LOAD HALFWORD RELATIVE LONG (64←16)	LGHRL	RIL-b GE	A					7-222
C45	LOAD HALFWORD RELATIVE LONG (32←16)	LHRL	RIL-b GE	A					7-222
C46	LOAD LOGICAL HALFWORD RELATIVE LONG (64←16)	LLGHRL	RIL-b GE	A					7-224

Figure B-3. Instructions Arranged by Operation Code (Part 12 of 18)

Op-code	Name	Mne-monic	Characteristics						Page
C47	STORE HALFWORD RELATIVE LONG	STHRL	RIL-b	GE	A		ST		7-307
C48	LOAD RELATIVE LONG (64)	LGRL	RIL-b	GE	A SP				7-214
C4B	STORE RELATIVE LONG (64)	STGRL	RIL-b	GE	A SP		ST		7-301
C4C	LOAD RELATIVE LONG (64←32)	LGFRLL	RIL-b	GE	A SP				7-214
C4D	LOAD RELATIVE LONG (32)	LRL	RIL-b	GE	A SP				7-214
C4E	LOAD LOGICAL RELATIVE LONG (64←32)	LLGFRLL	RIL-b	GE	A SP				7-223
C4F	STORE RELATIVE LONG (32)	STRL	RIL-b	GE	A SP		ST		7-301
C60	EXECUTE RELATIVE LONG	EXRL	RIL-b	XX	AI	EX			7-207
C62	PREFETCH DATA RELATIVE LONG	PFDRLL	RIL-c	GE					7-285
C64	COMPARE HALFWORD RELATIVE LONG (64←16)	CGHRL	RIL-b	C GE	A				7-128
C65	COMPARE HALFWORD RELATIVE LONG (32←16)	CHRL	RIL-b	C GE	A				7-128
C66	COMPARE LOGICAL RELATIVE LONG (64←16)	CLGHRL	RIL-b	C GE	A				7-131
C67	COMPARE LOGICAL RELATIVE LONG (32←16)	CLHRL	RIL-b	C GE	A				7-131
C68	COMPARE RELATIVE LONG (64)	CGRL	RIL-b	C GE	A SP				7-112
C6A	COMPARE LOGICAL RELATIVE LONG (64)	CLGRL	RIL-b	C GE	A SP				7-131
C6C	COMPARE RELATIVE LONG (64←32)	CGFRLL	RIL-b	C GE	A SP				7-112
C6D	COMPARE RELATIVE LONG (32)	CRL	RIL-b	C GE	A SP				7-112
C6E	COMPARE LOGICAL RELATIVE LONG (64←32)	CLGFRLL	RIL-b	C GE	A SP				7-131
C6F	COMPARE LOGICAL RELATIVE LONG (32)	CLRL	RIL-b	C GE	A SP				7-130
C80	MOVE WITH OPTIONAL SPECIFICATIONS	MVCOS	SSF	C MO	Q A	SO	G0	ST B ₁ B ₂	10-60
C81	EXTRACT CPU TIME	ECTG	SSF	ET	A		GM	R ₃ B ₁ B ₂	7-210
C82	COMPARE AND SWAP AND STORE	CSST	SSF	C CS	A SP	\$	GM	ST B ₁ B ₂	7-124
C84	LOAD PAIR DISJOINT (32)	LPD	SSF	C IA	A SP			B ₁ B ₂	7-229
C85	LOAD PAIR DISJOINT (64)	LPDG	SSF	C IA	A SP			B ₁ B ₂	7-229
CC6	BRANCH RELATIVE ON COUNT HIGH (32)	BRCTH	RIL-b	HW				B	7-41
CC8	ADD IMMEDIATE HIGH (32)	AIH	RIL-a	C HW		IF			7-25
CCA	ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	ALSIH	RIL-a	C HW					7-31
CCB	ADD LOGICAL WITH SIGNED IMMEDIATE HIGH (32)	ALSIHN	RIL-a	HW					7-31
CCD	COMPARE IMMEDIATE HIGH (32)	CIH	RIL-a	C HW					7-129
CCF	COMPARE LOGICAL IMMEDIATE HIGH (32)	CLIH	RIL-a	C HW					7-135
D0	TRANSLATE AND TEST REVERSE	TRTR	SS-a	C E3	A		GM	B ₁ B ₂	7-323
D1	MOVE NUMERICS	MVN	SS-a		A			ST B ₁ B ₂	7-244
D2	MOVE (character)	MVC	SS-a		A			ST B ₁ B ₂	7-232
D3	MOVE ZONES	MVZ	SS-a		A			ST B ₁ B ₂	7-246
D4	AND (character)	NC	SS-a	C	A			ST B ₁ B ₂	7-32
D5	COMPARE LOGICAL (character)	CLC	SS-a	C	A			B ₁ B ₂	7-130
D6	OR (character)	OC	SS-a	C	A			ST B ₁ B ₂	7-251
D7	EXCLUSIVE OR (character)	XC	SS-a	C	A			ST B ₁ B ₂	7-206
D9	MOVE WITH KEY	MVCK	SS-d	C	Q A			ST B ₁ B ₂	10-59
DA	MOVE TO PRIMARY	MVCP	SS-d	C	Q A	SO	¢	ST	10-56
DB	MOVE TO SECONDARY	MVCS	SS-d	C	Q A	SO	¢	ST	10-56
DC	TRANSLATE	TR	SS-a		A			ST B ₁ B ₂	7-317
DD	TRANSLATE AND TEST	TRT	SS-a	C	A		GM	B ₁ B ₂	7-318
DE	EDIT	ED	SS-a	C	A	Dd		ST B ₁ B ₂	8-7
DF	EDIT AND MARK	EDMK	SS-a	C	A	Dd	G1	ST B ₁ B ₂	8-10
E1	PACK UNICODE	PKU	SS-f	E2	A SP			ST B ₁ B ₂	7-254
E2	UNPACK UNICODE	UNPKU	SS-a	C E2	A SP			ST B ₁ B ₂	7-333
E302	LOAD AND TEST (64)	LTG	RXY-a	C EI	A			B ₂	7-220
E303	LOAD REAL ADDRESS (64)	LRAG	RXY-a	C N	P A ¹			BP	10-49
E304	LOAD (64)	LG	RXY-a	N	A			B ₂	7-213
E306	CONVERT TO BINARY (32)	CVBY	RXY-a	LD	A		Dd IK	B ₂	7-181
E308	ADD (64)	AG	RXY-a	C N	A		IF	B ₂	7-25
E309	SUBTRACT (64)	SG	RXY-a	C N	A		IF	B ₂	7-311
E30A	ADD LOGICAL (64)	ALG	RXY-a	C N	A			B ₂	7-28
E30B	SUBTRACT LOGICAL (64)	SLG	RXY-a	C N	A			B ₂	7-313

Figure B-3. Instructions Arranged by Operation Code (Part 13 of 18)

Op-code	Name	Mne-monic	Characteristics						Page
E30C	MULTIPLY SINGLE (64)	MSG	RXY-a	N	A			B ₂	7-249
E30D	DIVIDE SINGLE (64)	DSG	RXY-a	N	A SP	IK		B ₂	7-205
E30E	CONVERT TO BINARY (64)	CVBG	RXY-a	N	A	Dd IK		B ₂	7-181
E30F	LOAD REVERSED (64)	LRVG	RXY-a	N	A			B ₂	7-231
E312	LOAD AND TEST (32)	LT	RXY-a	C	EI	A		B ₂	7-220
E313	LOAD REAL ADDRESS (32)	LRAY	RXY-a	C	LD P A ¹	SO		BP	10-49
E314	LOAD (64←32)	LGF	RXY-a	N	A			B ₂	7-213
E315	LOAD HALFWORD (64)	LGH	RXY-a	N	A			B ₂	7-222
E316	LOAD LOGICAL (64←32)	LLGF	RXY-a	N	A			B ₂	7-223
E317	LOAD LOGICAL THIRTY ONE BITS	LLGT	RXY-a	N	A			B ₂	7-226
E318	ADD (64←32)	AGF	RXY-a	C	N	A	IF	B ₂	7-25
E319	SUBTRACT (64←32)	SGF	RXY-a	C	N	A	IF	B ₂	7-311
E31A	ADD LOGICAL (64←32)	ALGF	RXY-a	C	N	A		B ₂	7-28
E31B	SUBTRACT LOGICAL (64←32)	SLGF	RXY-a	C	N	A		B ₂	7-313
E31C	MULTIPLY SINGLE (64←32)	MSGF	RXY-a	N	A			B ₂	7-249
E31D	DIVIDE SINGLE (64←32)	DSGF	RXY-a	N	A SP	IK		B ₂	7-205
E31E	LOAD REVERSED (32)	LRV	RXY-a	N3	A			B ₂	7-231
E31F	LOAD REVERSED (16)	LRVH	RXY-a	N3	A			B ₂	7-231
E320	COMPARE (64)	CG	RXY-a	C	N	A		B ₂	7-112
E321	COMPARE LOGICAL (64)	CLG	RXY-a	C	N	A		B ₂	7-130
E324	STORE (64)	STG	RXY-a	N	A		ST	B ₂	7-301
E326	CONVERT TO DECIMAL (32)	CVDY	RXY-a	LD	A		ST	B ₂	7-182
E32E	CONVERT TO DECIMAL (64)	CVDG	RXY-a	N	A		ST	B ₂	7-182
E32F	STORE REVERSED (64)	STRVG	RXY-a	N	A		ST	B ₂	7-311
E330	COMPARE (64←32)	CGF	RXY-a	C	N	A		B ₂	7-112
E331	COMPARE LOGICAL (64←32)	CLGF	RXY-a	C	N	A		B ₂	7-130
E332	LOAD AND TEST (64←32)	LTGF	RXY-a	C	GE	A		B ₂	7-220
E334	COMPARE HALFWORD (64←16)	CGH	RXY-a	C	GE	A		B ₂	7-128
E336	PREFETCH DATA	PFD	RXY-b	GE				B ₂	7-285
E33E	STORE REVERSED (32)	STRV	RXY-a	N3	A		ST	B ₂	7-310
E33F	STORE REVERSED (16)	STRVH	RXY-a	N3	A		ST	B ₂	7-310
E346	BRANCH ON COUNT (64)	BCTG	RXY-a	N			B		7-37
E350	STORE (32)	STY	RXY-a	LD	A		ST	B ₂	7-301
E351	MULTIPLY SINGLE (32)	MSY	RXY-a	LD	A			B ₂	7-249
E354	AND (32)	NY	RXY-a	C	LD	A		B ₂	7-32
E355	COMPARE LOGICAL (32)	CLY	RXY-a	C	LD	A		B ₂	7-130
E356	OR (32)	OY	RXY-a	C	LD	A		B ₂	7-250
E357	EXCLUSIVE OR (32)	XY	RXY-a	C	LD	A		B ₂	7-205
E358	LOAD (32)	LY	RXY-a	LD	A			B ₂	7-213
E359	COMPARE (32)	CY	RXY-a	C	LD	A		B ₂	7-112
E35A	ADD (32)	AY	RXY-a	C	LD	A	IF	B ₂	7-25
E35B	SUBTRACT (32)	SY	RXY-a	C	LD	A	IF	B ₂	7-311
E35C	MULTIPLY (64←32)	MFY	RXY-a	GE	A SP			B ₂	7-247
E35E	ADD LOGICAL (32)	ALY	RXY-a	C	LD	A		B ₂	7-28
E35F	SUBTRACT LOGICAL (32)	SLY	RXY-a	C	LD	A		B ₂	7-313
E370	STORE HALFWORD	STHY	RXY-a	LD	A		ST	B ₂	7-307
E371	LOAD ADDRESS	LAY	RXY-a	LD					7-215
E372	STORE CHARACTER	STCY	RXY-a	LD	A		ST	B ₂	7-302
E373	INSERT CHARACTER	ICY	RXY-a	LD	A			B ₂	7-211
E375	LOAD ADDRESS EXTENDED	LAEY	RXY-a	GE			U ₁	BP	7-215
E376	LOAD BYTE (32)	LB	RXY-a	LD	A				7-220
E377	LOAD BYTE (64)	LGB	RXY-a	LD	A				7-220
E378	LOAD HALFWORD (32)	LHY	RXY-a	LD	A			B ₂	7-221
E379	COMPARE HALFWORD (32←16)	CHY	RXY-a	C	LD	A		B ₂	7-128
E37A	ADD HALFWORD	AHY	RXY-a	C	LD	A	IF	B ₂	7-27

Figure B-3. Instructions Arranged by Operation Code (Part 14 of 18)

Op-code	Name	Mne-monic	Characteristics						Page
E37B	SUBTRACT HALFWORD	SHY	RXY-a	C	LD	A	IF		B ₂ 7-312
E37C	MULTIPLY HALFWORD (32)	MHY	RXY-a		GE	A			B ₂ 7-248
E380	AND (64)	NG	RXY-a	C	N	A			B ₂ 7-32
E381	OR (64)	OG	RXY-a	C	N	A			B ₂ 7-250
E382	EXCLUSIVE OR (64)	XG	RXY-a	C	N	A			B ₂ 7-205
E386	MULTIPLY LOGICAL (128←64)	MLG	RXY-a		N	A SP			B ₂ 7-249
E387	DIVIDE LOGICAL (64←128)	DLG	RXY-a		N	A SP	IK		B ₂ 7-204
E388	ADD LOGICAL WITH CARRY (64)	ALCG	RXY-a	C	N	A			B ₂ 7-29
E389	SUBTRACT LOGICAL WITH BORROW (64)	SLBG	RXY-a	C	N	A			B ₂ 7-315
E38E	STORE PAIR TO QUADWORD	STPQ	RXY-a		N	A SP		ST	B ₂ 7-310
E38F	LOAD PAIR FROM QUADWORD	LPQ	RXY-a		N	A SP			B ₂ 7-230
E390	LOAD LOGICAL CHARACTER (64)	LLGC	RXY-a		N	A			B ₂ 7-224
E391	LOAD LOGICAL HALFWORD (64)	LLGH	RXY-a		N	A			B ₂ 7-224
E394	LOAD LOGICAL CHARACTER (32)	LLC	RXY-a		EI	A			B ₂ 7-224
E395	LOAD LOGICAL HALFWORD (32)	LLH	RXY-a		EI	A			B ₂ 7-224
E396	MULTIPLY LOGICAL (64←32)	ML	RXY-a		N3	A SP			B ₂ 7-249
E397	DIVIDE LOGICAL (32←64)	DL	RXY-a		N3	A SP	IK		B ₂ 7-204
E398	ADD LOGICAL WITH CARRY (32)	ALC	RXY-a	C	N3	A			B ₂ 7-29
E399	SUBTRACT LOGICAL WITH BORROW (32)	SLB	RXY-a	C	N3	A			B ₂ 7-315
E3C0	LOAD BYTE HIGH (32←8)	LBH	RXY-a		HW	A			B ₂ 7-221
E3C2	LOAD LOGICAL CHARACTER HIGH (32←8)	LLCH	RXY-a		HW	A			B ₂ 7-224
E3C3	STORE CHARACTER HIGH (8)	STCH	RXY-a		HW	A		ST	B ₂ 7-303
E3C4	LOAD HALFWORD HIGH (32←16)	LHH	RXY-a		HW	A			B ₂ 7-222
E3C6	LOAD LOGICAL HALFWORD HIGH (32←16)	LLHH	RXY-a		HW	A			B ₂ 7-225
E3C7	STORE HALFWORD HIGH (16)	STHH	RXY-a		HW	A		ST	B ₂ 7-308
E3CA	LOAD HIGH (32)	LFH	RXY-a		HW	A			B ₂ 7-223
E3CB	STORE HIGH (32)	STFH	RXY-a		HW	A		ST	B ₂ 7-308
E3CD	COMPARE HIGH (32)	CHF	RXY-a	C	HW	A			B ₂ 7-129
E3CF	COMPARE LOGICAL HIGH (32)	CLHF	RXY-a	C	HW	A			B ₂ 7-135
E500	LOAD ADDRESS SPACE PARAMETERS	LASP	SSE	C		P A ¹ SP	SO		B ₁ 10-34
E501	TEST PROTECTION	TPROT	SSE	C		P A ¹			B ₁ 10-155
E502	STORE REAL ADDRESS	STRAG	SSE		N	P A ¹ SP		ST	B ₁ BP 10-129
E50E	MOVE WITH SOURCE KEY	MVCSK	SSE			Q A	GM	ST	B ₁ B ₂ 10-63
E50F	MOVE WITH DESTINATION KEY	MVCDK	SSE			Q A	GM	ST	B ₁ B ₂ 10-58
E544	MOVE (16←16)	MVHHI	SIL		GE	A		ST	B ₁ 7-232
E548	MOVE (64←16)	MVGHI	SIL		GE	A		ST	B ₁ 7-233
E54C	MOVE (32←16)	MVHI	SIL		GE	A		ST	B ₁ 7-233
E554	COMPARE HALFWORD IMMEDIATE (16)	CHHSI	SIL	C	GE	A			B ₁ 7-128
E555	COMPARE LOGICAL IMMEDIATE (16)	CLHHSI	SIL	C	GE	A			B ₁ 7-130
E558	COMPARE HALFWORD IMMEDIATE (64←16)	CGHSI	SIL	C	GE	A			B ₁ 7-128
E559	COMPARE LOGICAL IMMEDIATE (64←16)	CLGHSI	SIL	C	GE	A			B ₁ 7-130
E55C	COMPARE HALFWORD IMMEDIATE (32←16)	CHSI	SIL	C	GE	A			B ₁ 7-128
E55D	COMPARE LOGICAL IMMEDIATE (32←16)	CLFHSI	SIL	C	GE	A			B ₁ 7-130
E8	MOVE INVERSE	MVCIN	SS-a			A		ST	B ₁ B ₂ 7-233
E9	PACK ASCII	PKA	SS-f		E2	A SP		ST	B ₁ B ₂ 7-253
EA	UNPACK ASCII	UNPKA	SS-a	C	E2	A SP		ST	B ₁ B ₂ 7-332
EB04	LOAD MULTIPLE (64)	LMG	RSY-a		N	A			B ₂ 7-226
EB0A	SHIFT RIGHT SINGLE (64)	SRAG	RSY-a	C	N		IF		7-300
EB0B	SHIFT LEFT SINGLE (64)	SLAG	RSY-a	C	N				7-297
EB0C	SHIFT RIGHT SINGLE LOGICAL (64)	SRLG	RSY-a		N				7-301
EB0D	SHIFT LEFT SINGLE LOGICAL (64)	SLLG	RSY-a		N				7-298
EB0F	TRACE (64)	TRACG	RSY-a		N	P A SP	T ¢		B ₂ 10-157
EB14	COMPARE AND SWAP (32)	CSY	RSY-a	C	LD	A SP	\$	ST	B ₂ 7-122
EB1C	ROTATE LEFT SINGLE LOGICAL (64)	RLLG	RSY-a		N				7-286
EB1D	ROTATE LEFT SINGLE LOGICAL (32)	RLL	RSY-a		N3				7-286

Figure B-3. Instructions Arranged by Operation Code (Part 15 of 18)

Op-code	Name	Mne-monic	Characteristics						Page
EB20	COMPARE LOGICAL CHAR. UNDER MASK (high)	CLMH	RSY-b	C	N	A		B ₂	7-134
EB21	COMPARE LOGICAL CHAR. UNDER MASK (low)	CLMY	RSY-b	C	LD	A		B ₂	7-134
EB24	STORE MULTIPLE (64)	STMG	RSY-a	N	A	A	ST	B ₂	7-308
EB25	STORE CONTROL (64)	STCTG	RSY-a	N	P	A SP	ST	B ₂	10-126
EB26	STORE MULTIPLE HIGH	STMH	RSY-a	N	A	A	ST	B ₂	7-309
EB2C	STORE CHARACTERS UNDER MASK (high)	STCMH	RSY-b	N	A	A	ST	B ₂	7-303
EB2D	STORE CHARACTERS UNDER MASK (low)	STCMY	RSY-b	LD	A	A	ST	B ₂	7-303
EB2F	LOAD CONTROL (64)	LCTLG	RSY-a	N	P	A SP		B ₂	10-44
EB30	COMPARE AND SWAP (64)	CSG	RSY-a	C	N	A SP	\$	ST	B ₂
EB31	COMPARE DOUBLE AND SWAP (32)	CDSY	RSY-a	C	LD	A SP	\$	ST	B ₂
EB3E	COMPARE DOUBLE AND SWAP (64)	CDSG	RSY-a	C	N	A SP	\$	ST	B ₂
EB44	BRANCH ON INDEX HIGH (64)	BXHG	RSY-a	N			B		7-38
EB45	BRANCH ON INDEX LOW OR EQUAL (64)	BXLEG	RSY-a	N			B		7-38
EB4C	EXTRACT CACHE ATTRIBUTE	ECAG	RSY-a	GE					7-208
EB51	TEST UNDER MASK	TMY	SIY	C	LD	A		B ₁	7-316
EB52	MOVE (immediate)	MVIY	SIY	LD	A	A	ST	B ₁	7-233
EB54	AND (immediate)	NIY	SIY	C	LD	A	ST	B ₁	7-32
EB55	COMPARE LOGICAL (immediate)	CLYI	SIY	C	LD	A		B ₁	7-130
EB56	OR (immediate)	OYI	SIY	C	LD	A	ST	B ₁	7-251
EB57	EXCLUSIVE OR (immediate)	XIY	SIY	C	LD	A	ST	B ₁	7-205
EB6A	ADD IMMEDIATE (32←8)	ASI	SIY	C	GE	A	IF	ST	B ₁
EB6E	ADD LOGICAL WITH SIGNED IMMEDIATE (32←8)	ALSI	SIY	C	GE	A		ST	B ₁
EB7A	ADD IMMEDIATE (64←8)	AGSI	SIY	C	GE	A	IF	ST	B ₁
EB7E	ADD LOGICAL WITH SIGNED IMMEDIATE (64←8)	ALGSI	SIY	C	GE	A		ST	B ₁
EB80	INSERT CHARACTERS UNDER MASK (high)	ICMH	RSY-b	C	N	A		B ₂	7-212
EB81	INSERT CHARACTERS UNDER MASK (low)	ICMY	RSY-b	C	LD	A		B ₂	7-212
EB8E	MOVE LONG UNICODE	MVCLU	RSY-a	C	E2	A SP	IC	ST	R ₁ R ₃
EB8F	COMPARE LOGICAL LONG UNICODE	CLCLU	RSY-a	C	E2	A SP	IC		R ₁ R ₂
EB90	STORE MULTIPLE (32)	STMY	RSY-a	LD	A	A		ST	B ₂
EB96	LOAD MULTIPLE HIGH	LMH	RSY-a	N	A	A			B ₂
EB98	LOAD MULTIPLE (32)	LMY	RSY-a	LD	A	A			B ₂
EB9A	LOAD ACCESS MULTIPLE	LAMY	RSY-a	LD	A	A SP		ST	UB
EB9B	STORE ACCESS MULTIPLE	STAMY	RSY-a	LD	A	A SP		ST	UB
EBC0	TEST DECIMAL	TP	RSL	C	E2	A		B ₁	B ₂
EBDC	SHIFT RIGHT SINGLE (32)	SRAK	RSY-a	C	DO				
EBDD	SHIFT LEFT SINGLE (32)	SLAK	RSY-a	C	DO		IF		
EBDE	SHIFT RIGHT SINGLE LOGICAL (32)	SRLK	RSY-a	DO					
EBDF	SHIFT LEFT SINGLE LOGICAL (32)	SLLK	RSY-a	DO					
EBE2	LOAD ON CONDITION (64)	LOC	RSY-b	CL	A	A			B ₂
EBE3	STORE ON CONDITION (64)	STOC	RSY-b	CL	A	A	ST		B ₂
EBE4	LOAD AND AND (64)	LANG	RSY-a	C	IA	A SP	ST		B ₂
EBE6	LOAD AND OR (64)	LAOG	RSY-a	C	IA	A SP	ST		B ₂
EBE7	LOAD AND EXCLUSIVE OR (64)	LAXG	RSY-a	C	IA	A SP	ST		B ₂
EBE8	LOAD AND ADD (64)	LAAG	RSY-a	C	IA	A SP	IF	ST	B ₂
EBEA	LOAD AND ADD LOGICAL (64)	LAALG	RSY-a	C	IA	A SP		ST	B ₂
EBF2	LOAD ON CONDITION (32)	LOC	RSY-b	CL	A	A			B ₂
EBF3	STORE ON CONDITION (32)	STOC	RSY-b	CL	A	A		ST	B ₂
EBF4	LOAD AND AND (32)	LAN	RSY-a	C	IA	A SP		ST	B ₂
EBF6	LOAD AND OR (32)	LAO	RSY-a	C	IA	A SP		ST	B ₂
EBF7	LOAD AND EXCLUSIVE OR (32)	LAX	RSY-a	C	IA	A SP		ST	B ₂
EBF8	LOAD AND ADD (32)	LAA	RSY-a	C	IA	A SP	IF	ST	B ₂
EBFA	LOAD AND ADD LOGICAL (32)	LAAL	RSY-a	C	IA	A SP		ST	B ₂
EC44	BRANCH RELATIVE ON INDEX HIGH (64)	BRXHG	RIE-e	N			B		
EC45	BRANCH RELATIVE ON INDEX LOW OR EQ. (64)	BRXLG	RIE-e	N			B		
EC51	ROTATE THEN INSERT SELECTED BITS LOW	RISBLG	RIE-f	HW					

Figure B-3. Instructions Arranged by Operation Code (Part 16 of 18)

Op-code	Name	Mne-monic	Characteristics										Page
EC54	ROTATE THEN AND SELECTED BITS	RNSBG	RIE-f	C	GE								7-287
EC55	ROTATE THEN INSERT SELECTED BITS	RISBG	RIE-f	C	GE								7-288
EC56	ROTATE THEN OR SELECTED BITS	ROSBG	RIE-f	C	GE								7-287
EC57	ROTATE THEN EXCLUSIVE OR SELECTED BITS	RXSBG	RIE-f	C	GE								7-287
EC5D	ROTATE THEN INSERT SELECTED BITS HIGH	RISBHG	RIE-f		HW								7-290
EC64	COMPARE AND BRANCH RELATIVE (64)	CGRJ	RIE-b		GE						B		7-113
EC65	COMPARE LOGICAL AND BRANCH RELATIVE (64)	CLGRJ	RIE-b		GE						B		7-132
EC70	COMPARE IMMEDIATE AND TRAP (64←16)	CGIT	RIE-a		GE			Dc					7-127
EC71	COMPARE LOGICAL IMMEDIATE AND TRAP (64←16)	CLGIT	RIE-a		GE			Dc					7-133
EC72	COMPARE IMMEDIATE AND TRAP (32←16)	CIT	RIE-a		GE			Dc					7-127
EC73	COMPARE LOGICAL IMMEDIATE AND TRAP (32←16)	CLFIT	RIE-a		GE			Dc					7-133
EC76	COMPARE AND BRANCH RELATIVE (32)	CRJ	RIE-b		GE						B		7-113
EC77	COMPARE LOGICAL AND BRANCH RELATIVE (32)	CLRJ	RIE-b		GE						B		7-132
EC7C	COMPARE IMMEDIATE AND BRANCH RELATIVE (64←8)	CGIJ	RIE-c		GE						B		7-114
EC7D	COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (64←8)	CLGIJ	RIE-c		GE						B		7-132
EC7E	COMPARE IMMEDIATE AND BRANCH RELATIVE (32←8)	CIJ	RIE-c		GE						B		7-114
EC7F	COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE (32←8)	CLIJ	RIE-c		GE						B		7-132
ECD8	ADD IMMEDIATE (32←16)	AHIK	RIE-d	C	DO			IF					7-25
ECD9	ADD IMMEDIATE (64←16)	AGHIK	RIE-d	C	DO			IF					7-25
ECDA	ADD LOGICAL WITH SIGNED IMMEDIATE (32←16)	ALHSIK	RIE-d	C	DO								7-30
ECDB	ADD LOGICAL WITH SIGNED IMMEDIATE (64←16)	ALGHSIK	RIE-d	C	DO								7-30
ECE4	COMPARE AND BRANCH (64)	CGRB	RRS		GE						B		7-113
ECE5	COMPARE LOGICAL AND BRANCH (64)	CLGRB	RRS		GE						B		7-132
ECF6	COMPARE AND BRANCH (32)	CRB	RRS		GE						B		7-113
ECF7	COMPARE LOGICAL AND BRANCH (32)	CLRB	RRS		GE						B		7-132
ECFC	COMPARE IMMEDIATE AND BRANCH (64←8)	CGIB	RIS		GE						B		7-113
ECFD	COMPARE LOGICAL IMMEDIATE AND BRANCH (64←8)	CLGIB	RIS		GE						B		7-132
ECFE	COMPARE IMMEDIATE AND BRANCH (32←8)	CIB	RIS		GE						B		7-113
ECFF	COMPARE LOGICAL IMMEDIATE AND BRANCH (32←8)	CLIB	RIS		GE						B		7-132
ED04	LOAD LENGTHENED (short to long BFP)	LDEB	RXE			A		Db	Xi			B ₂	19-33
ED05	LOAD LENGTHENED (long to extended BFP)	LXDB	RXE			A	SP	Db	Xi			B ₂	19-33
ED06	LOAD LENGTHENED (short to extended BFP)	LXEB	RXE			A	SP	Db	Xi			B ₂	19-33
ED07	MULTIPLY (long to extended BFP)	MXDB	RXE			A	SP	Db	Xi			B ₂	19-35
ED08	COMPARE AND SIGNAL (short BFP)	KEB	RXE	C		A		Db	Xi			B ₂	19-17
ED09	COMPARE (short BFP)	CEB	RXE	C		A		Db	Xi			B ₂	19-16
ED0A	ADD (short BFP)	AEB	RXE	C		A		Db	Xi	Xo Xu Xx		B ₂	19-11
ED0B	SUBTRACT (short BFP)	SEB	RXE	C		A		Db	Xi	Xo Xu Xx		B ₂	19-39
ED0C	MULTIPLY (short to long BFP)	MDEB	RXE			A		Db	Xi			B ₂	19-35
ED0D	DIVIDE (short BFP)	DEB	RXE			A		Db	Xi	Xz Xo Xu Xx		B ₂	19-26
ED0E	MULTIPLY AND ADD (short BFP)	MAEB	RXF			A		Db	Xi	Xo Xu Xx		B ₂	19-37
ED0F	MULTIPLY AND SUBTRACT (short BFP)	MSEB	RXF			A		Db	Xi	Xo Xu Xx		B ₂	19-37
ED10	TEST DATA CLASS (short BFP)	TCEB	RXE	C				Db					19-39
ED11	TEST DATA CLASS (long BFP)	TCDB	RXE	C				Db					19-39
ED12	TEST DATA CLASS (extended BFP)	TCXB	RXE	C			SP	Db					19-39
ED14	SQUARE ROOT (short BFP)	SQEB	RXE			A		Db	Xi		Xx	B ₂	19-38
ED15	SQUARE ROOT (long BFP)	SQDB	RXE			A		Db	Xi		Xx	B ₂	19-38
ED17	MULTIPLY (short BFP)	MEEB	RXE			A		Db	Xi	Xo Xu Xx		B ₂	19-35
ED18	COMPARE AND SIGNAL (long BFP)	KDB	RXE	C		A		Db	Xi			B ₂	19-17

Figure B-3. Instructions Arranged by Operation Code (Part 17 of 18)

Op-code	Name	Mne- monic	Characteristics						Page
ED19	COMPARE (long BFP)	CDB	RXE	C	A	Db Xi		B ₂	19-16
ED1A	ADD (long BFP)	ADB	RXE	C	A	Db Xi	Xo Xu Xx	B ₂	19-11
ED1B	SUBTRACT (long BFP)	SDB	RXE	C	A	Db Xi	Xo Xu Xx	B ₂	19-39
ED1C	MULTIPLY (long BFP)	MDB	RXE		A	Db Xi	Xo Xu Xx	B ₂	19-35
ED1D	DIVIDE (long BFP)	DDB	RXE		A	Db Xi Xz	Xo Xu Xx	B ₂	19-26
ED1E	MULTIPLY AND ADD (long BFP)	MADB	RXF		A	Db Xi	Xo Xu Xx	B ₂	19-37
ED1F	MULTIPLY AND SUBTRACT (long BFP)	MSDB	RXF		A	Db Xi	Xo Xu Xx	B ₂	19-37
ED24	LOAD LENGTHENED (short to long HFP)	LDE	RXE		A	Da		B ₂	18-15
ED25	LOAD LENGTHENED (long to extended HFP)	LXD	RXE		A SP	Da		B ₂	18-15
ED26	LOAD LENGTHENED (short to extended HFP)	LXE	RXE		A SP	Da		B ₂	18-15
ED2E	MULTIPLY AND ADD (short HFP)	MAE	RXF	HM	A	Da EU EO		B ₂	18-19
ED2F	MULTIPLY AND SUBTRACT (short HFP)	MSE	RXF	HM	A	Da EU EO		B ₂	18-19
ED34	SQUARE ROOT (short HFP)	SQE	RXE		A	Da	SQ	B ₂	18-23
ED35	SQUARE ROOT (long HFP)	SQD	RXE		A	Da	SQ	B ₂	18-23
ED37	MULTIPLY (short HFP)	MEE	RXE		A	Da EU EO		B ₂	18-17
ED38	MULTIPLY AND ADD UNNRM. (long to ext. low HFP)	MAYL	RXF	UE	A	Da		B ₂	18-20
ED39	MULTIPLY UNNORM. (long to ext. low HFP)	MYL	RXF	UE	A	Da		B ₂	18-21
ED3A	MULTIPLY & ADD UNNORMALIZED (long to ext. HFP)	MAY	RXF	UE	A	Da		B ₂	18-20
ED3B	MULTIPLY UNNORMALIZED (long to ext. HFP)	MY	RXF	UE	A SP	Da		B ₂	18-21
ED3C	MULTIPLY AND ADD UNNRM. (long to ext. high HFP)	MAYH	RXF	UE	A	Da		B ₂	18-20
ED3D	MULTIPLY UNNORM. (long to ext. high HFP)	MYH	RXF	UE	A	Da		B ₂	18-21
ED3E	MULTIPLY AND ADD (long HFP)	MAD	RXF	HM	A	Da EU EO		B ₂	18-19
ED3F	MULTIPLY AND SUBTRACT (long HFP)	MSD	RXF	HM	A	Da EU EO		B ₂	18-19
ED40	SHIFT SIGNIFICAND LEFT (long DFP)	SLDT	RXF	TF		Dt			20-49
ED41	SHIFT SIGNIFICAND RIGHT (long DFP)	SRDT	RXF	TF		Dt			20-50
ED48	SHIFT SIGNIFICAND LEFT (extended DFP)	SLXT	RXF	TF	SP	Dt			20-49
ED49	SHIFT SIGNIFICAND RIGHT (extended DFP)	SRXT	RXF	TF	SP	Dt			20-50
ED50	TEST DATA CLASS (short DFP)	TDCET	RXE	C TF		Dt			20-51
ED51	TEST DATA GROUP (short DFP)	TDGET	RXE	C TF		Dt			20-52
ED54	TEST DATA CLASS (long DFP)	TDCDT	RXE	C TF		Dt			20-51
ED55	TEST DATA GROUP (long DFP)	TDGDT	RXE	C TF		Dt			20-52
ED58	TEST DATA CLASS (extended DFP)	TDCXT	RXE	C TF	SP	Dt			20-51
ED59	TEST DATA GROUP (extended DFP)	TDGXT	RXE	C TF	SP	Dt			20-52
ED64	LOAD (short)	LEY	RXY-a	LD	A	Da		B ₂	9-30
ED65	LOAD (long)	LDY	RXY-a	LD	A	Da		B ₂	9-30
ED66	STORE (short)	STEY	RXY-a	LD	A	Da		ST B ₂	9-47
ED67	STORE (long)	STDY	RXY-a	LD	A	Da		ST B ₂	9-47
EE	PERFORM LOCKED OPERATION	PLO	SS-e	C	A SP		\$ GM	ST FC	7-270
EF	LOAD MULTIPLE DISJOINT	LMD	SS-e	N	A			B ₂ B ₄	7-227
F0	SHIFT AND ROUND DECIMAL	SRP	SS-c	C	A	Dd DF		ST B ₁ B ₂	8-12
F1	MOVE WITH OFFSET	MVO	SS-b		A			ST B ₁ B ₂	7-246
F2	PACK	PACK	SS-b		A			ST B ₁ B ₂	7-252
F3	UNPACK	UNPK	SS-b		A			ST B ₁ B ₂	7-331
F8	ZERO AND ADD	ZAP	SS-b	C	A	Dd DF		ST B ₁ B ₂	8-13
F9	COMPARE DECIMAL	CP	SS-b	C	A	Dd		B ₁ B ₂	8-6
FA	ADD DECIMAL	AP	SS-b	C	A	Dd DF		ST B ₁ B ₂	8-6
FB	SUBTRACT DECIMAL	SP	SS-b	C	A	Dd DF		ST B ₁ B ₂	8-13
FC	MULTIPLY DECIMAL	MP	SS-b		A SP	Dd		ST B ₁ B ₂	8-11
FD	DIVIDE DECIMAL	DP	SS-b		A SP	Dd	DK	ST B ₁ B ₂	8-7

Figure B-3. Instructions Arranged by Operation Code (Part 18 of 18)

Appendix C. Condition-Code Settings

This appendix lists the condition-code setting for instructions in z/Architecture which set the condition code. In addition to those instructions listed which set the condition code, the condition code may be changed by DIAGNOSE and the target of EXECUTE. The condition code is loaded by LOAD PSW, LOAD PSW EXTENDED, PROGRAM RETURN, RESUME PROGRAM, and SET PROGRAM MASK and by an

interruption. The condition code is set to zero by initial CPU reset and is loaded by the successful conclusion of the initial-program-loading sequence.

Some models may offer instructions which set the condition code and do not appear in this document, such as those provided for assists or as part of special or custom features.

Instruction	Condition Code			
	0	1	2	3
ADD (general)	Zero	< zero	> zero	Overflow
ADD (BFP, DFP)	Zero	< zero	> zero	NaN
ADD DECIMAL	Zero	< zero	> zero	Overflow
ADD HALFWORD	Zero	< zero	> zero	Overflow
ADD HALFWORD IMMEDIATE	Zero	< zero	> zero	Overflow
ADD HIGH	Zero	< zero	> zero	Overflow
ADD IMMEDIATE	Zero	< zero	> zero	Overflow
ADD IMMEDIATE HIGH	Zero	< zero	> zero	Overflow
ADD LOGICAL	Zero, no carry	Not zero, no carry	Zero, carry	Not zero, carry
ADD LOGICAL HIGH	Zero, no carry	Not zero, no carry	Zero, carry	Not zero, carry
ADD LOGICAL IMMEDIATE	Zero, no carry	Not zero, no carry	Zero, carry	Not zero, carry
ADD LOGICAL WITH CARRY	Zero, no carry	Not zero, no carry	Zero, carry	Not zero, carry
ADD LOGICAL WITH SIGNED IMMEDIATE	Zero, no carry	Not zero, no carry	Zero, carry	Not zero, carry
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH	Zero, no carry	Not zero, no carry	Zero, carry	Not zero, carry
ADD NORMALIZED	Zero	< zero	> zero	—
ADD UNNORMALIZED	Zero	< zero	> zero	—
AND	Zero	Not zero	—	—
AND IMMEDIATE	Zero	Not zero	—	—
CANCEL SUBCHANNEL	Function initiated	—	—	Not operational
CHECKSUM	Checksum complete	—	—	CPU-determined completion
CIPHER MESSAGE	Normal completion	Verification mismatch	—	Partial completion
CIPHER MESSAGE WITH CHAINING	Normal completion	Verification mismatch	—	Partial completion
CIPHER MESSAGE WITH CFB	Normal completion	Verification mismatch	—	Partial completion
CIPHER MESSAGE WITH COUNTER	Normal completion	Verification mismatch	—	Partial completion
CIPHER MESSAGE WITH OFB	Normal completion	Verification mismatch	—	Partial completion
CLEAR SUBCHANNEL	Function initiated	—	—	Not operational
COMPARE (BFP, DFP)	Equal	Low	High	Unordered
COMPARE (general, HFP)	Equal	Low	High	—
COMPARE AND FORM CODEWORD	Equal	OCB=0: low OCB=1: high	OCB=0: high OCB=1: low	—
COMPARE AND SIGNAL	Equal	Low	High	Unordered
COMPARE AND SWAP	Equal	Not equal	—	—
COMPARE AND SWAP AND PURGE	Equal	Not equal	—	—
COMPARE AND SWAP AND STORE	Equal	Not equal	—	—
COMPARE BIASED EXPONENT	Equal	Low	High	Unordered
COMPARE DECIMAL	Equal	Low	High	—
COMPARE DOUBLE AND SWAP	Equal	Not equal	—	—

Figure C-1. Summary of Condition-Code Settings (Part 1 of 6)

Instruction	Condition Code			
	0	1	2	3
COMPARE HALFWORD	Equal	Low	High	—
COMPARE HALFWORD IMMEDIATE	Equal	Low	High	—
COMPARE HALFWORD RELATIVE LONG	Equal	Low	High	—
COMPARE HIGH	Equal	Low	High	—
COMPARE IMMEDIATE	Equal	Low	High	—
COMPARE IMMEDIATE HIGH	Equal	Low	High	—
COMPARE LOGICAL	Equal	Low	High	—
COMPARE LOGICAL CHARACTERS UNDER MASK	Equal	Low	High	—
COMPARE LOGICAL HIGH	Equal	Low	High	—
COMPARE LOGICAL IMMEDIATE	Equal	Low	High	—
COMPARE LOGICAL IMMEDIATE HIGH	Equal	Low	High	—
COMPARE LOGICAL LONG	Equal	Low	High	—
COMPARE LOGICAL LONG EXTENDED	Equal	Low	High	CPU-determined completion
COMPARE LOGICAL LONG UNICODE	Equal	Low	High	CPU-determined completion
COMPARE LOGICAL RELATIVE LONG	Equal	Low	High	—
COMPARE LOGICAL STRING	Equal	Low	High	CPU-determined completion
COMPARE RELATIVE LONG	Equal	Low	High	—
COMPARE UNTIL SUBSTRING EQUAL	Equal substrings	Last bytes equal	Last bytes unequal	CPU-determined completion
COMPRESSION CALL	Op2 processed	Op1 full and op2 not processed	—	CPU-determined completion
COMPUTE INTERMEDIATE MESSAGE DIGEST	Normal completion	—	—	Partial completion
COMPUTE LAST MESSAGE DIGEST	Normal completion	—	—	Partial completion
COMPUTE MESSAGE AUTHENTICATION CODE	Normal completion	Verification mismatch	—	Partial completion
CONVERT BFP TO HFP	Zero	< zero	> zero	Special case
CONVERT HFP TO BFP	Zero	< zero	> zero	Special case
CONVERT TO FIXED	Zero	< zero	> zero	Special case
CONVERT TO LOGICAL	Zero	< zero	> zero	Special case
CONVERT UTF-16 TO UTF-32	Data processed	Op1 full	Invalid low surrogate	CPU-determined completion
CONVERT UTF-16 TO UTF-8	Data processed	Op1 full	Invalid low surrogate	CPU-determined completion
CONVERT UTF-32 TO UTF-16	Data processed	Op1 full	Invalid UTF-32 character	CPU-determined completion
CONVERT UTF-32 TO UTF-8	Data processed	Op1 full	Invalid UTF-32 character	CPU-determined completion
CONVERT UTF-8 TO UTF-16	Data processed	Op1 full	Invalid UTF-8 character	CPU-determined completion
CONVERT UTF-8 TO UTF-32	Data processed	Op1 full	Invalid UTF-8 character	CPU-determined completion
DIAGNOSE	See note	See note	See note	See note
DIVIDE TO INTEGER	Remainder complete; normal quotient	Remainder complete; quotient overflow or NaN	Remainder incomplete; normal quotient	Remainder incomplete; quotient overflow or NaN
EDIT	Zero	< zero	> zero	—
EDIT AND MARK	Zero	< zero	> zero	—
EXCLUSIVE OR	Zero	Not zero	—	—

Figure C-1. Summary of Condition-Code Settings (Part 2 of 6)

Instruction	Condition Code			
	0	1	2	3
EXCLUSIVE OR IMMEDIATE EXTRACT STACKED STATE	Zero Branch state entry	Not zero Program-call state entry	— —	— —
FIND LEFTMOST ONE HALT SUBCHANNEL	No one bit found Function initiated	— Status-pending with other than intermediate status	One bit found Busy	— Not operational
INSERT ADDRESS SPACE CONTROL	Primary-space mode	Secondary-space mode	Access-register mode	Home-space mode
INSERT CHARACTERS UNDER MASK LOAD ADDRESS SPACE PARAMETERS	All zeros Parameters loaded	First bit one Primary ASN not available	First bit zero Secondary ASN not available or not authorized	— Space-switch event
LOAD AND TEST (BFP, DFP)	Zero	< zero	> zero	NaN
LOAD AND TEST (general, HFP)	Zero	< zero	> zero	—
LOAD COMPLEMENT (BFP)	Zero	< zero	> zero	NaN
LOAD COMPLEMENT (gen)	Zero	< zero	> zero	Overflow
LOAD COMPLEMENT (HFP)	Zero	< zero	> zero	—
LOAD NEGATIVE (BFP)	Zero	< zero	—	NaN
LOAD NEGATIVE (gen, HFP)	Zero	< zero	—	—
LOAD PAGE TABLE ENTRY ADDRESS	Translation, STE.P = 0	Translation, STE.P = 1	1 bit on in RTE or STE, or enhanced-DAT applies and STE.FC=1	Exception condition exists
LOAD POSITIVE (BFP)	Zero	—	> zero	NaN
LOAD POSITIVE (gen)	Zero	—	> zero	Overflow
LOAD POSITIVE (HFP)	Zero	—	> zero	—
LOAD PSW / LOAD PSW EXTENDED	See note	See note	See note	See note
LOAD REAL ADDRESS	Translation available	ST entry invalid	PT entry invalid	ASCE or entry not available or length violation
MODIFY SUBCHANNEL	SCHIB information placed in subchannel	Status-pending	Busy	Not operational
MOVE LONG	Length equal	Length low	Length high	Destructive overlap
MOVE LONG EXTENDED	Length equal	Length low	Length high	CPU-determined completion
MOVE LONG UNICODE	Length equal	Length low	Length high	CPU-determined completion
MOVE PAGE	Data moved	Operand 1 invalid, both valid in ES, locked, or ES error	Operand 2 invalid	—
MOVE STRING	—	Data moved	—	CPU-determined completion
MOVE TO PRIMARY	Length ≤ 256	—	—	Length > 256
MOVE TO SECONDARY	Length ≤ 256	—	—	Length > 256
MOVE WITH KEY	Length ≤ 256	—	—	Length > 256
MOVE WITH OPTIONAL SPECIFICATIONS	Length ≤ 4,096	—	—	Length > 4,096
OR	Zero	Not zero	—	—
OR IMMEDIATE	Zero	Not zero	—	—
PAGE IN	Page-in operation completed	Expanded-storage data error	—	Expanded-storage block not available

Figure C-1. Summary of Condition-Code Settings (Part 3 of 6)

Instruction	Condition Code			
	0	1	2	3
PAGE OUT	Page-out operation completed	Expanded-storage data error	—	Expanded-storage block not available
PERFORM CRYPTOGRAPHIC COMPUTATION	Normal completion	Verification mismatch	Invalid index or length	Partial completion
PERFORM FLOATING-POINT OPERATION (test bit one)	Function code valid	—	—	Function code invalid
PERFORM FLOATING-POINT OPERATION (test bit zero)	Normal result	Nontrap exception	Trap exception with alternate action	—
PERFORM LOCKED OPERATION (test bit one)	Function code valid	—	—	Function code invalid
PERFORM LOCKED OPERATION (test bit zero)	Equal	Op1 not equal	Op1 equal, op3 not equal (dcs only)	—
PERFORM TIMING FACILITY FUNCTION	Function performed	—	—	Function not available
PERFORM TOPOLOGY FUNCTION (FC 0 or 1)	Change initiated	—	Request rejected	—
PERFORM TOPOLOGY FUNCTION (FC 2)	Report not pending	Report pending	—	—
POPULATION COUNT	Zero	Not zero	—	—
PROGRAM RETURN	See note	See note	See note	See note
RESET CHANNEL PATH	Function initiated	—	Busy	Not operational
RESET REFERENCE BIT EXTENDED	R bit zero, C bit zero	R bit zero, C bit one	R bit one, C bit zero	R bit one, C bit one
RESUME PROGRAM	See note	See note	See note	See note
RESUME SUBCHANNEL	Function initiated	Status pending	Function not applicable	Not operational
ROTATE THEN AND SELECTED BITS	Zero	Not zero	—	—
ROTATE THEN EXCLUSIVE OR SELECTED BITS	Zero	Not zero	—	—
ROTATE THEN INSERT SELECTED BITS	Zero	< zero	> zero	—
ROTATE THEN OR SELECTED BITS	Zero	Not zero	—	—
SEARCH STRING	—	Found	Not found	CPU-determined completion
SEARCH STRING UNICODE	—	Found	Not found	CPU-determined completion
SET CLOCK	Set	Secure	—	Not operational
SET PROGRAM MASK	See note	See note	See note	See note
SET STORAGE KEY EXTENDED	Storage key not set	Entire storage key set	Partial storage key set	Entire storage key set; bits 48-55 of GR R ₁ unpredictable
SHIFT AND ROUND DECIMAL	Zero	< zero	> zero	Overflow
SHIFT LEFT (DOUBLE/SINGLE)	Zero	< zero	> zero	Overflow
SHIFT RIGHT (DOUBLE/SINGLE)	Zero	< zero	> zero	—
SIGNAL PROCESSOR	Order accepted	Status stored	Busy	Not operational
START SUBCHANNEL	Function initiated	Status-pending	Busy	Not operational
STORE CHANNEL REPORT WORD	CRW stored	Zeros stored	—	—
STORE CLOCK	Set	Not set	Error	Stopped or not operational
STORE CLOCK EXTENDED	Set	Not set	Error	Stopped or not operational
STORE CLOCK FAST	Set	Not set	Error	Stopped or not operational
STORE FACILITY LIST EXTENDED	Complete facility list stored	—	—	Incomplete facility list stored
STORE SUBCHANNEL	SCHIB stored	—	—	Not operational
STORE SYSTEM INFORMATION	Information provided	—	—	Information not available

Figure C-1. Summary of Condition-Code Settings (Part 4 of 6)

Instruction	Condition Code			
	0	1	2	3
SUBTRACT (BFP, DFP)	Zero	< zero	> zero	NaN
SUBTRACT (general)	Zero	< zero	> zero	Overflow
SUBTRACT DECIMAL	Zero	< zero	> zero	Overflow
SUBTRACT HALFWORD	Zero	< zero	> zero	Overflow
SUBTRACT HIGH	Zero	< zero	> zero	Overflow
SUBTRACT LOGICAL	—	Not zero, borrow	Zero, no borrow	Not zero, no borrow
SUBTRACT LOGICAL HIGH	—	Not zero, borrow	Zero, no borrow	Not zero, no borrow
SUBTRACT LOGICAL IMMEDIATE	—	Not zero, borrow	Zero, no borrow	Not zero, no borrow
SUBTRACT LOGICAL WITH BORROW	Zero, borrow	Not zero, borrow	Zero, no borrow	Not zero, no borrow
SUBTRACT NORMALIZED (HFP)	Zero	< zero	> zero	—
SUBTRACT UNNORMALIZED (HFP)	Zero	< zero	> zero	—
TEST ACCESS	ALET 0	DU access list, no exceptions	PS access list, no exceptions	ALET 1 or exceptions
TEST ADDRESSING MODE	Twenty-four bit mode	Thirty-one bit mode	—	Sixty-four bit mode
TEST AND SET	Left bit zero	Left bit one	—	—
TEST BLOCK	Usable	Not usable	—	—
TEST DATA CLASS	Zero (no match)	One (match)	—	—
TEST DATA GROUP	Zero (no match)	One (match)	—	—
TEST DECIMAL	Digits and sign valid	Sign invalid	Digit invalid	Sign and digit invalid
TEST PENDING INTERRUPTION	Interruption code not stored	Interruption code stored	—	—
TEST PROTECTION	Can fetch, can store	Can fetch, cannot store	Cannot fetch, cannot store	Translation not available
TEST SUBCHANNEL	IRB stored; subchannel status-pending	IRB stored; subchannel not status-pending	—	Not operational
TEST UNDER MASK	All zeros	Mixed	—	All ones
TEST UNDER MASK (HIGH/LOW)	All zeros	Mixed, left bit zero	Mixed, left bit one	All ones
TRANSLATE AND TEST	All zeros	Incomplete	Complete	—
TRANSLATE AND TEST EXTENDED	All zeros	Nonzero code selected	—	CPU-determined completion
TRANSLATE AND TEST REVERSE	All zeros	Incomplete	Complete	—
TRANSLATE AND TEST REVERSE EXTENDED	All zeros	Nonzero code selected	—	CPU-determined completion
TRANSLATE EXTENDED	Data processed	Op1 byte equal test byte	—	CPU-determined completion
TRANSLATE ONE TO ONE, ONE TO TWO, TWO TO ONE, TWO TO TWO	Character equal test character not found	Character equal test character found	—	CPU-determined completion
UNPACK ASCII	Sign plus	Sign minus	—	Sign invalid
UNPACK UNICODE	Sign plus	Sign minus	—	Sign invalid
UPDATE TREE	Equal	Not equal or no comparison	—	GR5 nonzero, GR0 negative
ZERO AND ADD	Zero	< zero	> zero	Overflow

Figure C-1. Summary of Condition-Code Settings (Part 5 of 6)

Instruction	Condition Code			
	0	1	2	3
Notes: <ol style="list-style-type: none"> 1. For DIAGNOSE, the resulting condition code is model-dependent. 2. For LOAD REAL ADDRESS, condition code 3 is set if address-space-control element not available, region-table entry outside table or invalid, segment-table entry outside table, or, for LRA in 24- or 31-bit mode when bits 0-32 of entry address not all zeros, segment- or page-table entry invalid. 3. For LOAD PSW, LOAD PSW EXTENDED, and RESUME PROGRAM, the condition code is loaded from the condition-code field of the second operand. 4. For SET PROGRAM MASK, the condition code is loaded from bit positions 2 and 3 of the first operand. 				

Figure C-1. Summary of Condition-Code Settings (Part 6 of 6)

Appendix G. Table of Powers of 2

Plus	Minus
1	0 1.
2	1 0.5
4	2 0.25
8	3 0.125
16	4 0.0625
32	5 0.03125
64	6 0.01562 5
128	7 0.00781 25
256	8 0.00390 625
512	9 0.00195 3125
1,024	10 0.00097 65625
2,048	11 0.00048 82812 5
4,096	12 0.00024 41406 25
8,192	13 0.00012 20703 125
16,384	14 0.00006 10351 5625
32,768	15 0.00003 05175 78125
65,536	16 0.00001 52587 89062 5
131,072	17 0.00000 76293 94531 25
262,144	18 0.00000 38146 97265 625
524,288	19 0.00000 19073 48632 8125
1,048,576	20 0.00000 09536 74316 40625
2,097,152	21 0.00000 04768 37158 20312 5
4,194,304	22 0.00000 02384 18579 10156 25
8,388,608	23 0.00000 01192 09289 55078 125
16,777,216	24 0.00000 00596 04644 77539 0625
33,554,432	25 0.00000 00298 02322 38769 53125
67,108,864	26 0.00000 00149 01161 19384 76562 5
134,217,728	27 0.00000 00074 50580 59692 38281 25
268,435,456	28 0.00000 00037 25290 29846 19140 625
536,870,912	29 0.00000 00018 62645 14923 09570 3125
1,073,741,824	30 0.00000 00009 31322 57461 54785 15625
2,147,483,648	31 0.00000 00004 65661 28730 77392 57812 5
4,294,967,296	32 0.00000 00002 32830 64365 38696 28906 25
8,589,934,592	33 0.00000 00001 16415 32182 69348 14453 125
17,179,869,184	34 0.00000 00000 58207 66091 34674 07226 5625
34,359,738,368	35 0.00000 00000 29103 83045 67337 03613 28125
68,719,476,736	36 0.00000 00000 14551 91522 83668 51806 64062 5
137,438,953,472	37 0.00000 00000 07275 95761 41834 25903 32031 25
274,877,906,944	38 0.00000 00000 03637 97880 70917 12951 66015 625
549,755,813,888	39 0.00000 00000 01818 98940 35458 56475 83007 8125
1,099,511,627,776	40 0.00000 00000 00909 49470 17729 28237 91503 90625
2,199,023,255,552	41 0.00000 00000 00454 74735 08864 64118 95751 95312 5
4,398,046,511,104	42 0.00000 00000 00227 37367 54432 32059 47875 97656 25
8,796,093,022,208	43 0.00000 00000 00113 68683 77216 16029 73937 98828 125
17,592,186,044,416	44 0.00000 00000 00056 84341 88608 08014 86968 99414 0625
35,184,372,088,832	45 0.00000 00000 00028 42170 94304 04007 43484 49707 03125
70,368,744,177,664	46 0.00000 00000 00014 21085 47152 02003 71742 24853 51562 5
140,737,488,355,328	47 0.00000 00000 00007 10542 73576 01001 85871 12426 75781 25
281,474,976,710,656	48 0.00000 00000 00003 55271 36788 00500 92935 56213 37890 625
562,949,953,421,312	49 0.00000 00000 00001 77635 68394 00250 46467 78106 68945 3125
1,125,899,906,842,624	50 0.00000 00000 00000 88817 84197 00125 23233 89053 34472 65625
2,251,799,813,685,248	51 0.00000 00000 00000 44408 92098 50062 61616 94526 67236 32812 5
4,503,599,627,370,496	52 0.00000 00000 00000 22204 46049 25031 30808 47263 33618 16406 25
9,007,199,254,740,992	53 0.00000 00000 00000 11102 23024 62515 65404 23631 66809 08203 125
18,014,398,509,481,984	54 0.00000 00000 00000 05551 11512 31257 82702 11815 83404 54101 5625
36,028,797,018,963,968	55 0.00000 00000 00000 02775 55756 15628 91351 05907 91702 27050 78125
72,057,594,037,927,936	56 0.00000 00000 00000 01387 77878 07814 45675 52953 95851 13525 39062 5
144,115,188,075,855,872	57 0.00000 00000 00000 00693 88939 03907 22837 76476 97925 56762 69531 25
288,230,376,151,711,744	58 0.00000 00000 00000 00346 94469 51953 61418 88238 48962 78381 34765 625
576,460,752,303,423,488	59 0.00000 00000 00000 00173 47234 75976 80709 44119 24481 39190 67382 8125
1,152,921,504,606,846,976	60 0.00000 00000 00000 00086 73617 37988 40354 72059 62240 69595 33691 40625
2,305,843,009,213,693,952	61 0.00000 00000 00000 00043 36808 68994 20177 36029 81120 34797 66845 70312 5
4,611,686,018,427,387,904	62 0.00000 00000 00000 00021 68404 34497 10088 68014 90560 17398 83422 85156 25
9,223,372,036,854,775,808	63 0.00000 00000 00000 00010 84202 17248 55044 34007 45280 08699 41711 42578 125

Figure G-1. Powers of 2

Plus		Minus
18,446,744,073,709,551,616	64	0.00000 00000 00000 00005 42101 08624 27522 17003 72640 04349 70855 71289 0625
36,893,488,147,419,103,232	65	
73,786,976,294,838,206,464	66	
147,573,952,589,676,412,928	67	
295,147,905,179,352,825,856	68	
590,295,810,358,705,651,712	69	
1,180,591,620,717,411,303,424	70	
2,361,183,241,434,822,606,848	71	
4,722,366,482,869,645,213,696	72	
9,444,732,965,739,290,427,392	73	
18,889,465,931,478,580,854,784	74	
37,778,931,862,957,161,709,568	75	
75,557,863,725,914,323,419,136	76	
151,115,727,451,828,646,838,272	77	
302,231,454,903,657,293,676,544	78	
604,462,909,807,314,587,353,088	79	
1,208,925,819,614,629,174,706,176	80	
2,417,851,639,229,258,349,412,352	81	
4,835,703,278,458,516,698,824,704	82	
9,671,406,556,917,033,397,649,408	83	
19,342,813,113,834,066,795,298,816	84	
38,685,626,227,668,133,590,597,632	85	
77,371,252,455,336,267,181,195,264	86	
154,742,504,910,672,534,362,390,528	87	
309,485,009,821,345,068,724,781,056	88	
618,970,019,642,690,137,449,562,112	89	
1,237,940,039,285,380,274,899,124,224	90	
2,475,880,078,570,760,549,798,248,448	91	
4,951,760,157,141,521,099,596,496,896	92	
9,903,520,314,283,042,199,192,993,792	93	
19,807,040,628,566,084,398,385,987,584	94	
39,614,081,257,132,168,796,771,975,168	95	
79,228,162,514,264,337,593,543,950,336	96	
158,456,325,028,528,675,187,087,900,672	97	
316,912,650,057,057,350,374,175,801,344	98	
633,825,300,114,114,700,748,351,602,688	99	
1,267,650,600,228,229,401,496,703,205,376	100	
2,535,301,200,456,458,802,993,406,410,752	101	
5,070,602,400,912,917,605,986,812,821,504	102	
10,141,204,801,825,835,211,973,625,643,008	103	
20,282,409,603,651,670,423,947,251,286,016	104	
40,564,819,207,303,340,847,894,502,572,032	105	
81,129,638,414,606,681,695,789,005,144,064	106	
162,259,276,829,213,363,391,578,010,288,128	107	
324,518,553,658,426,726,783,156,020,576,256	108	
649,037,107,316,853,453,566,312,041,152,512	109	
1,298,074,214,633,706,907,132,624,082,305,024	110	
2,596,148,429,267,413,814,265,248,164,610,048	111	
5,192,296,858,534,827,628,530,496,329,220,096	112	
10,384,593,717,069,655,257,060,992,658,440,192	113	
20,769,187,434,139,310,514,121,985,316,880,384	114	
41,538,374,868,278,621,028,243,970,633,760,768	115	
83,076,749,736,557,242,056,487,941,267,521,536	116	
166,153,499,473,114,484,112,975,882,535,043,072	117	
332,306,998,946,228,968,225,951,765,070,086,144	118	
664,613,997,892,457,936,451,903,530,140,172,288	119	
1,329,227,995,784,915,872,903,807,060,280,344,576	120	
2,658,455,991,569,831,745,807,614,120,560,689,152	121	
5,316,911,983,139,663,491,615,228,241,121,378,304	122	
10,633,823,966,279,326,983,230,456,482,242,756,608	123	
21,267,647,932,558,653,966,460,912,964,485,513,216	124	
42,535,295,865,117,307,932,921,825,928,971,026,432	125	
85,070,591,730,234,615,865,843,651,857,942,052,864	126	
170,141,183,460,469,231,731,687,303,715,884,105,728	127	
340,282,366,920,938,463,463,374,607,431,768,211,456	128	

Figure G-1. Powers of 2

Appendix H. Hexadecimal Tables

The following tables aid in converting hexadecimal values to decimal values, or the reverse.

Direct Conversion Table

This table provides direct conversion of decimal and hexadecimal numbers in these ranges:

Hexadecimal
000 to FFF

Decimal
0000 to 4095

To convert numbers outside these ranges, and to convert fractions, use the hexadecimal and decimal conversion tables that follow the direct conversion table in this appendix.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00_	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01_	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02_	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03_	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04_	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05_	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06_	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07_	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08_	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09_	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A_	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B_	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C_	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D_	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E_	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F_	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10_	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11_	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12_	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13_	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14_	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15_	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16_	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17_	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18_	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19_	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A_	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B_	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C_	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D_	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E_	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F_	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20_	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21_	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22_	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23_	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24_	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25_	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26_	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27_	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28_	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29_	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A_	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B_	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C_	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D_	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E_	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F_	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30_	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31_	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32_	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33_	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34_	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35_	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36_	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37_	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38_	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39_	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A_	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B_	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C_	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D_	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E_	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F_	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40_	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41_	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42_	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43_	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44_	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45_	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46_	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47_	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48_	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49_	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A_	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B_	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C_	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D_	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E_	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F_	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50_	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51_	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52_	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53_	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54_	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55_	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56_	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57_	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58_	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59_	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A_	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B_	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C_	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D_	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E_	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F_	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60_	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61_	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62_	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63_	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64_	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65_	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66_	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67_	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68_	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69_	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A_	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B_	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C_	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D_	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E_	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F_	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
70_	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71_	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72_	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73_	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74_	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75_	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76_	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77_	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78_	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79_	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A_	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B_	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C_	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D_	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E_	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F_	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80_	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81_	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82_	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83_	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84_	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85_	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86_	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87_	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88_	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89_	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A_	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B_	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C_	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D_	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E_	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F_	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90_	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91_	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92_	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93_	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94_	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95_	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96_	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97_	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98_	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99_	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A_	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B_	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C_	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D_	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E_	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F_	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A0_	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1_	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2_	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3_	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4_	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5_	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6_	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7_	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8_	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9_	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA_	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB_	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC_	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD_	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE_	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF_	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0_	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1_	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2_	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3_	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4_	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5_	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6_	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7_	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8_	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9_	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA_	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB_	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC_	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD_	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE_	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF_	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0_	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1_	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2_	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3_	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4_	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5_	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6_	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7_	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8_	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9_	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA_	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB_	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC_	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD_	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE_	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF_	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0_	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1_	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2_	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3_	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4_	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5_	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6_	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7_	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8_	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9_	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA_	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB_	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC_	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD_	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE_	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF_	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0_	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1_	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2_	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3_	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4_	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5_	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6_	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7_	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8_	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9_	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA_	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB_	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC_	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED_	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE_	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF_	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0_	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1_	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2_	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3_	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4_	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5_	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6_	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7_	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8_	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9_	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA_	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB_	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC_	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD_	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE_	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF_	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Conversion Table: Hexadecimal and Decimal Integers

Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
8		7		6		5		4		3		2		1	

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of the decimal numbers corresponding to the leftmost digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position.
4. Add the numbers selected from the table to form the decimal number.

EXAMPLE

Conversion of Hexadecimal Value		D34
1. D		3328
2. 3		48
3. 4		<u>+ 4</u>
4. Decimal		3380

To convert integer numbers greater than the capacity of the table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D34_{16} = 3380_{10}$

$$\begin{array}{r}
 D = 13 \\
 \times 16 \\
 \hline
 208 \\
 3 = + 3 \\
 \hline
 211 \\
 \times 16 \\
 \hline
 3376 \\
 4 = + 4 \\
 \hline
 3380
 \end{array}$$

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.
(b) Record the hexadecimal of the column containing the selected number.
(c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1(c), repeat all of step 1 to develop the second position of the hexadecimal (and a remainder)
3. Using the remainder from step 2, repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine the terms to form the hexadecimal number.

EXAMPLE

Conversion of Decimal Value		3380
1. D		<u>-3328</u>
		52
2. 3		<u>- 48</u>
		4
3. 4		<u>- 4</u>
4. Hexadecimal		D34

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order

Example: $3380_{10} = X_{16}$

$$\begin{array}{rcl}
 16 \overline{) 3380} & \rightarrow & 4 \\
 16 \overline{) 211} & \rightarrow & 3 \\
 16 \overline{) 13} & \rightarrow & D
 \end{array}$$

Remainder ↑

$3380_{10} = D34_{16}$

POWERS OF 16 TABLE

Example: $268,435,456_{10} = (2.68435456 \times 10^8)_{10} = 1000\ 0000_{16} = (10^7)_{16}$

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7

16^n	n
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10 = A
17 592 186 044 416	11 = B
281 474 976 710 656	12 = C
4 503 599 627 370 496	13 = D
72 057 594 037 927 936	14 = E
1 152 921 504 606 846 976	15 = F
← Decimal Values	→

Conversion Table: Hexadecimal and Decimal Fractions

HALFWORD													
BYTE				BYTE									
Bits 0123		4567		0123				4567					
Hex	Decimal	Hex	Decimal	Hex	Decimal			Hex	Decimal Equivalent				
.0	.0000	.00	.0000 0000	.000	.0000	0000	0000	.0000	.0000	0000	0000	0000	0000
.1	.0625	.01	.0039 0625	.001	.0002	4414	0625	.0001	.0000	1525	8789	0625	
.2	.1250	.02	.0078 1250	.002	.0004	8828	1250	.0002	.0000	3051	7578	1250	
.3	.1875	.03	.0117 1875	.003	.0007	3242	1875	.0003	.0000	4577	6367	1875	
.4	.2500	.04	.0156 2500	.004	.0009	7656	2500	.0004	.0000	6103	5156	2500	
.5	.3125	.05	.0195 3125	.005	.0012	2070	3125	.0005	.0000	7629	3945	3125	
.6	.3750	.06	.0234 3750	.006	.0014	6484	3750	.0006	.0000	9155	2734	3750	
.7	.4375	.07	.0273 4375	.007	.0017	0898	4375	.0007	.0001	0681	1523	4375	
.8	.5000	.08	.0312 5000	.008	.0019	5312	5000	.0008	.0001	2207	0312	5000	
.9	.5625	.09	.0351 5625	.009	.0021	9726	5625	.0009	.0001	3732	9101	5625	
.A	.6250	.0A	.0390 6250	.00A	.0024	4140	6250	.000A	.0001	5258	7890	6250	
.B	.6875	.0B	.0429 6875	.00B	.0026	8554	6875	.000B	.0001	6784	6679	6875	
.C	.7500	.0C	.0468 7500	.00C	.0029	2968	7500	.000C	.0001	8310	5468	7500	
.D	.8125	.0D	.0507 8125	.00D	.0031	7382	8125	.000D	.0001	9836	4257	8125	
.E	.8750	.0E	.0546 8750	.00E	.0034	1796	8750	.000E	.0002	1362	3046	8750	
.F	.9375	.0F	.0585 9375	.00F	.0036	6210	9375	.000F	.0002	2888	1835	9375	
1		2		3				4					

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

Find .A in position 1 .6250
 Find .0B in position 2 .0429 6825
 Find .00C in position 3 + .0029 2968 7500
 .ABC is equal to .6708 9843 7500

TO CONVERT .13 DECIMAL TO HEXADECIMAL

- Find .1250 next lowest to .1300
 subtract — .1250 = .2 Hex
- Find .0039 0625 next lowest to .0050 0000
 — .0039 0625 = .01
- Find .0009 7656 2500 .0010 9375 0000
 — .0009 7656 2500 = .004
- Find .0001 0681 1523 4375 .0001 1718 7500 0000
 — .0001 0681 1523 4375 = .0007
 .0000 1037 5976 5625 = .2147 Hex
- .13 Decimal is approximately equal to _____ ↑

To convert fractions beyond the capacity of the table, use techniques below:

HEXADECIMAL TO FRACTION DECIMAL

Convert the hexadecimal fraction to its decimal equivalent using the same technique as for integer numbers. Divide the results by 16ⁿ (n is the number of fraction positions).

Example: .8A7₁₆ = .540771₁₀

$$\begin{array}{r} 8A7_{16} = 2215_{10} \\ 16^3 = 4096 \quad 4096 \overline{) 2215.000000} \end{array}$$

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: .5408₁₀ = .8A7₁₆

.5408

$$\begin{array}{r} \times 16 \\ 8 \leftarrow 8.6528 \\ \times 16 \\ A \leftarrow 10.4448 \\ \times 16 \\ 7 \leftarrow 7.1168 \end{array}$$

Hexadecimal Addition and Subtraction Table

Example: $6 + 2 = 8$, $8 - 2 = 6$, and $8 - 6 = 2$

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Hexadecimal Multiplication Table

Example: $2 \times 4 = 08$, $F \times 2 = 1E$

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Appendix I. EBCDIC and ISO-8 Codes

Dec	Hex	EBCDIC-037	ISO-8
0	00	NUL	NUL
1	01	SOH	SOH
2	02	STX	STX
3	03	ETX	ETX
4	04	SEL	EOT
5	05	HT	ENQ
6	06	RNL	ACK
7	07	DEL	BEL
8	08	GE	BS
9	09	SPS	HT
10	0A	RPT	LF
11	0B	VT	VT
12	0C	FF	FF
13	0D	CR	CR
14	0E	SO	SO
15	0F	SI	SI
16	10	DLE	DLE
17	11	DC1	DC1
18	12	DC2	DC2
19	13	DC3	DC3
20	14	RES/ENP	DC4
21	15	NL	NAK
22	16	BS	SYN
23	17	POC	ETB
24	18	CAN	CAN
25	19	EM	EM
26	1A	UBS	SUB
27	1B	CU1	ESC
28	1C	IFS	IFS
29	1D	IGS	IGS
30	1E	IRS	IRS
31	1F	ITB/IUS	IUS
32	20	DS	SP
33	21	SOS	!
34	22	FS	"
35	23	WUS	#
36	24	BYP/INP	\$
37	25	LF	%
38	26	ETB	&
39	27	ESC	'
40	28	SA	(
41	29	SFE)
42	2A	SM/SW	*
43	2B	CSP	+
44	2C	MFA	,
45	2D	ENQ	-
46	2E	ACK	.
47	2F	BEL	/
48	30		0
49	31		1
50	32	SYN	2
51	33	IR	3
52	34	PP	4
53	35	TRN	5
54	36	NBS	6
55	37	EOT	7
56	38	SBS	8
57	39	IT	9
58	3A	RFF	:
59	3B	CU3	;
60	3C	DC4	<
61	3D	NAK	=
62	3E		>
63	3F	SUB	?

Dec	Hex	EBCDIC-037	ISO-8
64	40	SP	@
65	41	RSP	A
66	42	â	B
67	43	ä	C
68	44	à	D
69	45	á	E
70	46	â	F
71	47	ä	G
72	48	ç	H
73	49	ñ	I
74	4A	ç	J
75	4B	.	K
76	4C	<	L
77	4D	(M
78	4E	+	N
79	4F	l	O
80	50	&	P
81	51	é	Q
82	52	ê	R
83	53	è	S
84	54	è	T
85	55	í	U
86	56	î	V
87	57	ï	W
88	58	ì	X
89	59	ß	Y
90	5A	!	Z
91	5B	\$	[
92	5C	*	\
93	5D)]
94	5E	;	^
95	5F	¬	_
96	60	-	`
97	61	/	a
98	62	Â	b
99	63	Ã	c
100	64	Ä	d
101	65	Å	e
102	66	Ä	f
103	67	Å	g
104	68	Ç	h
105	69	Ñ	i
106	6A	!	j
107	6B	.	k
108	6C	%	l
109	6D	_	m
110	6E	>	n
111	6F	?	o
112	70	ø	p
113	71	É	q
114	72	Ê	r
115	73	Ë	s
116	74	Ë	t
117	75	Í	u
118	76	Î	v
119	77	Ï	w
120	78	Ì	x
121	79	'	y
122	7A	:	z
123	7B	#	{
124	7C	@	
125	7D	'	}
126	7E	=	~
127	7F	"	-

Dec	Hex	EBCDIC-037	ISO-8
128	80	Ø	
129	81	a	
130	82	b	BPH
131	83	c	NBH
132	84	d	IND
133	85	e	NEL
134	86	f	SSA
135	87	g	ESA
136	88	h	HTS
137	89	i	HTJ
138	8A	«	VTS
139	8B	»	PLD
140	8C	ð	PLU
141	8D	ý	RI
142	8E	þ	SS2
143	8F	±	SS3
144	90	°	DCS
145	91	j	PU1
146	92	k	PU2
147	93	l	STS
148	94	m	CCH
149	95	n	MW
150	96	o	SPA
151	97	p	EPA
152	98	q	SOS
153	99	r	
154	9A	ª	SCI
155	9B	º	CSI
156	9C	æ	ST
157	9D	.	OSC
158	9E	Æ	PM
159	9F	α	APC
160	A0	μ	RSP
161	A1	~	¡
162	A2	s	¢
163	A3	t	£
164	A4	u	¤
165	A5	v	¥
166	A6	w	¦
167	A7	x	§
168	A8	y	¨
169	A9	z	©
170	AA	¡	ª
171	AB	¿	«
172	AC	Ð	¬
173	AD	Ý	SHY
174	AE	þ	®
175	AF	®	¯
176	B0	^	°
177	B1	£	±
178	B2	¥	²
179	B3	.	³
180	B4	©	´
181	B5	§	µ
182	B6	¶	¶
183	B7	¼	·
184	B8	½	¸
185	B9	¾	¹
186	BA	[º
187	BB]	»
188	BC	~	¼
189	BD	¨	½
190	BE	'	¾
191	BF	x	¿

Dec	Hex	EBCDIC-037	ISO-8
192	C0	{	À
193	C1	A	Á
194	C2	B	Â
195	C3	C	Ã
196	C4	D	Ä
197	C5	E	Å
198	C6	F	Æ
199	C7	G	Ç
200	C8	H	È
201	C9	I	É
202	CA	SHY	Ê
203	CB	ð	Ë
204	CC	ö	Ì
205	CD	ö	Í
206	CE	ó	Î
207	CF	ö	Ï
208	D0	}	Ð
209	D1	J	Ñ
210	D2	K	Ò
211	D3	L	Ó
212	D4	M	Ô
213	D5	N	Õ
214	D6	O	Ö
215	D7	P	×
216	D8	Q	Ø
217	D9	R	Ù
218	DA	'	Ú
219	DB	û	Û
220	DC	ü	Ü
221	DD	ü	Ý
222	DE	ú	Þ
223	DF	ÿ	ß
224	E0	\	à
225	E1	÷	á
226	E2	S	â
227	E3	T	ã
228	E4	U	ä
229	E5	V	å
230	E6	W	æ
231	E7	X	ç
232	E8	Y	è
233	E9	Z	é
234	EA	²	ê
235	EB	Ô	ë
236	EC	Ö	ì
237	ED	Ò	í
238	EE	Ó	î
239	EF	Ô	ï
240	F0	0	ð
241	F1	1	ñ
242	F2	2	ò
243	F3	3	ó
244	F4	4	ô
245	F5	5	õ
246	F6	6	ö
247	F7	7	÷
248	F8	8	ø
249	F9	9	ù
250	FA	ª	ú
251	FB	Û	û
252	FC	Ü	ü
253	FD	Ü	ý
254	FE	Ú	þ
255	FF	EO	ÿ

Control Character Representations

ACK	Acknowledge	ENQ	Enquiry	ITB	Intermediate Transmission Block	SEL	Select
BEL	Bell	EO	Eight Ones	IUS	International Unit Separator	SFE	Start Field Extended
BS	Backspace	EOT	End of Transmission	LF	Line Feed	SI	Shift In
BYP	Bypass	ESC	Escape	MFA	Modify Field Attribute	SM	Set Mode
CAN	Cancel	ETB	End of Transmission Block	NAK	Negative Acknowledge	SO	Shift Out
CR	Carriage Return	ETX	End of Text	NBS	Numeric Backspace	SOH	Start of Heading
CSP	Control Sequence Prefix	FF	Form Feed	NL	New Line	SOS	Start of Significance
CU1	Customer Use 1	FS	Field Separator	NUL	Null	SPS	Superscript
CU3	Customer Use 3	GE	Graphic Escape	POC	Program-Operator Communication	STX	Start of Text
DC1	Device Control 1	HT	Horizontal Tab	PP	Presentation Position	SUB	Substitute
DC2	Device Control 2	IFS	Interchange File Separator	RES	Restore	SW	Switch
DC3	Device Control 3	IGS	Interchange Group Separator	RFF	Required Form Feed	SYN	Synchronous Idle
DC4	Device Control 4	INP	Inhibit Presentation	RNL	Required New Line	TRN	Transparent
DEL	Delete	IR	Index Return	RPT	Repeat	UBS	Unit Backspace
DLE	Data Link Escape	IRS	Interchange Record Separator	SA	Set Attribute	VT	Vertical Tab
DS	Digit Select	IT	Indent Tab	SBS	Subscript	WUS	Word Underscore
EM	End of Medium						
ENP	Enable Presentation						

Formatting Character Representations

NSP	Numeric Space	RSP	Required Space	SP	Space	SHY	Syllable Hyphen
-----	---------------	-----	----------------	----	-------	-----	-----------------

Additional ISO-8 Control Character Representations

APC	Application Program Command	HTJ	Character Tabulation with Justification	OSC	Operating System Command	SPA	Start of Guarded Area
BPH	Break Permitted Here	HTS	Character Tabulation Set	PLD	Partial Line Down	SSA	Start of Selected Area
CCH	Cancel Character	IFS	Information Separator Four	PLU	Partial Line Up	SS2	Single Shift Two
CSI	Control Sequence Introducer	IGS	Information Separator Three	PM	Privacy Message	SS3	Single Shift Three
DCS	Device Control String	IND	Index	PU1	Private Use One	ST	String Terminator
EPA	End of Guarded Area	IRS	Information Separator Two	PU2	Private Use Two	STS	Set Transmit State
ESA	End of Selected Area	MW	Message Waiting	RI	Reverse linefeed (or index)	US	Information Separator One
		NBH	No Break Here	SCI	Single Character Introducer	VT	Line Tabulation Set
		NEL	Next Line	SOS	Start of String		

Note: The ISO-8 controls are from ISO 6429, and the graphics are from ISO 8859-1. The ISO-8 graphics are code page 00819, named ISO/ANSI Multilingual.

Index

Numerics

2K-IDAW control 15-28
370-XA architecture 1-25

A

A (ADD) binary instruction 7-25
absolute address 3-4
absolute storage 3-4
access-control bits
 in segment-table entry 3-42
access-control bits in storage key 3-8
access exceptions 6-40, 6-46
 priority of 6-46
 recognition of 6-40
access key 3-10
 for channel-program execution 3-10, 15-25, 15-30
 for channel-subsystem monitoring 3-10
 for CPU 3-10
access list 5-53
 See also access-list entry
 accessing capability, revocation of 5-48
 allocation and invalidation of entries in 5-46
 authorizing the use of entries in 5-46
 concepts 5-44
 designation (ALD) 5-53
 length (ALL) 5-53
 origin (ALO) 5-53
access-list-controlled protection 3-12
 exception for 6-30
access-list designation 5-53
access-list entry (ALE) 5-53
 authorization index (ALEAX) 5-53
 number
 See ALEN
 sequence exception 6-19
 as an access exception 6-40
 sequence number (ALESN)
 in ALE 5-53
 in ALET 5-51
 token
 See ALET
access-register mode 3-36
access-register translation (ART) 5-50
 as part of LOAD REAL ADDRESS, STORE REAL ADDRESS, TEST ACCESS, and TEST PROTECTION 5-55
 introduction to 5-44
 lookaside buffer
 See ALB
 sequence of table fetches 5-90
access-register-translation (ART) tables 5-52
access registers 2-5
 designation of 5-43
 functions of 5-43
 instructions for use of 5-49
 save areas for 3-68
 validity bit for 11-20
access to storage 5-85
 See also reference
active
 device 16-16, 16-36
 subchannel 16-16, 16-36
active allegiance 15-13
active communication 15-13
activity-control field (SCSW) 16-14, 16-36
 following TEST SUBCHANNEL 14-21
AD (ADD NORMALIZED) HFP instruction 18-8
 example A-41
adaptive dictionary 7-155
ADB (ADD) BFP instruction 19-11
ADBR (ADD) BFP instruction 19-11
ADD BFP instructions 19-11
ADD binary instructions 7-24
ADD DECIMAL instruction 8-6
 example A-36
ADD DFP instructions 20-19
ADD HALFWORD IMMEDIATE instruction 7-27
ADD HALFWORD instruction 7-27
 example A-8
ADD HIGH binary instructions 7-27
ADD IMMEDIATE binary instructions 7-25
ADD IMMEDIATE HIGH binary instructions 7-28
ADD LOGICAL HIGH binary instructions 7-29
ADD LOGICAL IMMEDIATE instructions 7-28
ADD LOGICAL instructions 7-28
ADD LOGICAL WITH CARRY instructions 7-29
ADD LOGICAL WITH SIGNED IMMEDIATE HIGH binary instructions 7-31
ADD LOGICAL WITH SIGNED IMMEDIATE instructions 7-30
ADD NORMALIZED HFP instructions 18-8
 example A-41
ADD UNNORMALIZED HFP instructions 18-9
 example A-42
additional floating-point (AFP) registers 9-9
address 3-2
 24-bit, 31-bit, and 64-bit 3-5
 in branch-address generation 5-10

- in operand address generation 5-9
- absolute 3-4
- AR-specified virtual
 - See AR-specified
- arithmetic 3-5, 5-9
 - unsigned binary 7-7
- backward stack-entry 5-76
- base
 - See base address
- branch
 - See branch address
- channel-program
 - See channel-program address
- comparison 12-1
 - controls for 12-1
 - effect on CPU state 4-2
- CPU
 - See CPU address
- data (I/O)
 - See data address
- effective
 - See effective address
- failing-storage
 - See failing-storage address
- format 3-3
- forward-section header 5-77
- generation 5-8
 - for storage addressing 3-6
- home virtual
 - See home virtual address
- I/O 13-5
- instruction
 - See instruction address
- invalid 6-17
- logical
 - See logical address
- numbering of for byte locations 3-2
- PER
 - See PER address
- prefixing
 - See prefix
- primary virtual
 - See primary virtual address
- real 3-4
- secondary virtual
 - See secondary virtual address
- size of
 - controlled by addressing mode 5-8
- storage 3-2
- summary information 3-58
- translation
 - See dynamic address translation, prefix
- types 3-4
- virtual 3-5

- wraparound
 - See wraparound
- address-limit checking (I/O)
 - effect of I/O-system reset on 17-15
 - limit mode (bits in PMCW) 15-3
- address-limit-checking control (I/O) 15-27, 16-12
 - used for IPL 17-17
- address of] 15-35
- address space 3-18
 - AR-specified 5-43
 - changing of 3-19
 - control bits
 - control bit 5-71
 - in the PSW 4-6
 - use in address translation 3-36
 - created by DAT 3-34
 - number
 - See ASN
- address-space-control element (ASCE) 3-38
 - effective 3-45
 - home 3-40
 - in AST entry 3-28
 - used for access-register translation 5-55
 - used for subspace groups 5-64
 - primary 3-38
 - secondary 3-39
 - type exception 6-19
- addressing
 - trimodal 7-9
- address-and-translation-mode identification (ATMID) 4-26
- addressing exception 6-17
 - as an access exception 6-40, 6-46
- addressing mode 5-8
 - bit in linkage-stack state entry 5-78
 - bit in the PSW 4-6
 - effect on address size 3-6
 - effect on operand-address generation 5-9
 - effect on sequential instruction-address generation 5-8
 - effect on wraparound 3-6
 - in branch-address generation 5-10
 - in examples A-7
 - in operand address generation 5-9
 - set by BRANCH AND SAVE AND SET MODE instruction 7-35
 - set by BRANCH AND SET MODE instruction 7-36
 - set by SET ADDRESSING MODE instruction 7-295
 - use of 5-15
- ADR (ADD NORMALIZED) HFP instruction 18-8
- ADTR (ADD) DFP instruction 20-19
- AE (ADD NORMALIZED) HFP instruction 18-8

- example A-41
- AEB (ADD) BFP instruction 19-11
- AEBR (ADD) BFP instruction 19-11
- AER (ADD NORMALIZED) HFP instruction 18-8
- AFI (ADD IMMEDIATE) instruction 7-25, 7-28, 7-31
- AFP (additional floating-point) registers 9-9
- AFP-register data exception 6-22
- AFT (ASN first table) 3-27
- AFTE (ASN-first-table entry) 3-27
- AFTO (ASN-first-table origin) 3-27
- AFX (ASN-first-table index) 3-26
 - invalid bit 3-27
 - translation exception 6-18
- AG (ADD) binary instruction 7-25
- AGF (ADD) binary instruction 7-25
- AGFI (ADD IMMEDIATE) instruction 7-25
- AGFR (ADD) binary instruction 7-25
- AGHI (ADD HALFWORD IMMEDIATE) instruction 7-27
- AGHIK (ADD IMMEDIATE) instruction 7-25
- AGR (ADD) binary instruction 7-25
- AGRK (ADD) binary instruction 7-25
- AGSI (ADD IMMEDIATE) instruction 7-25
- AH (ADD HALFWORD) instruction 7-27
 - example A-8
- AHHHR (ADD HIGH) binary instruction 7-27
- AHI (ADD HALFWORD IMMEDIATE) instruction 7-27
- AHIK (ADD IMMEDIATE) instruction 7-25
- AHY (ADD HALFWORD) instruction 7-27
- AIH (ADD IMMEDIATE HIGH) binary instruction 7-28, 7-31
- AIH (ADD LOGICAL WITH SIGNED IMMEDIATE HIGH) instruction 7-31
- AKM (authorization key mask) 5-34
- AL (ADD LOGICAL) instruction 7-28
- ALB (ART-lookaside buffer) 5-60
 - entry
 - clearing of 5-62
 - effect of translation changes on 5-61
 - usable state 5-61
- ALC (ADD LOGICAL WITH CARRY) instruction 7-29
- ALCG (ADD LOGICAL WITH CARRY) instruction 7-29
- ALCGR (ADD LOGICAL WITH CARRY) instruction 7-29
- ALCR (ADD LOGICAL WITH CARRY) instruction 7-29
- ALD (access-list designation) 5-53
- ALE
 - See access-list entry
- ALEAX (access-list-entry authorization index) 5-53
- ALEN (access-list-entry number) 5-51
 - invalid bit 5-53
 - translation exception 6-18
 - as an access exception 6-40
- alert (class of machine-check condition) 11-10
- alert interruption condition (I/O) 16-5
- alert-status bit (I/O) 16-17
- ALESN (access-list-entry sequence number)
 - in ALE 5-53
 - in ALET 5-51
- ALET (access-list-entry token) 5-45, 5-51
 - specification exception 6-19
 - as an access exception 6-40
- ALFI (ADD LOGICAL IMMEDIATE) instruction 7-28
- ALG (ADD LOGICAL) instruction 7-28
- ALGF (ADD LOGICAL) instruction 7-28
- ALGFI (ADD LOGICAL IMMEDIATE) instruction 7-28
- ALGFR (ADD LOGICAL) instruction 7-28
- ALGHSIK (ADD LOGICAL WITH SIGNED IMMEDIATE) instruction 7-30
- ALGR (ADD LOGICAL) instruction 7-28
- ALGRK (ADD) binary instruction 7-28
- ALGSI (ADD LOGICAL WITH SIGNED IMMEDIATE) instruction 7-30
- ALHHHR (ADD LOGICAL HIGH) binary instruction 7-29
- ALHSIK (ADD LOGICAL WITH SIGNED IMMEDIATE) instruction 7-30
- alignment 3-3
- ALL (access-list length) 5-53
- allegiance
 - active 15-13
 - channel-path 15-12
 - dedicated 15-13
 - effect on CLEAR SUBCHANNEL of 15-12
 - working 15-12
- allowed interruptions 6-5
- ALO (access-list origin) 5-53
- ALR (ADD LOGICAL) instruction 7-28
- ALRK (ADD) binary instruction 7-28
- ALSI (ADD LOGICAL WITH SIGNED IMMEDIATE) instruction 7-30
- ALSIHN (ADD LOGICAL WITH SIGNED IMMEDIATE HIGH) binary instruction 7-31
- alter-and-display controls 12-2
- alteration
 - storage (PER event) 4-32
- ALY (ADD LOGICAL) instruction 7-28
- ancillary-report bit
 - in channel-report word 17-26
 - in machine-check-interruption code 11-17
 - in subchannel logout 16-49
- AND IMMEDIATE instructions 7-32
- AND instructions 7-31
 - examples A-8

- AP (ADD DECIMAL) instruction 8-6
 - example A-36
- AR (ADD) binary instruction 7-24
- AR-specified (access-register-specified) 3-5, 3-19
 - address space 3-19, 5-43
 - virtual address 3-5
- AR-specified (access-register-specified) virtual address
 - effective address-space-control element for 3-45
- architectural mode
 - identification 3-63
 - indication of 12-2
 - selection of by IML controls 12-3
 - selection of by manual controls 12-2
 - selection of by signal-processor order 4-60
- architecture
 - compatibility 1-26
- arithmetic
 - address
 - See* address arithmetic
 - binary 7-6
 - examples A-2
 - decimal 8-3
 - examples A-4, A-36
 - floating-point 9-2
 - examples A-5, A-41
 - logical (unsigned binary) 7-7
 - examples A-3
- ARK (ADD) binary instruction 7-25
- ART
 - See* access-register translation
- ART-lookaside buffer
 - See* ALB
- ASCE
 - See* address-space-control element
- ASCE (address-space-control element)
 - type exception 6-19
- ASCII character code
 - handled by architecture xxvii
- ASI (ADD IMMEDIATE) instruction 7-25
- ASN (address-space number) 3-20
 - AST (ASN second table) 3-27
 - authorization 3-31
 - first table (AFT) 3-27
 - first-table index
 - See* AFX
 - first-table origin (AFTO) 3-27
 - in entry-table entry 5-34
 - second-table (AST) 3-27
 - second-table address in ETE 5-35
 - second-table entry (ASTE)
 - for subspace groups 5-64
 - instance exception 6-20
 - instance number (ASTEIN), used in ASN reuse 3-22
 - origin, in ALE 5-54
 - primary (PASTE) 5-32
 - pseudo 3-20
 - sequence exception 6-20
 - sequence exception as an access exception 6-40
 - sequence number (ASTESN), in ALE 5-54
 - sequence number (ASTESN), in ASTE 5-55
 - validity exception 6-20
 - validity exception as an access exception 6-40
- second-table index
 - See* ASX
- second-table origin (ASTO) 3-27
- trace-control bit 4-12
- translation 3-26
 - exceptions 6-49
 - translation-control bit 3-27, 5-26
- ASN-and-LX-reuse-control bit 3-22
- ASN-and-LX-reuse facility 1-7, 3-21
 - facility indication 4-67
- assembler language A-7
 - instruction formats in
 - See* instruction lists and page numbers in Appendix B
- assigned storage locations 3-61
- AST (ASN second table) 3-27
- AST entry 5-54
 - See also* ASN-second-table entry
- ASTE
 - See* ASN-second-table entry
- ASTESN (AST-entry sequence number)
 - in ALE 5-54
 - in ASTE 5-55
- ASTO (ASN-second-table origin) 3-27
- ASX (ASN-second-table index) 3-26
 - invalid bit 3-28
 - use in ART 5-54
 - translation exception 6-21
- AT
 - See* authority table
- ATL (authority-table length) 3-28
 - use in ART 5-55
- ATMID (addressing-and-translation-mode identification) 4-26
- ATO (authority-table origin) 3-28
 - use in ART 5-54
- attached ART-table entry 5-61
- attached region-table, segment-table, or page-table entry 3-53
- attachment of I/O devices 13-3
- AU (ADD UNNORMALIZED) HFP instruction 18-9
 - example A-42

- AUR (ADD UNNORMALIZED) HFP instruction 18-9
- authority table (AT) 5-26
 - designation 3-28, 5-54
 - length 3-28, 5-55
 - origin 3-28, 5-54
- authorization
 - ASN 3-31
 - index (AX) 3-31, 5-26
 - key mask (AKM) 5-34
 - mechanisms 5-24
 - summary of 5-27
 - testing of 5-70
- authorization check 16-52
 - indicator for (in ERW) 16-52
- automatic reconfiguration 1-24
- auxiliary storage 3-2, 3-34
- availability (characteristic of a system) 1-27
- AW (ADD UNNORMALIZED) HFP instruction 18-9
- AWR (ADD UNNORMALIZED) HFP instruction 18-9
- AX (authorization index) 3-31, 5-26
- AXBR (ADD) BFP instruction 19-11
- AXR (ADD NORMALIZED) HFP instruction 18-8
- AY (ADD) binary instruction 7-25

B

- B field of instruction 5-8
- backed-up bit (machine-check condition) 11-16
- backup
 - processing (synchronous machine-check condition) 11-17
- backward stack-entry address 5-76
- backward stack-entry validity bit 5-76
- BAKR (BRANCH AND STACK) instruction 10-10
 - examples A-10
- BAL (BRANCH AND LINK) instruction 7-33
 - examples A-8
- BALR (BRANCH AND LINK) instruction 7-33
 - examples A-8
- BAS (BRANCH AND SAVE) instruction 7-34
 - example A-8
- base address 5-8
 - register for 2-3
- base-AST-entry origin (BASTEO) 5-62
- base-authority state 10-7
- base space 5-14
- base-space bit 5-64
- basic addressing mode
 - bit in entry-table entry 5-34
- basic I/O functions 15-1
- basic PROGRAM CALL 5-67, 10-84
- BASR (BRANCH AND SAVE) instruction 7-34
 - example A-8

- BASSM (BRANCH AND SAVE AND SET MODE)
 - instruction 7-35
 - example A-8
- BASTEO (base-AST-entry origin) 5-62
- BC (BRANCH ON CONDITION) instruction 7-36
 - example A-12
- BCR (BRANCH ON CONDITION) instruction 7-36
- BCT (BRANCH ON COUNT) instruction 7-37
 - example A-12
- BCTG (BRANCH ON COUNT) instruction 7-37
- BCTGR (BRANCH ON COUNT) instruction 7-37
- BCTR (BRANCH ON COUNT) instruction 7-37
 - example A-12
- BFP
 - data class
 - testing of 19-39
- BFP (binary floating point) 9-4
- BFP data 19-2
 - conversion of 9-27
- BFP facility 19-1
- BFP-instruction data exception 6-22
- bias for exponent 9-2, 19-2, 20-3
 - function of view 9-2
 - summary for BFP 19-2, 19-3
 - summary for DFP 20-3, 20-6
- biased exponent 9-2, 19-2, 20-3
- big endian 7-231, 7-311
- bimodal addressing
 - See* addressing mode
- binary
 - See also* fixed point
 - arithmetic 7-6
 - examples A-2
 - negative zero 7-5
 - number representation 7-5
 - examples A-2
 - overflow 7-6
 - example A-2
 - See also* fixed point
 - sign bit 7-5
- binary floating point (BFP) 9-4
- binary integer
 - conversion from floating point 18-11, 19-21, 19-23, 20-27, 20-29
 - conversion to floating point 18-10, 19-18, 20-24, 20-25
- binary-to-decimal conversion 7-182
 - example A-19
- bit 3-2
 - numbering of within a group of bytes 3-3
- bits to left of compressed data 7-151
- bits to right of compressed data 7-151
- block-concurrent storage references 5-97
- block number

- expanded storage 2-2
- block of I/O data 15-24
- block of storage 3-4
 - See also* page
 - testing for usability of 10-153
- borrow 7-314
- boundary alignment 3-3
 - for instructions 5-3
- branch address 5-10
 - control bit 4-24
 - in linkage-stack state entry 5-79
 - in trace entry 4-19
- BRANCH AND LINK instructions 7-33
 - examples A-8
- BRANCH AND SAVE AND SET MODE instruction 7-35
 - examples A-8
- BRANCH AND SAVE instructions 7-34
 - examples A-8
- BRANCH AND SET AUTHORITY instruction 10-6
- BRANCH AND SET MODE instruction 7-36
 - examples A-8
- BRANCH AND STACK instruction 10-10
 - examples A-10
- BRANCH IN SUBSPACE GROUP instruction 10-13
- BRANCH ON CONDITION instructions 7-36
 - example A-12
- BRANCH ON COUNT instructions 7-37
 - example A-12
- BRANCH ON INDEX HIGH instruction 7-38
 - examples A-12
- BRANCH ON INDEX LOW OR EQUAL instruction 7-38
 - examples A-13
- BRANCH RELATIVE AND SAVE instruction 7-39
- BRANCH RELATIVE AND SAVE LONG instruction 7-39
- BRANCH RELATIVE ON CONDITION instruction 7-40
- BRANCH RELATIVE ON CONDITION LONG instruction 7-40
- BRANCH RELATIVE ON COUNT HIGH instruction 7-41
- BRANCH RELATIVE ON COUNT instruction 7-41
- BRANCH RELATIVE ON INDEX HIGH instruction 7-41
- BRANCH RELATIVE ON INDEX LOW OR EQUAL instruction 7-41
- branch state entry 5-77, 10-11
- branch-trace-control bit 4-12
- branching
 - branch-address generation 5-10
 - in a channel program 15-71
 - relative 5-10

- to perform decision making, loop control, and subroutine linkage 5-11
- using the linkage stack 5-69
- BRAS (BRANCH RELATIVE AND SAVE) instruction 7-39
- BRASL (BRANCH RELATIVE AND SAVE LONG) instruction 7-39
- BRC (BRANCH RELATIVE ON CONDITION) instruction 7-40
- BRCL (BRANCH RELATIVE ON CONDITION LONG) instruction 7-40
- BRCT (BRANCH RELATIVE ON COUNT) instruction 7-41
- BRCTG (BRANCH RELATIVE ON COUNT) instruction 7-41
- BRCTH (BRANCH RELATIVE ON COUNT HIGH) instruction 7-41
- breaking-event address
 - assigned storage locations for 3-66
- breaking-event-address register 4-37
- broadcasted-purging facility 1-20
- BRXH (BRANCH RELATIVE ON INDEX HIGH) instruction 7-41
- BRXHG (BRANCH RELATIVE ON INDEX HIGH) instruction 7-41
- BRXLE (BRANCH RELATIVE ON INDEX LOW OR EQUAL) instruction 7-41
- BRXLG (BRANCH RELATIVE ON INDEX LOW OR EQUAL) instruction 7-42
- BSA (BRANCH AND SET AUTHORITY) instruction 10-6
- BSG (BRANCH IN SUBSPACE GROUP) instruction 10-13
- BSM (BRANCH AND SET MODE) instruction 7-36
 - example A-8
- buffer storage (cache) 3-2
- burst mode (channel-path operation) 13-3
- busy
 - in I/O operations 13-10
 - in SIGNAL PROCESSOR 4-62
- BX (byte index) 3-35
- BXH (BRANCH ON INDEX HIGH) instruction 7-38
 - examples A-12
- BXHG (BRANCH ON INDEX HIGH) instruction 7-38
- BXLE (BRANCH ON INDEX LOW OR EQUAL) instruction 7-38
 - examples A-13
- BXLEG (BRANCH ON INDEX LOW OR EQUAL) instruction 7-38
- byte 3-2
 - numbering of in storage 3-2
- byte index (BX) 3-35
- byte-multiplex mode (channel-path operation) 13-3

C

C (COMPARE) binary instruction 7-112

cache 3-2

called-space identification 5-79

cancel-I/O facility 1-20

CANCEL SUBCHANNEL instruction 14-4

candidates

selection of 9-13

canonical

declefs 20-6

canonicalization

of DFP data 20-7

capability list 5-48

carry 7-6

CBC (checking-block code) 11-2

invalid 11-2

in registers 11-8

in storage 11-6

in storage keys 11-7

near-valid 11-2

valid 11-2

CBN (compressed-data bit number) 7-149

example of 7-154

CCC (channel-control check) 16-29

CCW (channel-command word) 15-31

address of 16-19

byte count in 15-33

check (in subchannel logout) 16-48

command codes

See commands

contents of 15-31

current 15-31

designation of storage area in 15-32

format-0 and format-1 15-31

format control 16-11, 16-12, 16-35, 16-36

used for IPL 17-17

IDA flag in 15-32

indirect data addressing used in 13-9, 15-61

invalid format of 16-27

invalid specification of 16-25, 16-40

MIDA flag in 15-33

modified indirect data addressing used in 15-63

PCI flag in 15-32

prefetch control in 16-11

used for IPL 17-17

prefetching 15-57

retry of

See command retry

skip flag in 15-32

suspend flag in 15-32

CCW channel program 13-7

CCW-type IPL (CCW-type initial program loading)

17-17

CD (COMPARE) HFP instruction 18-10

CDB (COMPARE) BFP instruction 19-16

CDBR (COMPARE) BFP instruction 19-16

CDFBR (CONVERT FROM FIXED) BFP instruction
19-18, 19-20

CDFBRA (CONVERT FROM FIXED) BFP instruction
19-18

CDFR (CONVERT FROM FIXED) HFP instruction
18-10

CDFTR (CONVERT FROM FIXED) DFP instruction
20-24

CDGBR (CONVERT FROM FIXED) BFP instruction
19-18, 19-20

CDGBRA (CONVERT FROM FIXED) BFP instruction
19-18

CDGR (CONVERT FROM FIXED) HFP instruction
18-10

CDGTR (CONVERT FROM FIXED) DFP instruction
20-24

CDGTRA (CONVERT FROM FIXED) DFP instruction
20-24

CDLFBR (CONVERT FROM LOGICAL) BFP
instruction 19-20

CDLFTR (CONVERT FROM LOGICAL) DFP
instruction 20-25

CDLGBR (CONVERT FROM LOGICAL) BFP
instruction 19-20

CDLGTR (CONVERT FROM LOGICAL) DFP
instruction 20-25

CDR (COMPARE) HFP instruction 18-10
examples A-42

CDS (COMPARE DOUBLE AND SWAP) instruction
7-122

examples A-46

CDSG (COMPARE DOUBLE AND SWAP) instruction
7-122

CDSS (compressed-data symbol size) 7-150

CDSTR (CONVERT FROM SIGNED PACKED) DFP
instruction 20-26

CDSY (COMPARE DOUBLE AND SWAP) instruction
7-122

CDTR (COMPARE) DFP instruction 20-21

CDUTR (CONVERT FROM UNSIGNED PACKED)
DFP instruction 20-27

CE (COMPARE) HFP instruction 18-10

CEB (COMPARE) BFP instruction 19-16

CEBR (COMPARE) BFP instruction 19-16

CEDTR (COMPARE BIASED EXPONENT) DFP
instruction 20-23

CEFBR (CONVERT FROM FIXED) BFP instruction
19-18, 19-20

CEFBRA (CONVERT FROM FIXED) BFP instruction
19-18

CEFR (CONVERT FROM FIXED) HFP instruction
18-10

CEGBR (CONVERT FROM FIXED) BFP instruction 19-18, 19-20

CEGBRA (CONVERT FROM FIXED) BFP instruction 19-18

CEGR (CONVERT FROM FIXED) HFP instruction 18-10

CELFBR (CONVERT FROM LOGICAL) BFP instruction 19-20

CELGBR (CONVERT FROM LOGICAL) BFP instruction 19-20

central processing unit
See CPU

CER (COMPARE) HFP instruction 18-10

CEXTR (COMPARE BIASED EXPONENT) DFP instruction 20-23

CFC (COMPARE AND FORM CODEWORD) instruction 7-115
 example A-53

CFDBR (CONVERT TO FIXED) BFP instruction 19-21

CFDBRA (CONVERT TO FIXED) BFP instruction 19-21

CFDR (CONVERT TO FIXED) HFP instruction 18-11

CFEBR (CONVERT TO FIXED) BFP instruction 19-21

CFEBRA (CONVERT TO FIXED) BFP instruction 19-21

CFER (CONVERT TO FIXED) HFP instruction 18-11

CFI (COMPARE IMMEDIATE) instruction 7-112

CFXBR (CONVERT TO FIXED) BFP instruction 19-21

CFXBRA (CONVERT TO FIXED) BFP instruction 19-21

CFXR (CONVERT TO FIXED) HFP instruction 18-11

CG (COMPARE) binary instruction 7-112

CGDBR (CONVERT TO FIXED) BFP instruction 19-21

CGDBRA (CONVERT TO FIXED) BFP instruction 19-21

CGDR (CONVERT TO FIXED) HFP instruction 18-11

CGDTR (CONVERT TO FIXED) DFP instruction 20-27

CGEBR (CONVERT TO FIXED) BFP instruction 19-21

CGEBRA (CONVERT TO FIXED) BFP instruction 19-21

CGER (CONVERT TO FIXED) HFP instruction 18-11

CGF (COMPARE) binary instruction 7-112

CGFI (COMPARE IMMEDIATE) instruction 7-112

CGFR (COMPARE) binary instruction 7-112

CGFRL (COMPARE RELATIVE LONG) instruction 7-112

CGH (COMPARE HALFWORD) binary instruction 7-128

CGHI (COMPARE HALFWORD IMMEDIATE) instruction 7-128

CGHRL (COMPARE HALFWORD RELATIVE LONG) binary instruction 7-128

CGHSI (COMPARE HALFWORD IMMEDIATE) instruction 7-128

CGIB (COMPARE IMMEDIATE AND BRANCH) instruction 7-113

CGIJ (COMPARE IMMEDIATE AND BRANCH RELATIVE) instruction 7-114

CGIT (COMPARE IMMEDIATE AND TRAP) instruction 7-127

CGR (COMPARE) binary instruction 7-112

CGRB (COMPARE AND BRANCH) instruction 7-113

CGRJ (COMPARE AND BRANCH RELATIVE) instruction 7-113

CGRL (COMPARE RELATIVE LONG) binary instruction 7-112

CGRT (COMPARE AND TRAP) instruction 7-127

CGXBR (CONVERT TO FIXED) BFP instruction 19-21

CGXBRA (CONVERT TO FIXED) BFP instruction 19-21

CGXR (CONVERT TO FIXED) HFP instruction 18-11

CGXTR (CONVERT TO FIXED) DFP instruction 20-27, 20-29

CH (COMPARE HALFWORD) instruction 7-128
 example A-14

chaining check (subchannel status) 16-30

chaining of CCWs 15-58
 command
See command chaining of CCWs
 data
See data chaining of CCWs

chaining of CRWs 17-25, 17-26

change bit in storage key 3-9

change recording 3-16

change-recording override 3-17

change-bit override
 bit for in region-table entry 3-43

channel-command word
See CCW

channel commands
See commands (I/O)

channel-control check (subchannel status) 16-29

channel-data check (subchannel status) 16-28, 16-35, 16-44

channel path 13-3
 active allegiance for 15-13
 available for selection 15-13
 dedicated allegiance for 15-13

- effect of I/O-system reset on 17-14
- masks in SCHIB
 - See* LPM, LPUM, PAM, PIM, PNOM, POM
- multipath mode of 15-4, 15-23
- not operational 16-13
- parallel-I/O-interface type 13-3
- serial-I/O-interface type 13-3
- storing of status for 14-17
- type of 13-3, 13-6
- working allegiance for 15-12
- channel-path identifier
 - See* CHPID
- channel-path reset 17-12
 - effect of I/O-system reset on 17-15
- channel-path-reset function 15-75
 - completion of 15-76
 - initiation by RESET CHANNEL PATH 14-9
 - reset signal issued as part of 17-12
 - signaling for 15-76
- channel-path-status word 14-17
- channel-path timeout
 - indicator for (in ERW) 16-52
- channel program 15-31
 - 2K-IDAW control for 15-28
 - branching in
 - See* TIC
 - CCW channel program 13-7
 - execution of 13-11, 15-23
 - resumption of 14-11
 - suspension of 13-11, 15-68
 - format-2-IDAW control for 15-27
 - modification control for 15-26
 - serialization 5-102
 - streaming-mode control for 15-25
 - suspend control for 15-25
 - synchronization control for 15-26
 - TCW channel program 13-7
- channel-program address 16-19
 - field-validity flag for in IRB 16-49
 - used for IPL 17-17
- channel-program type
 - CCW channel program 13-7
 - TCW channel program 13-7
- channel-program-type control 15-27
- channel report 17-25
 - generated as a result of RCHP 14-10
- channel report pending 11-16, 17-25
 - effect of I/O-system reset on 17-15
 - subclass-mask bit for 11-23
- channel-report word
 - See* CRW
- channel subsystem 2-5, 13-1
 - addressing used in 13-5
 - damage 11-16
 - effect of I/O-system reset on 17-13
 - effect of power-on reset on 4-56
 - isolated state of 16-52
- channel-subsystem-call facility 1-24
- channel-subsystem monitoring 17-1
 - effect of I/O-system reset on 17-15
- channel-subsystem recovery 11-4, 17-24
- channel-subsystem timer 17-2
 - effect of I/O-system reset on 17-15
- channel-subsystem timing 17-2
- channel-subsystem timing-facility bit (in PMCW) 15-4
- characteristic (of HFP number) 18-1
 - See also* exponent
- characters
 - represented by eight-bit code xxvii
- check bits 3-3, 11-2
- check stop 4-3, 11-9
 - as signal-processor status 4-65
 - during manual operation 12-1
 - effect on CPU timer 4-50
 - entering of 11-12
 - indicator 12-2
 - malfunction alert for 6-12
 - system 11-10
- checking block 11-2
- checking-block code
 - See* CBC
- checkpoint 11-2
- checkpoint synchronization 11-3
 - action 11-4
 - operations 11-3
- CHECKSUM instruction 7-43
- CHH (COMPARE HIGH) instruction 7-129
- CHHR (COMPARE HIGH) instruction 7-129
- CHHSI (COMPARE HALFWORD IMMEDIATE)
 - instruction 7-128
- CHI (COMPARE HALFWORD IMMEDIATE)
 - instruction 7-128
- CHLR (COMPARE HIGH) instruction 7-129
- CHPID (channel-path identifier) 13-5
 - in PMCW 15-8
 - used in RESET CHANNEL PATH 14-9
- CHRL (COMPARE HALFWORD RELATIVE LONG)
 - binary instruction 7-128
- CHSI (COMPARE HALFWORD IMMEDIATE)
 - instruction 7-128
- CHY (COMPARE HALFWORD) instruction 7-128
- CIB (COMPARE IMMEDIATE AND BRANCH)
 - instruction 7-113
- CIH (COMPARE IMMEDIATE HIGH) instruction
 - 7-129
- CIJ (COMPARE IMMEDIATE AND BRANCH
 - RELATIVE) instruction 7-114
- CIPHER MESSAGE instruction 7-46

CIPHER MESSAGE WITH CFB instruction 7-71
 CIPHER MESSAGE WITH CHAINING instruction 7-46
 CIPHER MESSAGE WITH COUNTER instruction 7-86
 CIPHER MESSAGE WITH OFB instruction 7-98
 CIT (COMPARE IMMEDIATE AND TRAP) instruction 7-127
 CKSM (CHECKSUM) instruction 7-43
 CL (COMPARE LOGICAL) instruction 7-130
 class
 of BFP data 19-4
 testing of 19-39
 of DFP data
 testing of 20-51
 CLC (COMPARE LOGICAL) instruction 7-130
 example A-15
 CLCL (COMPARE LOGICAL LONG) instruction 7-135
 example A-16
 CLCLE (COMPARE LOGICAL LONG EXTENDED) instruction 7-138
 CLCLU (COMPARE LOGICAL LONG UNICODE) instruction 7-140
 clear function 15-14, 15-22
 bit in SCSW for 16-14
 completion of 15-15
 initiated by CLEAR SUBCHANNEL 14-5
 path management for 15-14, 15-22
 pending 16-16
 signaling for 15-15
 subchannel modification by 15-15
 clear reset 4-54
 clear signal 17-11
 issued as part of clear function 15-15
 CLEAR SUBCHANNEL instruction 14-5
 See also clear function
 effect on device status of 15-16
 function initiated by 15-14, 15-22
 use of after RESET CHANNEL PATH 14-10
 clearing operation
 by clear-reset function 4-54
 by load-clear key 12-3
 by load-clear-list-directed key 12-3
 by system-reset-clear key 12-5
 by TEST BLOCK instruction 10-153
 CLFDBR (CONVERT TO FIXED) BFP instruction 19-23
 CLFDTR (CONVERT TO LOGICAL) DFP instruction 20-29
 CLFEBR (CONVERT TO FIXED) BFP instruction 19-23
 CLFHSI (COMPARE LOGICAL IMMEDIATE) instruction 7-130
 CLFI (COMPARE LOGICAL IMMEDIATE) instruction 7-130
 CLFIT (COMPARE LOGICAL IMMEDIATE AND TRAP) instruction 7-133
 CLFXBR (CONVERT TO FIXED) BFP instruction 19-23
 CLG (COMPARE LOGICAL) binary instruction 7-130
 CLGDBR (CONVERT TO FIXED) BFP instruction 19-23
 CLGDTR (CONVERT TO LOGICAL) DFP instruction 20-29
 CLGEBR (CONVERT TO FIXED) BFP instruction 19-23
 CLGF (COMPARE LOGICAL) binary instruction 7-130
 CLGFI (COMPARE LOGICAL IMMEDIATE) instruction 7-130
 CLGFR (COMPARE LOGICAL) binary instruction 7-130
 CLGFRL (COMPARE LOGICAL RELATIVE LONG) binary instruction 7-131
 CLGHRL (COMPARE LOGICAL RELATIVE LONG) binary instruction 7-131
 CLGHSI (COMPARE LOGICAL IMMEDIATE) instruction 7-130
 CLGIB (COMPARE LOGICAL IMMEDIATE AND BRANCH) instruction 7-132
 CLGIJ (COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE) instruction 7-132
 CLGIT (COMPARE LOGICAL IMMEDIATE AND TRAP) instruction 7-133
 CLGR (COMPARE LOGICAL) binary instruction 7-130
 CLGRB (COMPARE LOGICAL AND BRANCH) instruction 7-132
 CLGRJ (COMPARE LOGICAL AND BRANCH RELATIVE) instruction 7-132
 CLGRL (COMPARE LOGICAL RELATIVE LONG) binary instruction 7-131
 CLGRT (COMPARE LOGICAL AND TRAP) instruction 7-133
 CLGXBR (CONVERT TO FIXED) BFP instruction 19-23
 CLGXTR (CONVERT TO LOGICAL) DFP instruction 20-29
 CLHH (COMPARE LOGICAL HIGH) instruction 7-135
 CLHHR (COMPARE LOGICAL HIGH) instruction 7-135
 CLHHSI (COMPARE LOGICAL IMMEDIATE) instruction 7-130
 CLHLR (COMPARE LOGICAL HIGH) instruction 7-135

- CLHRL (COMPARE LOGICAL RELATIVE LONG)
 - binary instruction 7-131
- CLI (COMPARE LOGICAL) instruction 7-130
 - example A-15
- CLIB (COMPARE LOGICAL IMMEDIATE AND BRANCH) instruction 7-132
- CLIH (COMPARE LOGICAL IMMEDIATE HIGH) instruction 7-135
- CLIJ (COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE) instruction 7-132
- CLiy (COMPARE LOGICAL) instruction 7-130
- CLM (COMPARE LOGICAL CHARACTERS UNDER MASK) instruction 7-134
 - example A-16
- CLMH (COMPARE LOGICAL CHARACTERS UNDER MASK) instruction 7-134
- CLMY (COMPARE LOGICAL CHARACTERS UNDER MASK) instruction 7-134
- clock
 - See* TOD clock
- clock comparator 4-48
 - external interruption 6-10
 - save areas for 3-67
 - validity bit for 11-20
- clock unit 4-42, 7-306
- CLR (COMPARE LOGICAL) instruction 7-130
 - example A-15
- CLRB (COMPARE LOGICAL AND BRANCH) instruction 7-132
- CLRJ (COMPARE LOGICAL AND BRANCH RELATIVE) instruction 7-132
- CLRL (COMPARE LOGICAL RELATIVE LONG)
 - binary instruction 7-130
- CLRT (COMPARE LOGICAL AND TRAP) instruction 7-133
- CLST (COMPARE LOGICAL STRING) instruction 7-143
 - examples A-18
- CLY (COMPARE LOGICAL) instruction 7-130
- CMPSC (COMPRESSION CALL) instruction 7-147
- CMPSC-enhancement facility 1-7
 - facility indication CMPSC enhancement facility facility indication] 4-68
- code
 - ASCII
 - handled by architecture xxvii
 - checking-block
 - See* CBC
 - command (in CCW)
 - See* command code in CCW
 - condition
 - See* condition code
 - data-exception (DXC) 6-14, 20-14
 - decimal digit and sign 8-2
 - deferred condition (I/O) 16-9
 - EBCDIC
 - handled by architecture xxvii
 - table for 1-2
 - eight-bit
 - handled by architecture xxvii
 - error-recovery (I/O) 17-27
 - external-damage 11-21
 - validity bit for 11-20
 - I/O-interruption subclass 15-2
 - instruction-length
 - See* ILC
 - interruption
 - See* interruption code
 - linkage-stack-entry type 5-75
 - monitor
 - See* monitor code
 - operation 5-2
 - PER
 - See* PER code
 - reporting-source (I/O) 17-26
 - storage-access (in subchannel logout) 16-50
 - version 10-127
- codeword (for sorting operations) 7-115
 - example A-54
- cohort 20-2
- combination field
 - in DFP data format 20-4
- command chaining of CCWs 15-58
 - effect of status modifier on 15-58
 - flag in CCW for 15-32
 - overview of 13-11
- command chaining of TCWs
 - overview of 13-11
- command code in CCW 15-52
 - See also* commands
 - See also* common I/O-device commands
 - invalid 16-25
- command codes
 - See* command code in CCW
- command mode 13-9
- command retry 15-72
 - effect on PCI of 15-60
- commands (I/O) 15-52
 - See also* common I/O-device commands
 - transfer in channel 15-72
- common I/O-device commands
 - publication referenced xxvii
- common-segment bit 3-43
- common-rounding-point view 9-13
- COMPARE AND BRANCH instructions 7-113
- COMPARE AND BRANCH RELATIVE instructions 7-113

COMPARE AND FORM CODEWORD instruction 7-115
 example A-53
 COMPARE AND SIGNAL BFP instructions 19-17
 COMPARE AND SIGNAL DFP instructions 20-22
 COMPARE AND SWAP AND PURGE instruction 10-17
 compare-and-swap-and-store facility 1-7
 facility indication 4-67
 compare-and-swap-and-store facility 2 1-7
 facility indication 4-67
 COMPARE AND SWAP AND STORE instruction 7-124
 COMPARE AND SWAP instruction 7-122
 examples A-46
 compare-and-trap-instruction data exception 6-22
 COMPARE BFP instructions 19-16
 COMPARE BIASED EXPONENT DFP instructions 20-23
 COMPARE binary instructions 7-112
 COMPARE DECIMAL instruction 8-6
 example A-36
 COMPARE DFP instructions 20-21
 COMPARE DOUBLE AND SWAP instruction 7-122
 examples A-46
 COMPARE HALFWORD IMMEDIATE instructions 7-128
 COMPARE HALFWORD instruction 7-128
 example A-14
 COMPARE HFP instructions 18-10
 examples A-42
 COMPARE HIGH instructions 7-129
 COMPARE IMMEDIATE AND BRANCH instructions 7-113
 COMPARE IMMEDIATE AND BRANCH RELATIVE instructions 7-114
 COMPARE IMMEDIATE AND TRAP instructions 7-127
 COMPARE IMMEDIATE binary instructions 7-112
 COMPARE IMMEDIATE HIGH instruction 7-129
 COMPARE LOGICAL AND BRANCH instructions 7-132
 COMPARE LOGICAL AND BRANCH RELATIVE instructions 7-132
 COMPARE LOGICAL AND TRAP instructions 7-133
 COMPARE LOGICAL CHARACTERS UNDER MASK instruction 7-134
 example A-16
 COMPARE LOGICAL HIGH instruction 7-135
 COMPARE LOGICAL IMMEDIATE AND BRANCH instructions 7-132
 COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE instructions 7-132
 COMPARE LOGICAL IMMEDIATE AND TRAP instructions 7-133
 COMPARE LOGICAL IMMEDIATE HIGH instruction 7-135
 COMPARE LOGICAL IMMEDIATE instructions 7-130
 COMPARE LOGICAL instructions 7-130
 examples A-15
 COMPARE LOGICAL LONG EXTENDED instruction 7-138
 COMPARE LOGICAL LONG instruction 7-135
 example A-16
 COMPARE LOGICAL LONG UNICODE instruction 7-140
 COMPARE LOGICAL STRING instruction 7-143
 examples A-18
 COMPARE RELATIVE LONG instructions 7-112
 COMPARE UNTIL SUBSTRING EQUAL instruction 7-144
 comparison
 address
 See address comparison
 decimal 8-6
 example A-36
 hexadecimal-floating-point
 examples A-42
 logical 7-8
 examples A-15
 of IEEE data 9-24
 signed-binary 7-8
 TOD-clock 4-48
 compatibility 1-26
 among systems implementing different architectures 1-27
 among systems implementing same architecture 1-26
 control-program 1-27
 problem-state 1-27
 completion of I/O functions
 by channel-path-reset function 15-76
 by clear function 15-15
 by halt function 15-17
 during initiation 15-73
 completion of instruction execution 5-19
 completion of unit of operation 5-21
 compressed-data bit number (CBN) 7-149
 example of 7-154
 compressed-data symbol size (CDSS) 7-150
 COMPRESSION CALL instruction 7-147
 restriction related to
 on leftmost bits in symbol-translation-table entry 7-152
 on length of chain of compression-dictionary entries 7-153

- on length of chain of expansion-dictionary entries 7-153
- on length of character symbol during compression 7-153
- on length of character symbol during expansion 7-153
- on number of children 7-153, 7-155
- on offset to symbol-translation table 7-150
- on symbol translation with format-1 sibling descriptors 7-150
- compression dictionary
 - size of with symbol translation 7-150
- compression facility
 - publication referenced xxvii
- COMPUTE INTERMEDIATE MESSAGE DIGEST instruction 7-159
- COMPUTE LAST MESSAGE DIGEST instruction 7-159
- COMPUTE MESSAGE AUTHENTICATION CODE instruction 7-171
- conceptual sequence 5-85
 - as related to storage-operand accesses 5-98
- conclusion of I/O operations 13-10, 16-1
 - during initiation 15-73
- conclusion of instruction execution 5-19
- concurrency of access for storage references 5-97
- concurrent sense
 - in ECW 16-56
 - indicator for (in ERW) 16-52
- concurrent-sense count (in ERW) 16-52
- concurrent-sense facility 17-24
- condition code 4-6
 - deferred 16-9
 - for IEEE instructions 9-24
 - in the PSW 4-6
 - inserted by INSERT PROGRAM MASK 7-213
 - preserved by BRANCH AND LINK 5-12, 7-34
 - set by SET PROGRAM MASK 7-296
 - summary C-1
 - tested by BRANCH ON CONDITION instruction 7-36
 - used for decision making 5-11
 - validity bit for 11-19
- conditional emergency signal (external interruption)
 - signal-processor order 4-61
- conditional-SSKE facility 1-7
 - and invalid CBC in storage key 11-8
- conditional-swapping instructions
 - See COMPARE AND SWAP instruction, COMPARE DOUBLE AND SWAP instruction
- conditional-SSKE facility
 - facility indication 4-67
- conditions for interruption
 - See interruption conditions
- configuration 2-1
 - of storage 3-4
- configuration-alert facility (I/O) 17-23
- configuration-topology facility 1-7
 - facility indication 4-67
- connective
 - See logical connective
- consistency (storage operand) 5-95
 - examples A-49, A-51
- console device 12-1
- console integration 1-24
- control 4-1
 - instructions 10-1
 - manual
 - See manual operation
- control-program compatibility 1-27
- control register 2-5, 4-7
 - save areas for 3-68
 - validity bit 11-20
- control-register assignment 4-8
 - (CRx.y indicates control register x, bit position y)
- CR0.33:
 - SSM-suppression-control bit 6-33, 10-124
- CR0.34:
 - TOD-clock-sync-control bit 4-38, 4-43
- CR0.35:
 - low-address-protection-control bit 3-13
- CR0.36:
 - extraction-authority-control bit 5-25
- CR0.37:
 - secondary-space-control bit 3-37, 5-25
- CR0.38:
 - fetch-protection-override-control bit 3-12
- CR0.39:
 - storage-protection-override-control bit 3-11
- CR0.44:
 - ASN-and-LX-reuse-control bit 3-22
- CR0.45:
 - AFP-register-control bit 9-11
- CR0.48:
 - malfunction-alert subclass-mask bit 6-12
- CR0.49:
 - emergency-signal subclass-mask bit 6-11
- CR0.50:
 - external-call subclass-mask bit 6-11
- CR0.52:
 - clock-comparator subclass-mask bit 6-11
- CR0.53:
 - CPU-timer subclass-mask bit 6-11
- CR0.54:
 - service-signal subclass-mask bit 6-12
- CR0.57:
 - interrupt-key subclass-mask bit 6-12
- CR0.59:

timing-alert subclass-mask bit 6-12
 CR0.61:
 crypto-control bit 6-21
 CR1:
 primary address-space-control element (PASCE) 3-38
 CR1.0-51:
 primary real-space token origin (PRSTKO) 3-39
 primary region-table origin (PRT0) 3-38
 primary segment-table origin (PSTO) 3-38
 CR1.54:
 primary subspace-group-control bit 3-38
 CR1.55:
 primary private-space-control bit 3-38
 CR1.56:
 primary storage-alteration-event-control bit 3-38
 CR1.57:
 primary space-switch-event-control bit 3-38, 6-32
 CR1.58:
 primary real-space-control bit 3-39
 CR1.60-61:
 primary designation-type-control bits 3-39
 CR1.62-63:
 primary region-table length (PRTL) 3-39
 primary segment-table length (PSTL) 3-39
 CR2.33-57:
 dispatchable-unit-control-table origin (DUCTO) 5-50
 CR3.0-31:
 secondary ASTEIN (SASTEIN) 3-22
 CR3.32-47:
 PSW-key mask (PKM) 5-25
 CR3.48-63:
 secondary ASN (SASN) 3-20
 CR4.0-31:
 primary ASTEIN (PASTEIN) 3-22
 CR4.32-47:
 authorization index (AX) 3-31, 5-26
 CR4.48-63:
 primary ASN (PASN) 3-20
 CR5.33-57:
 primary-AST-entry origin (PASTE0) 5-32, 5-50
 CR6.32-39:
 I/O-interruption subclass mask 6-13
 CR7:
 secondary address-space-control element (SASCE) 3-39
 CR7.0-51:
 secondary real-space token origin (SRSTKO) 3-40
 secondary region-table origin (SRT0) 3-40
 secondary segment-table origin (SSTO) 3-40
 CR7.54:
 secondary subspace-group-control bit 3-40
 CR7.55:
 secondary private-space-control bit 3-40
 CR7.56:
 secondary storage-alteration-event-control bit 3-40
 CR7.58:
 secondary real-space-control bit 3-40
 CR7.60-61:
 secondary designation-type-control bits 3-40
 CR7.62-63:
 secondary region-table length (SRTL) 3-40
 secondary segment-table length (SSTL) 3-40
 CR8.32-47:
 extended authorization index (EAX) 5-51
 CR8.48-63:
 monitor-mask bits 6-26
 CR9.32:
 PER successful-branching-event-mask bit 4-24
 CR9.33:
 PER instruction-fetching-event-mask bit 4-24
 CR9.34:
 PER storage-alteration-event-mask bit 4-24
 CR9.36:
 PER store-using-real-address-event-mask bit 4-24
 CR9.40:
 PER branch-address-control bit 4-24
 CR9.42:
 PER storage-alteration-space-control bit 4-24
 CR10.0-63:
 PER starting address 4-24
 CR11.0-63:
 PER ending address 4-24
 CR12.0:
 branch-trace-control bit 4-12
 CR12.1:
 mode-trace-control bit 4-12
 CR12.2-61:
 trace-entry address 4-12
 CR12.62:
 ASN-trace-control bit 4-12
 CR12.63:
 explicit-trace-control bit 4-12
 CR13:
 home address-space-control element (HASCE) 3-40
 CR13.0-51:
 home real-space token origin (HRSTKO) 3-40
 home region-table origin (HRT0) 3-40

- home segment-table origin (HSTO) 3-40
- CR13.55:
 - home private-space-control bit 3-40
- CR13.56:
 - home storage-alteration-event-control bit 3-40
- CR13.57:
 - home space-switch-event-control bit 3-40, 6-33
- CR13.58:
 - home real-space-control bit 3-40
- CR13.60-61:
 - home designation-type-control bits 3-40
- CR13.62-63:
 - home region-table length (HRTL) 3-40
 - home segment-table length (HSTL) 3-40
- CR14.35:
 - channel-report-pending subclass-mask bit 11-23
- CR14.36:
 - recovery subclass-mask bit 11-23
- CR14.37:
 - degradation subclass-mask bit 11-24
- CR14.38:
 - external-damage subclass-mask bit 11-24
- CR14.39:
 - warning subclass-mask bit 11-24
- CR14.42:
 - TOD-clock-control-override control 4-38
- CR14.44:
 - ASN-translation-control bit 3-27, 5-26
- CR14.45-63:
 - ASN-first-table origin (AFTO) 3-27
- CR15.0-60:
 - linkage-stack-entry address 5-74
- control unit 2-6, 13-4
 - effect of I/O-system reset on 17-13
 - sharing of 13-4
 - type of 15-14
- control-unit-defer time (I/O) 17-8
- control unit defer time interval (in measurement block) 17-6
- control-unit-queuing measurement (I/O) 17-8
- control-unit-queuing-time interval (in measurement block) 17-5
- controlled-ASN bit 3-22
- conversion
 - between HFP and BFP data 9-27
 - binary-to-decimal 7-182
 - example A-19
 - decimal-to-binary 7-181
 - example A-18
 - hexadecimal-floating-point-number
 - basic example A-6
 - examples with instructions A-44
 - of floating-point format 9-23, 19-5
- CONVERT BFP TO HFP floating-point-support instructions 9-27
- CONVERT FROM FIXED BFP instructions 19-18, 19-21, 19-32
- CONVERT FROM FIXED DFP instructions 20-24
- CONVERT FROM FIXED HFP instructions 18-10
- CONVERT FROM LOGICAL DFP instructions 20-25
- CONVERT FROM SIGNED PACKED DFP instructions 20-26
- CONVERT FROM UNSIGNED PACKED DFP instructions 20-27
- CONVERT HFP TO BFP floating-point-support instructions 9-28
- CONVERT TO BINARY instruction
 - example A-18
- CONVERT TO BINARY instructions 7-181
- CONVERT TO DECIMAL instruction
 - example A-19
- CONVERT TO DECIMAL instructions 7-182
- CONVERT TO FIXED BFP instructions 19-21, 19-23
- CONVERT TO FIXED DFP instructions 20-27
- CONVERT TO FIXED HFP instructions 18-11
- CONVERT TO LOGICAL DFP instructions 20-29
- CONVERT TO SIGNED PACKED DFP instructions 20-31
- CONVERT TO UNSIGNED PACKED DFP instructions 20-32
- CONVERT UNICODE TO UTF-8 instruction 7-186
- CONVERT UTF-16 TO UTF-32 instruction 7-183
- CONVERT UTF-16 TO UTF-8 instruction 7-186
- CONVERT UTF-32 TO UTF-16 instruction 7-189
- CONVERT UTF-32 TO UTF-8 instruction 7-192
- CONVERT UTF-8 TO UNICODE instruction 7-195
- CONVERT UTF-8 TO UTF-16 instruction 7-195
- CONVERT UTF-8 TO UTF-32 instruction 7-199
- Coordinated Universal Time (UTC) used in TOD epoch 4-41
- COPY ACCESS instruction 7-203
- COPY SIGN floating-point-support instruction 9-29
- count field
 - in CCW 15-33
 - invalid 16-25
 - in SCSW 16-31, 16-44, 16-45
- counter updating (example) A-47
- counting operations 7-37
- coupling facility 1-24
- CP (COMPARE DECIMAL) instruction 8-6
 - example A-36
- CPA
 - See channel-program address
- CPSDR (COPY SIGN) floating-point-support instruction 9-29

- CPU (central processing unit) 2-2
 - address 4-57
 - assigned storage locations for 3-61
 - when stored during external interruptions 6-9
 - checkpoint 11-2
 - effect of power-on reset on 4-55
 - hangup due to string of interruptions 4-3
 - identification (ID) 10-127
 - machine-type number 10-127
 - model number 10-127
 - registers 2-3
 - save areas for 3-67
 - reset 4-53
 - signal-processor order 4-59
 - retry 11-2
 - serialization 5-100
 - signaling 4-58
 - state 4-2
 - check-stop 4-3
 - load 4-3
 - no effect on TOD clock 4-38
 - operating 4-2
 - stopped 4-2
 - version code 10-127
- CPU-measurement-counter facility
 - facility indication 4-68
- CPU-measurement-sampling facility
 - facility indication 4-68
- CPU state 4-2
- CPU timer 4-49
 - external interruption 6-11
 - save areas for 3-67
 - validity bit for 11-20
- CPYA (COPY ACCESS) instruction 7-203
- CR
 - See* control register
- CR (COMPARE) binary instruction 7-112
- CRB (COMPARE AND BRANCH) instruction 7-113
- CRJ (COMPARE AND BRANCH RELATIVE)
 - instruction 7-113
- CRL (COMPARE RELATIVE LONG) binary
 - instruction 7-112
- CRT (COMPARE AND TRAP) instruction 7-127
- CRW (channel-report word)
 - chaining of 17-25, 17-26
 - error-recovery code (ERC) in 17-27
 - overflow in 17-26
 - reporting-source code (RSC) in 17-26
 - reporting-source ID (RSID) in 17-28
 - solicited 17-26
 - storing of 14-17
- crypto-operation exception 6-21
- cryptographic facility 1-23, 2-5
- CS (COMPARE AND SWAP) instruction 7-122
 - examples A-46
- CSCH (CLEAR SUBCHANNEL) instruction 14-5
- CSDTR (CONVERT TO SIGNED PACKED) DFP
 - instruction 20-31
- CSG (COMPARE AND SWAP) instruction 7-122
- CSP (COMPARE AND SWAP AND PURGE)
 - instruction 10-17
- CSPG (COMPARE AND SWAP AND PURGE)
 - instruction 10-17
- CSS priority 15-29
- CSST (COMPARE AND SWAP AND STORE)
 - instruction 7-124
- CSXTR (CONVERT TO SIGNED PACKED) DFP
 - instruction 20-31
- CSY (COMPARE AND SWAP) instruction 7-122
- CTN configuration change (machine-check condition)
 - 11-22
- CU12 (CONVERT UTF-8 TO UTF-16) instruction
 - 7-195
- CU14 (CONVERT UTF-8 TO UTF-32) instruction
 - 7-199
- CU21 (CONVERT UTF-16 TO UTF-8) instruction
 - 7-186
- CU24 (CONVERT UTF-16 TO UTF-32) instruction
 - 7-183
- CU41 (CONVERT UTF-32 TO UTF-8) instruction
 - 7-192
- CU42 (CONVERT UTF-32 TO UTF-16) instruction
 - 7-189
- CUDTR (CONVERT TO UNSIGNED PACKED) DFP
 - instruction 20-32
- current CCW 15-31
 - See also* CCW
- current PSW 4-3, 5-11
 - See also* PSW
 - stored during interruption 6-2
- CUSE (COMPARE UNTIL SUBSTRING EQUAL)
 - instruction 7-144
- CUTFU (CONVERT UTF-8 TO UNICODE) instruction
 - 7-195
- CUUTF (CONVERT UNICODE TO UTF-8) instruction
 - 7-186
- CUXTR (CONVERT TO UNSIGNED PACKED) DFP
 - instruction 20-32
- CVB (CONVERT TO BINARY) instruction 7-181
 - example A-18
- CVBG (CONVERT TO BINARY) instruction 7-181
- CVBY (CONVERT TO BINARY) instruction 7-181
- CVD (CONVERT TO DECIMAL) instruction 7-182
 - example A-19
- CVDG (CONVERT TO DECIMAL) instruction 7-182
- CVDY (CONVERT TO DECIMAL) instruction 7-182
- CXBR (COMPARE) BFP instruction 19-16

- CXFBR (CONVERT FROM FIXED) BFP instruction 19-18, 19-20
- CXFBRA (CONVERT FROM FIXED) BFP instruction 19-18
- CXFR (CONVERT FROM FIXED) HFP instruction 18-10
- CXFTR (CONVERT FROM FIXED) DFP instruction 20-24
- CXGBR (CONVERT FROM FIXED) BFP instruction 19-18, 19-20
- CXGBRA (CONVERT FROM FIXED) BFP instruction 19-18
- CXGR (CONVERT FROM FIXED) HFP instruction 18-10
- CXGTR (CONVERT FROM FIXED) DFP instruction 20-24
- CXGTRA (CONVERT FROM FIXED) DFP instruction 20-24
- CXLFB (CONVERT FROM LOGICAL) BFP instruction 19-20
- CXLFBRA (CONVERT FROM LOGICAL) BFP instruction 19-20
- CXLFR (CONVERT FROM LOGICAL) HFP instruction 18-10
- CXLGTR (CONVERT FROM LOGICAL) DFP instruction 20-24
- CXR (COMPARE) HFP instruction 18-10
- CXSTR (CONVERT FROM SIGNED PACKED) DFP instruction 20-26
- CXTR (COMPARE) DFP instruction 20-21
- CXUTR (CONVERT FROM UNSIGNED PACKED) DFP instruction 20-27
- CY (COMPARE) binary instruction 7-112

D

- D (DIVIDE) binary instruction 7-203
 - example A-19
- D field of instruction 5-9
- damage
 - channel-subsystem 11-16
 - code (external) 11-21
 - validity bit for 11-20
 - external 11-15
 - subclass-mask bit for 11-24
 - instruction-processing 11-14
 - processing 11-17
 - service-processor 11-16
 - system 11-14
 - timing-facility 11-15
- DAT
 - See dynamic address translation
- DAT-enhancement facility
 - facility indication 4-67

- DAT-enhancement facility 1 1-8
- DAT-enhancement facility 2 1-8
- DAT mode (bit in PSW)
 - use in address translation 3-36
- DAT mode (bit in the PSW) 4-5
- DAT protection 3-12
 - bit for in page-table entry 3-44
 - bit for in region-table entry 3-41
 - bit for in segment-table entry 3-43
- DAT-table format error 6-40
- data
 - blocking of (I/O) 15-24
 - format for
 - binary-floating-point instructions 19-2
 - decimal instructions 8-1
 - decimal-floating-point instructions 20-3
 - general instructions 7-5
 - hexadecimal-floating-point instructions 18-2
 - indirect addressing of (I/O) 13-9, 15-61
 - measurement (I/O)
 - See measurement data
 - modified indirect addressing of (I/O) 15-63
 - prefetching of for I/O operation 15-53
 - transport indirect addressing of (I/O) 15-65
- data address (I/O)
 - invalid 16-26
 - invalid specification of 16-26
- data chaining of CCWs 15-56
 - flag in CCW for 15-32
 - overview of 13-11
- data check
 - measurement-block 16-48
- data exception 6-21
 - AFP-register 6-22
 - BFP-instruction 6-22
 - compare-and-trap-instruction 6-22
 - decimal-operand 6-21, 8-4
 - DFP-instruction 6-22
 - IEEE-exception 6-22, 9-17
 - priority of program interruptions for 6-15
 - simulated IEEE exception 6-22
- data-exception code (DXC) 6-14, 20-14
- data signaling and quiet NaNs
 - classes of BFP data 19-4
 - classes of DFP data 20-7
- data streaming (I/O) 13-4
 - effect of CCW count on 15-58
- DCTI (device-connect-time interval)
 - in ESW 16-54
 - in extended-measurement word 16-57
 - in measurement block 17-4
- DCW (device-command word)
 - contents of 15-38
 - current 15-37

- DD (DIVIDE) HFP instruction 18-12
- DDB (DIVIDE) BFP instruction 19-26
- DDBR (DIVIDE) BFP instruction 19-26
- DDR (DIVIDE) HFP instruction 18-11
- DDTR (DIVIDE) DFP instruction 20-32
- DDTRA (DIVIDE) DFP instruction 20-32
- DE (DIVIDE) HFP instruction 18-12
- DEB (DIVIDE) BFP instruction 19-26
- DEBR (DIVIDE) BFP instruction 19-26
- decimal
 - arithmetic 8-3
 - comparison 8-6
 - digit codes 8-2
 - divide exception 6-22
 - instructions 8-1
 - examples A-36
 - number representation 8-1
 - examples A-4
 - operand overlap 8-3
 - overflow
 - exception 6-22
 - mask in PSW 4-6
 - sign codes 8-2
- decimal (DFP)
 - data formats 20-3
- decimal floating point (DFP) 9-4
 - See also* DFP
- decimal-floating-point (DFP)
 - facility 20-1
 - facility indication 4-68
- decimal-floating-point facility 1-8
- decimal-floating-point-rounding facility 1-8
- decimal-operand data exception 6-21, 8-4
- decimal-to-binary conversion 7-181
 - example A-18
- decimal-floating-point (DFP)
 - facility 1-8
 - rounding facility 1-8
- dedicated allegiance 15-13
- default QNaN 9-3, 19-4, 20-7
- deferred condition code 16-9
- degradation (machine-check condition) 11-15
 - subclass-mask bit for 11-24
- degradation, storage (machine-check condition) 11-18
- delay in storing 5-94
- delayed access exception (machine-check condition) 11-16
- deletion of malfunctioning unit 11-4
- denormalization rounding 9-12
- denormalized value 9-12
- DER (DIVIDE) HFP instruction 18-11
 - examples A-42
- derived quantum 20-3
- designation
 - authority-table 3-28
 - entry-table 5-33
 - home real-space (HRSD) 3-40
 - home region-table 3-40
 - home segment-table 3-40
 - linkage-first-table 5-32
 - linkage-table 5-32
 - primary real-space 3-38
 - primary region-table 3-38
 - primary segment-table 3-38
 - real-space 3-38
 - in AST entry 3-28
 - used for access-register translation 5-55
 - used for subspace groups 5-64
 - region-table 3-38
 - in AST entry 3-28
 - used for access-register translation 5-55
 - used for subspace groups 5-64
 - secondary real-space 3-40
 - secondary region-table 3-39
 - secondary segment-table 3-39
 - segment-table 3-38
 - in AST entry 3-28
 - used for access-register translation 5-55
 - used for subspace groups 5-64
- designation (origin and length)
 - access-list 5-52
- designation-type-control bits
 - home 3-40
 - primary 3-39
 - secondary 3-40
- destructive overlap 5-98, 7-234, 7-238, 7-242
 - in the access-register mode 5-87
- device 2-6, 13-5
 - console 12-1
 - effect of I/O-system reset on 17-13
- device-active bit 16-16, 16-36
- device-active-only measurement (I/O) 17-9
- device-active-only-time interval (in measurement block) 17-6
- device address 13-6
- device-busy time (in extended measurement word) 16-58
- device-busy time (in measurement block) 17-6
- device-command word
 - See* DCW
- device-connect-time interval
 - See* DCTI
- device-connect-time measurement 17-9
 - effect of suspension on 15-70
 - enable 15-4
- device-disconnect-time interval (in measurement block) 17-5

- device identifier 13-6
- device number 13-6
 - assignment of 13-6
 - in PMCW 15-5
- device-number valid (bit in PMCW) 15-4
- device status 16-24
 - field-validity flag for (in subchannel logout) 16-29, 16-49
 - with inappropriate bit combination 16-50
- device status check 16-50
- DFP
 - canonical data 20-6
 - canonical declets 20-6
 - classes of DFP data 20-6
 - data signaling and quiet NaNs 20-7
 - infinities 20-7
 - normal numbers 20-7
 - subnormal numbers 20-6
 - zero 20-6
 - cohort 20-2
 - data class
 - testing of 20-51
 - data formats 20-3
 - combination field 20-4
 - encoded trailing significand field 20-5
 - values of finite numbers 20-5
 - derived quantum 20-3
 - format conversion 20-7
 - formatting instructions 20-7
 - signed-packed-decimal format 20-26, 20-31
 - unsigned-packed-decimal format 20-27, 20-32
 - of DFP data
 - canonicalization 20-7
 - preferred quantum 20-2, 20-9
 - quantum 20-2
 - scaled preferred quantum 20-3
 - significand 20-5
 - significant digits 20-5
 - special quantum-handling operations 20-3
- DFP (decimal floating point) 9-4
- DFP-instruction data exception 6-22
- DIAGNOSE instruction 10-19
- dictionary
 - adaptive 7-155
- DIDBR (DIVIDE TO INTEGER) BFP instruction 19-27
- DIEBR (DIVIDE TO INTEGER) BFP instruction 19-27
- digit codes (decimal) 8-2
- digit selector (in EDIT) 8-8
- direct-access storage 3-2
- disabling for interruptions 6-5
- disallowed interruptions 6-5
- dispatchable unit (DU) 5-45
 - access-list designation (DUALD) 5-52
 - control table (DUCT) 5-52
 - origin (DUCTO) 5-50
 - when subspace-group facility installed 5-62
 - when trap facility installed 10-159
- displacement (in relative addressing) 5-9
- display (manual controls) 12-2
- distinct-operands facility 1-8
 - facility indication 4-68
- DIVIDE BFP instructions 19-25
- DIVIDE binary instructions 7-203
 - example A-19
- DIVIDE DECIMAL instruction 8-7
 - example A-37
- DIVIDE DFP instructions 20-32
- divide exception
 - decimal 6-22
 - fixed-point 6-23
 - HFP 6-24
- DIVIDE HFP instructions 18-11
 - examples A-42
- DIVIDE LOGICAL instructions 7-204
- DIVIDE SINGLE binary instructions 7-204
- DIVIDE TO INTEGER BFP instructions 19-27
 - remainder result of 19-5
- divisible instruction execution 5-86
- DL (DIVIDE LOGICAL) instruction 7-204
- DLG (DIVIDE LOGICAL) binary instruction 7-204
- DLGR (DIVIDE LOGICAL) binary instruction 7-204
- DLR (DIVIDE LOGICAL) instruction 7-204
- doubleword 3-3
- doubleword-concurrent storage references 5-97
- DP (DIVIDE DECIMAL) instruction 8-7
 - example A-37
- DR (DIVIDE) binary instruction 7-203
- DSG (DIVIDE SINGLE) binary instruction 7-205
- DSGF (DIVIDE SINGLE) binary instruction 7-205
- DSGFR (DIVIDE SINGLE) binary instruction 7-204
- DSGR (DIVIDE SINGLE) binary instruction 7-204
- DU (dispatchable unit) 5-45
- DUALD (dispatchable-unit access-list designation) 5-52
- DUCT (dispatchable-unit control table) 5-52, 5-62
- DUCTO (dispatchable-unit-control-table origin) 5-50
- dump (standalone) 12-5
- DXBR (DIVIDE) BFP instruction 19-26
- DXC (data-exception code) 6-14, 19-7, 20-14
 - summary figures 6-15
- DXR (DIVIDE) HFP instruction 18-12
- DXTRA (DIVIDE) DFP instruction 20-32
- dynamic address translation (DAT) 3-34
 - by LOAD REAL ADDRESS instruction 10-49
 - by STORE REAL ADDRESS instruction 10-129

- control of 3-36
- explicit and implicit 3-45
- mode bit in PSW
 - use in address translation 3-36
- mode bit in the PSW 4-5
- sequence of table fetches 5-90
- dynamic-reconnection feature 13-3

E

- E (exa) xxvi
- E instruction format 5-5
- EAR (EXTRACT ACCESS) instruction 7-208
- early exception recognition 6-8
- EAX
 - See extended authorization index
- EBCDIC (Extended Binary-Coded-Decimal Interchange Code)
 - architecture designed for xxvii
 - character code
 - table for 1-2
- ECAG (EXTRACT CACHE ATTRIBUTE) instruction 7-208
- ECC (error checking and correction) 11-2
- ECTG (EXTRACT CPU TIME) instruction 7-210
- ECW (extended-control word) 16-56
 - indication in SCSW 16-12
- ED (EDIT) instruction 8-7
 - examples A-37
- EDIT AND MARK instruction 8-10
 - example A-38
- EDIT instruction 8-7
 - examples A-37
- editing instructions 8-4
 - See also ED instruction, EDMK instruction
- EDMK (EDIT AND MARK) instruction 8-10
 - example A-38
- EEDTR (EXTRACT BIASED EXPONENT) DFP instruction 20-34
- EEXTR (EXTRACT BIASED EXPONENT) DFP instruction 20-34
- effective access-list designation 5-52
- effective address 3-5
 - controlled by addressing mode 5-8
 - generation 5-8
 - used for storage interlocks 5-86
- effective address-space-control element 3-45
- EFPC (EXTRACT FPC) floating-point-support instruction 9-30
- EKM (entry key mask) 5-35
- emergency signal (external interruption) 6-11
 - signal-processor order 4-58
- EMIF (ESCON-multiple-image facility) 1-24
- EMW (extended-measurement word) 16-56
 - in IRB 16-56
- enabled (bit for TRAP) 10-159
- enabled (bit in PMCW) 15-3
- enabling for interruptions 6-5
 - subchannel 16-6
- enabling of subchannel 15-3, 16-6
- encoded trailing significand field
 - in DFP data format 20-5
- endian 7-231, 7-311
- ending of instruction execution 5-19
- enhanced-DAT facility 1-8
 - and change-bit override 3-9
 - and DAT protection 3-13
 - and format-control in STE 3-9, 3-15
 - facility indication 4-67
- enhanced-monitor facility 1-9
 - facility indication 4-67
- enhanced-monitor masks 4-9
- enhanced-DAT-enablement control
 - and enhanced-DAT applicability 3-37
 - and the P bit in RTEs 3-41
 - definition 3-37
 - in control register 0 4-8
- Enterprise Systems Connection Architecture (ESCON) I/O interface
 - publication referenced xxvii
- entry
 - extended authorization index 5-72
 - key 5-72
- entry (for tracing) 4-12
- entry descriptor 5-75
- entry index (EX) 5-30
- entry key mask (EKM) 5-35
- entry table (ET)
 - designation 5-33
 - length (ETL) 5-33
 - origin (ETO) 5-33
- entry-table entry (ETE) 5-34, 5-71
- entry-type code 5-75
- EPAIR (EXTRACT PRIMARY ASN AND INSTANCE) instruction 10-20
- EPAR (EXTRACT PRIMARY ASN) instruction 10-20
- epoch (for TOD clock) 4-41
- EPSW (EXTRACT PSW) instruction 7-210
- equipment check
 - in signal-processor status 4-64
- ERC (error-recovery code) 17-27
 - See also CRW
- EREG (EXTRACT STACKED REGISTERS) instruction 10-21
- EREGG (EXTRACT STACKED REGISTERS) instruction 10-21
- error
 - checking and correction 11-2

- from DIAGNOSE instruction 10-20
- I/O-error alert 16-50
- indirect storage 11-19
- intermittent 11-5
- PSW-format 6-8
- secondary (I/O) 16-50
- solid 11-5
- state of TOD clock 4-39
- storage 11-18
- storage-key 11-18
- error-recovery code (ERC) 17-27
 - See also* CRW
- ERW (extended-report word) 16-47, 16-51
 - as result of channel-control check 16-29, 16-43
 - as result of channel-data check 16-28, 16-43
- ESA/370 architecture 1-25
- ESA/390
 - compatibility with z/Architecture 1-27
- ESA/390 architecture
 - architectural-mode controls 12-2
- ESAIR (EXTRACT SECONDARY ASN AND INSTANCE) instruction 10-21
- ESAR (EXTRACT SECONDARY ASN) instruction 10-21
- ESCON (Enterprise Systems Connection Architecture) 13-3
- ESCON (Enterprise Systems Connection Architecture) I/O interface
 - publication referenced xxvii
- ESCON channel-to-channel adapter
 - publication referenced xxvii
- ESCON-multiple-image facility (EMIF) 1-24
- ESDTR (EXTRACT SIGNIFICANCE) DFP instruction 20-35
- ESEA (EXTRACT AND SET EXTENDED AUTHORITY) instruction 10-20
- ESTA (EXTRACT STACKED STATE) instruction 10-23
- ESW (extended-status word) 16-47
 - See also* extended status
- ESW format bit (in SCSW) 16-9
- ESXTR (EXTRACT SIGNIFICANCE) DFP instruction 20-35
- ET
 - See* entry table
- ETE
 - See* entry-table entry
- ETF2-enhancement facility 1-9
 - facility indication 4-67
- ETF3-enhancement facility 1-9
 - facility indication 4-67
- ETL (entry-table length) 5-33
- ETO (entry-table origin) 5-33
- ETR (external time reference) 2-5
- ETR (external time reference) facility 1-23
- ETR sync check (machine-check condition) 11-21
- ETR-timing-lert condition
 - See* timing alert
- event 6-14
 - monitor 7-231
 - PER 4-23
 - space-switch 6-32
- EX (entry index) 5-30
 - translation exception 6-23
- EX (EXECUTE)
 - See* EXECUTE instruction
- EX-translation exception 6-23
- exception access identification 3-62
- exceptions 6-14
 - access (collective program-interruption name) 6-40, 6-46
 - addressing 6-17
 - AFX-translation 6-18
 - ALE-sequence 6-19
 - ALen-translation 6-18
 - ALET-specification 6-19
 - ASCE-type 6-19
 - ASN-translation (collective program-interruption name) 6-49
 - associated with
 - ART 5-60
 - stacking process 5-82
 - unstacking process 5-85
 - ASTE-instance 6-20
 - ASTE-sequence 6-20
 - ASTE-validity 6-20
 - ASX-translation 6-21
 - crypto-operation 6-21
 - data 6-21
 - decimal-divide 6-22
 - decimal-overflow 6-22
 - delayed access (machine-check condition) 11-16
 - during translation 3-52
 - EX-translation 6-23
 - extended-authority 6-23
 - fixed-point-divide 6-23
 - fixed-point-overflow 6-24
 - HFP-divide 6-24
 - HFP-exponent-overflow 6-24
 - HFP-exponent-underflow 6-24
 - HFP-significance 6-24
 - HFP-square-root 6-24
 - IEEE 9-17
 - LFX-translation 6-25
 - LSTE-sequence 6-25
 - LSX-translation 6-25
 - LX-translation 6-25
 - operand (of I/O instruction) 6-27

- operation 6-27
- page-translation 6-27
- PC-translation-specification 6-28
- primary-authority 6-28
- privileged-operation 6-29
- protection 6-30
- PSW-related 6-8
- recognition of
 - early and late 6-8
- region-first-translation 6-30
- region-second-translation 6-31
- region-third-translation 6-31
- region translation 6-30
- secondary-authority 6-32
- segment-translation 6-32
- special-operation 6-33
- specification 6-35
- stack-empty 6-38
- stack-full 6-38
- stack-operation 6-39
- stack-specification 6-39
- stack-type 6-39
- subspace-replacement (collective program-interruption name) 6-49
- trace (collective program-interruption name) 6-49
- trace-table 6-39
- translation-specification 6-40
- EXCLUSIVE OR IMMEDIATE instructions 7-206
- EXCLUSIVE OR instructions 7-205
 - examples A-20
- execute-extensions facility 1-10
 - facility indication 4-67
- EXECUTE instruction 7-207
 - effect of address comparison on 12-1
 - example A-21
 - exceptions while fetching target of 6-7
 - PER event for target of 4-31
- EXECUTE RELATIVE LONG instruction 7-207
- exigent machine-check conditions 11-10
- expanded storage 2-2
 - block number 2-2
- expansion dictionary
 - location of during compression 7-150
- expansion-operation bit 7-150
- expansion process
 - flowchart of 7-158
- explicit address translation 3-45
- explicit-trace-control bit 4-12
- exponent 18-1
 - See also* characteristic, floating point
 - overflow
 - HFP 18-1
 - signed value 9-2
 - underflow
 - HFP 18-1
 - mask in PSW 4-6
- exponent bias 9-2, 19-2, 20-3
 - summary for BFP 19-3
 - summary for DFP 20-6
- extended addressing mode
 - bit in entry-table entry 5-35
 - bit in trace entry 4-20
- extended-authority exception 6-23
 - as an access exception 6-40
- extended authorization 5-59
- extended authorization index (EAX) 5-51
 - control bit 5-71
 - in entry-table entry 5-72
 - in linkage-stack state entry 5-78
- extended binary-floating-point number 19-2
- extended control (bit in SCSW) 16-12
- extended-control word 16-56
 - See also* ECW
- extended decimal-floating-point number 20-4
- extended hexadecimal-floating-point number 18-2
- extended-I/O-measurement-block facility 1-10
- extended-I/O-measurement-word facility 1-10
- extended-immediate facility 1-10
 - facility indication 4-67
- extended-measurement-word 17-10
- extended-measurement-word enable (I/O)
 - update enable 15-8
- extended measurement word mode enable (I/O) 15-8
- extended-report word
 - See* ERW
- extended-sorting facility 1-24
- extended status
 - See also* ESW
 - flags in subchannel logout for 16-47
 - format-0 16-47
 - format-1 16-53
 - format-2 16-54
 - format-3 16-55
 - secondary-CCW address 16-53
- extended-status word 16-47
 - See also* extended status
- extended-status-word-format bit 16-9
- extended-subchannel-logout descriptor (in ERW) 16-53
 - in ESW 16-53
- extended-subchannel-logout pending 16-52
 - indicator for (in ERW) 16-52
- extended-translation facility 2 1-10
 - facility indication 4-67
- extended-translation facility 3 1-11
 - facility indication 4-67
- external call

- external interruption 6-11
 - pending (signal-processor status) 4-64
 - signal-processor order 4-58

- external damage 11-15

- subclass-mask bit for 11-24

- external-damage code 11-21

- assigned storage locations for 3-65

- validity bit for 11-20

- external interruption 6-9

- clock-comparator 6-10, 4-48

- CPU-timer 4-49, 6-11

- direct conditions 6-9

- emergency-signal 6-11

- external-call 6-11

- interrupt-key 6-11

- malfunction-alert 6-12

- mask in the PSW 4-5

- parameter 6-9

- assigned storage locations for 3-61

- pending conditions 6-9

- priority of conditions 6-10

- service-signal 6-12

- timing-alert 6-12

- external time reference (ETR) 2-5

- external-time-reference (ETR) facility 1-23

- externally initiated functions 4-50

- I/O 17-16

- EXTRACT ACCESS instruction 7-208

- EXTRACT AND SET EXTENDED AUTHORITY

- instruction 10-20

- EXTRACT BIASED EXPONENT DFP instructions

- 20-34

- extract-CPU-time facility 1-11, 7-210

- facility indication 4-67

- EXTRACT CPU TIME instruction 7-210

- EXTRACT FPC floating-point-support instruction

- 9-30

- EXTRACT PRIMARY ASN AND INSTANCE

- instruction 10-20

- EXTRACT PRIMARY ASN instruction 10-20

- EXTRACT PSW instruction 7-210

- EXTRACT SECONDARY ASN AND INSTANCE

- instruction 10-21

- EXTRACT SECONDARY ASN instruction 10-21

- EXTRACT SIGNIFICANCE DFP instructions 20-35

- EXTRACT STACKED REGISTERS instruction 10-21

- EXTRACT STACKED STATE instruction 10-23

- extraction-authority-control bit 5-25

F

- facility indication

- ASN-and-LX-reuse facility 4-67

- CMPSC-enhancement facility 4-68

- compare-and-swap-and-store facility 4-67

- compare-and-swap-and-store facility 2 4-67

- conditional-SSKE facility 4-67

- configuration-topology facility 4-67

- CPU-measurement-counter facility 4-68

- CPU-measurement-sampling facility 4-68

- DAT-enhancement facility 4-67

- decimal-floating-point facility 4-68

- decimal-floating-point high performance 4-68

- distinct-operands facility 4-68

- enhanced-DAT facility 4-67

- enhanced-monitor facility 4-67

- ETF2-enhancement facility 4-67

- ETF3-enhancement facility 4-67

- execute-extensions facility 4-67

- extended-immediate facility 4-67

- extended-translation facility 2 4-67

- extended-translation facility 3 4-67

- extract-CPU-time facility 4-67

- fast-BCR-serialization facility 4-68

- floating-point-extension facility 4-67

- floating-point-support-enhancement facilities 4-68

- general-instructions-extension facility 4-67

- HFP-multiply-and-add/subtract facility 4-67

- HFP-unnormalized-extension facility 4-67

- high-word facility 4-68

- interlocked-access facility 4-68

- IPTE-range facility 4-67

- load/store-on-condition facility 4-68

- long-displacement facility 4-67

- message-security-assist extension 3 4-68

- message-security-assist extension 4 4-68

- message-security assist 4-67

- move-with-optional-specifications facility 4-67

- parsing-enhancement facility 4-67

- PFPO facility 4-68

- population-count facility 4-68

- reset-reference-bits-multiple facility 4-68

- sense-running-status facility 4-67

- set-program-parameters facility 4-68

- store-clock-fast facility 4-67

- store-facility-list-extended facility 4-67

- TOD-clock-steering facility 4-67

- z/architecture active 4-67

- z/architecture installed 4-67

- z/architecture instructions in ESA/390 mode 4-67

- facility indications 4-66

- failing-storage address 11-22

- assigned storage locations for 3-65

- in ESW 16-47, 16-53

- as result of channel-control check 16-29, 16-43

- as result of channel-data check 16-28, 16-43

- validity bit for 11-20
 - validity flag for (in ERW) 16-52
- failing-storage-address format
 - indicator for (in ERW) 16-53
- fast-BCR-serialization facility 1-11
 - facility indication 4-68
- FCX facility 1-11
- fetch-only bit 5-53
- fetch protection 3-10
 - bit in segment-table entry 3-43
 - bit in storage key 3-9
 - override-control bit 3-12
- fetch reference 5-93
 - access exceptions for 6-43
- fetching
 - handling of invalid CBC in storage during 11-7
 - of ART-table and DAT-table entries 5-90
 - of instructions 5-89
 - of PSWs during interruptions 5-99
 - of storage operands 5-93
- Fibre-Channel Extensions 13-6
- fibre-channel-extensions facility 1-11
- Fibre-Channel Extensions 13-6
- FICON I/O interface
 - publication referenced xxvii
- FIDBR (LOAD FP INTEGER) BFP instruction 19-31
- FIDBRA (LOAD FP INTEGER) BFP instruction 19-31
- FIDR (LOAD FP INTEGER) HFP instruction 18-14
- FIDTR (LOAD FP INTEGER) DFP instruction 20-37
- FIEBR (LOAD FP INTEGER) BFP instruction 19-31
- FIEBRA (LOAD FP INTEGER) BFP instruction 19-31
- field 3-2
- field separator (in EDIT) 8-8
- field-validity flags (in subchannel logout) 16-49
 - relation to channel-control check of 16-29
- FIER (LOAD FP INTEGER) HFP instruction 18-14
- FIFO (first in first out) queuing
 - example for lock and unlock A-49
- fill byte (in EDIT) 8-8
- FIND LEFTMOST ONE instruction 7-211
 - example A-22
- FIXBR (LOAD FP INTEGER) BFP instruction 19-31
- FIXBRA (LOAD FP INTEGER) BFP instruction 19-31
- fixed-length field 3-3
- fixed logout
 - assigned storage locations for 3-67
 - machine-check 11-24
- fixed point
 - See also* binary
 - divide exception 6-23
 - overflow exception 6-24
 - mask in PSW 4-6
 - See also* binary
- FIXR (LOAD FP INTEGER) HFP instruction 18-14

- FIXTR (LOAD FP INTEGER) DFP instruction 20-37
- flags
 - for IEEE exceptions 9-10
- floating interruption conditions 6-6, 11-22
 - clearing of 4-54
- floating point
 - See also* exponent
 - binary (BFP) 9-4
 - binary data format 19-2
 - conversion
 - between formats 9-23, 19-5
 - conversion from binary integer 18-10, 19-18, 20-24
 - conversion from unsigned binary integer 19-20, 20-25
 - conversion to binary integer 18-11, 19-21, 20-27
 - conversion to unsigned binary integer 19-23, 20-29
- data
 - lengthening format of 18-15, 19-32, 20-40
 - shortening format of 18-16, 19-34, 20-41
- data class 19-4
- decimal (DFP) 9-4
- hexadecimal (HFP) 9-3
- hexadecimal data formats 18-2
- instructions 9-2
- numbers 19-2
- registers 2-3, 9-8
 - clearing of 9-34
 - save areas for 3-67
 - validity bit for 11-20
- shifting
 - See* normalization
- floating-point-control (FPC) register 9-9
 - save area for 3-67
 - validity bit for 11-20
- floating-point extension facility 1-12
- floating-point-extension facility
 - facility indication 4-67
- floating-point-support-enhancement facilities
 - facility indication 4-68
- floating-point-support-sign-handling facility 1-12
- FLOGR (FIND LEFTMOST ONE) instruction 7-211
 - example A-22
- format
 - address 3-3
 - binary-floating-point data 19-2
 - CCW
 - See* CCW format control
 - decimal data 8-1
 - error
 - in DAT-table entry 6-40
 - in PSW 6-8
 - general data 7-5

- information 3-2
- instruction 5-3
- PSW 4-5
- format-0 and format-1 CCWs 15-31
- format-2-IDAW control 15-27
- format control 15-27
- format-1-sibling-descriptor bit 7-150
- formats
 - hexadecimal-floating-point data 18-2
- forward section header address 5-77
- forward-section validity bit 5-76
- FPC
 - See* floating-point-control (FPC) register
- FPC (floating-point-control) register 9-9
- FPR-GR-transfer facility 1-13
- fraction 18-1
- fraction view of floating-point numbers 9-2
- free-pool manipulation
 - programming example A-50
- freeze 7-155
- fullword
 - See* word
- function control (I/O) 16-13
- function-pending time 17-2
 - in extended-measurement word 16-57
- functionally-constrained rounding 9-12
- functionally-rounded value 9-12
- function-pending time
 - in measurement block 17-5

G

- G (giga) xxvi
- general instructions 7-5
 - examples A-7
- general-instructions-extension facility 1-13
 - facility indication 4-67
- general register
 - save areas 3-67
- general registers 2-3
 - validity bit for 11-20
- glue module 5-18
- GMT (Greenwich Mean Time) obsolete term for UTC 4-41
- Greenwich Mean Time (GMT) obsolete term for Coordinated Universal Time 4-41
- guard digit 18-4

H

- halfword 3-3
- halfword-concurrent storage references 5-97
- halt function 15-16
 - bit in SCSW for 16-13
 - completion of 15-17

- initiated by HALT SUBCHANNEL 14-6
- path management for 15-16
- pending 16-16
- signaling for 15-17
- halt signal 17-11
 - issued as part of halt function 15-17
- HALT SUBCHANNEL instruction 14-6
 - See also* halt function
 - effect on SCSW count field 15-19
 - function initiated by 15-16
 - use of after RESET CHANNEL PATH 14-10
- HALVE HFP instructions 18-13
 - example A-43
- HASCE (home address-space-control element) 3-40
- HDR (HALVE) HFP instruction 18-13
 - example A-43
- header entry 5-76
- HER (HALVE) HFP instruction 18-13
- hex
 - See* hexadecimal
- hexadecimal (hex) representation 5-6
- hexadecimal floating point
 - conversion
 - examples with instructions A-44
 - instructions
 - examples A-41
- hexadecimal floating point (HFP) 9-3
 - conversion
 - basic example A-6
- hexadecimal-floating-point number
 - examples A-5
- HFP (hexadecimal floating point) 9-3
- HFP data 18-2
 - conversion of 9-27
- HFP exponent
 - overflow
 - exception 6-24
 - underflow
 - exception 6-24
- HFP-multiply-and-add/subtract facility 1-13
 - facility indication 4-67
- HFP significance
 - exception 6-24
- HFP square root
 - exception 6-24
- HFP-unnormalized-extension facility 1-13
 - facility indication 4-67
- HFP-unnormalized-extensions facility 1-13
- high-speed data transfer (I/O) 13-4
- high-word facility 1-13
 - facility indication 4-68
- home address space 3-19, 5-41
 - facilities 5-41
- home designation-type-control bits 3-40

- home private-space-control bit 3-40
- home real-space-control bit 3-40
- home real-space token origin (HRSTKO) 3-40
- home region table
 - designation (HRTD) 3-40
 - length (HRTL) 3-40
 - origin (HRTO) 3-40
- home segment table
 - designation (HSTD) 3-40
 - length (HSTL) 3-40
 - origin (HSTO) 3-40
- home-space mode 3-36
- home space-switch-event-control bit 3-40
- home storage-alteration-event-control bit 3-40
- home virtual address 3-5
 - effective address-space-control element for 3-45
- HRSD (home real-space designation) 3-40
- HRTD (home region-table designation) 3-40
- HRTL (home region-table length) 3-40
- HRTO (home region-table origin) 3-40
- HSCH (HALT SUBCHANNEL) instruction 14-6
- HSTD (home segment-table designation) 3-40
- HSTL (home segment-table length) 3-40
- HSTO (home segment-table origin) 3-40

I

- I field of instruction 5-6
- I instruction format 5-5
- I/O (input/output) 2-5
 - basic functions of 15-1
 - blocking of data for 15-24
 - effect on CPU timer 4-49
 - sense data
 - See* sense data
 - support functions of 17-1
- I/O addressing 13-5
- I/O-command words
 - CCW 13-6
 - DCW 13-6
- I/O commands
 - See also* commands
 - publication referenced xxvii
- I/O device
 - See* device
- I/O-error alert (in subchannel logout) 16-50
- I/O instructions 14-1, 14-2
 - deferred condition code for 16-9
 - operand access by 14-2
 - role of in I/O operations 13-8
- I/O interface
 - ESCON publication referenced xxvii
 - OEMI publication referenced xxvii
- I/O interruption 6-13, 16-1

- See also* interruption
 - action for 16-6
 - mask in the PSW 4-5
 - masking of 13-12
 - priority of 16-5
 - program-controlled interrupton
 - See* PCI)

- I/O-interruption code 6-13
 - interruption-identification word in 14-20
 - stored by TPI 14-20
- I/O-interruption condition 13-12, 16-2
 - alert 16-5
 - intermediate 16-4
 - primary 13-10, 16-4
 - secondary 13-10, 16-5
 - solicited 16-3
 - unsolicited 16-3
- I/O-interruption-identification word 14-20
 - assigned storage locations for 3-65
- I/O-interruption parameter
 - assigned storage locations for 3-65
 - in I/O-interruption code 14-20
 - in ORB 15-25
 - in PMCW 15-2
 - used for IPL 17-17
- I/O-interruption request
 - clearing of 13-12
 - from subchannels 16-6
- I/O-interruption subclass 13-12
- I/O-interruption subclass code
 - See* ISC
- I/O-interruption subclass mask 6-13, 16-6
 - relation to priority 16-5
- I/O operations 13-8
 - conclusion of
 - See* conclusion of I/O operations
 - initiated indication for 16-12
 - termination of
 - See* conclusion of I/O operations
- I/O-system reset 17-13
 - as part of subsystem reset 4-54
- IAC (INSERT ADDRESS SPACE CONTROL)
 - instruction 10-25
- IC (INSERT CHARACTER) instruction 7-211
- IC (instruction counter)
 - See* instruction address
- ICM (INSERT CHARACTERS UNDER MASK)
 - instruction 7-212
 - examples A-23
- ICMH (INSERT CHARACTERS UNDER MASK)
 - instruction 7-212
- ICMY (INSERT CHARACTERS UNDER MASK)
 - instruction 7-212
- ICY (INSERT CHARACTER) instruction 7-211

- ID
 - See* CPU identification, sense ID
- IDA (indirect-data address) 15-61
 - flag in CCW 15-32
- IDAW (indirect-data-address word) 15-61
 - check (in subchannel logout) 16-48
 - invalid address of 16-25
 - invalid address specification in 16-25
 - invalid address specification of 16-26
- idle state for subchannel 16-14
- IDTE (INVALIDATE DAT TABLE ENTRY) instruction 10-28
- IEDTR (INSERT BIASED EXPONENT) DFP instruction 20-35
- IEEE exception conditions
 - summary figure 19-7
- IEEE-exception data exception 6-22, 9-17
- IEEE-exception-simulation facility 1-14
- IEEE rounding 9-13
- IEEE standard 1-22
- IEEE trap action and nontrap action 9-17
- IEEE-exception data exception 19-7
- IEXTR (INSERT BIASED EXPONENT) DFP instruction 20-35
- IFCC (interface-control check) 16-29
- IIHF (INSERT IMMEDIATE) instruction 7-212
- IIHH (INSERT IMMEDIATE) instruction 7-212
- IIHL (INSERT IMMEDIATE) instruction 7-213
- IILF (INSERT IMMEDIATE) instruction 7-213
- IILH (INSERT IMMEDIATE) instruction 7-213
- IILL (INSERT IMMEDIATE) instruction 7-213
- ILC (instruction-length code) 6-6
 - assigned storage locations for 3-61
 - for program interruptions 6-14
- IML (initial machine loading) controls 12-3
- immediate operand 5-6
- immediate operation
 - SLI flag in CCW for 15-54
- implicit address translation 3-45
- in DFP data 20-5
- incorrect length (subchannel status) 16-24
 - for immediate operations 15-54
- incorrect-length-indication mode 15-28
- incorrect-length-indication-suppression facility 17-23
 - effect on immediate operation 15-54
- incorrect-length-suppression mode 15-28
- incorrect state (signal-processor status) 4-64
- index
 - for address generation 5-9
 - instructions for branching on 7-38
 - into access list 5-51
 - into ASN first and second tables 3-26
 - into authority table 5-26
 - into entry and linkage tables 5-30
 - register for 2-3
- indicator
 - check-stop 12-2
 - load 12-3
 - manual 12-4
 - mode 12-2
 - test 12-5
 - wait 12-6
- indirect-data address
 - See* IDA
- indirect-data-address word
 - See* IDAW
- indirect storage error 11-19
- infinities 9-2, 19-4
 - classes of DFP data 20-7
- information format 3-2
- initial-command-response measurement (I/O) 17-9
- initial-command-response time (in measurement block) 17-6
- initial CPU reset 4-54
 - signal-processor order 4-59
- initial-machine-loading (IML) controls 12-3
- initial program loading
 - See* IPL
- initial-status-interruption control 15-27, 16-12
 - relation to Z bit 16-12
 - used for IPL 17-17
- inoperative (signal-processor status) 4-65
- input/output
 - See* I/O
- INSERT ADDRESS SPACE CONTROL instruction 10-25
- INSERT BIASED EXPONENT DFP instructions 20-35
- INSERT CHARACTER instructions 7-211
- INSERT CHARACTERS UNDER MASK instruction 7-212
 - examples A-23
- INSERT IMMEDIATE instructions 7-212
- INSERT PROGRAM MASK instruction 7-213
- INSERT PSW KEY instruction 10-26
- INSERT STORAGE KEY EXTENDED instruction 10-26
- INSERT VIRTUAL STORAGE KEY instruction 10-27
- installation 2-1
- instruction address
 - as a type of address 3-5
 - handling by DAT 3-36
 - in entry-table entry 5-34
 - in the PSW 4-6
 - validity bit for 11-20
- instruction-fetching-nullification-event mask 4-24
- instruction format 5-5
- instruction-length code

- preserved by BRANCH AND LINK 5-12, 7-34
- See ILC
- instruction-processing damage 11-14
 - resulting in processing backup 11-17
 - resulting in processing damage 11-17
- instructions
 - See also instruction lists and page numbers in Appendix B
 - backing up of 11-17
 - classes of 2-2
 - control 10-1
 - damage to 11-14, 11-17
 - decimal 8-1
 - examples A-36
 - divisible execution of 5-86
 - ending of 5-19
 - examples of use A-7
 - execution of 5-11
 - fetching of 5-89
 - access exception for 6-43
 - PER event for 4-31
 - PER-event mask for 4-24
 - floating-point 9-2
 - format of 5-3
 - general 7-5
 - examples A-7
 - hexadecimal-floating-point
 - examples A-41
 - I/O
 - See I/O Instructions 13-1
 - interruptible
 - See interruptible instructions
 - length of 5-6
 - list of B-1
 - modification by execute-type instruction 7-207
 - prefetching of 5-89
 - privileged 4-5
 - for control 10-1
 - See instruction lists and page numbers in Appendix B
 - semiprivileged 4-5, 10-2
 - sequence of execution of 5-2
 - stepping of (rate control) 12-4
 - effect on CPU state 4-2
 - effect on CPU timer 4-50
 - unprivileged 4-5, 7-5
- integer
 - binary 7-5
 - address as 5-9
 - conversion from floating point 18-11, 19-21, 19-23, 20-27, 20-29
 - conversion to floating point 18-10, 19-18, 20-24, 20-25
 - examples A-2
 - decimal 8-3
 - unsigned binary
 - conversion to floating point 19-20
- integer quotient 19-27
- integral boundary 3-3
- interface
 - ESCON I/O
 - publication referenced xxvii
 - parallel-I/O
 - OEMI publication referenced xxvii
 - serial-I/O
 - publication referenced xxvii
- interface-control check (subchannel status) 16-29
- interlocked-access facility 1-14
 - facility indication 4-68
- interlocked-update storage reference 5-93, 5-94
- interlocks for virtual storage references 5-86
- intermediate interruption condition (I/O) 16-4
- intermediate-status bit (I/O) 16-18, 16-36
- intermediate values 9-11
- intermittent errors 11-5
- International Atomic Time (TAI) related to Coordinated Universal Time 4-41
- interpretive execution
 - publication referenced xxvii
- interpretive-execution facility 1-25
- interrogate TCW
 - address of 15-36
- interrupt key 12-3
 - external interruption 6-11
- interruptible instructions 5-20
 - COMPARE AND FORM CODEWORD 7-115
 - COMPARE LOGICAL LONG 7-136
 - COMPARE UNTIL SUBSTRING EQUAL 7-147
 - MOVE LONG 7-235
 - PER event affecting the ending of 4-29
 - stopping of 4-2
 - TEST BLOCK 10-153
 - UPDATE TREE 7-334
- interruption 6-1
 - See also masks
 - action 6-2
 - I/O 16-6
 - machine-check 11-11
 - classes of 6-4
 - effect on instruction sequence 5-19
 - external
 - See external interruption
 - I/O
 - See I/O interruption
 - machine-check
 - See machine-check interruption
 - masking of 6-5
 - pending 6-5

- external 6-9
- machine-check 11-12
- relation to CPU state 4-2
- priority of
 - See priority
- program
 - See program interruption
- program-controlled (I/O)
 - See PCI
- restart 6-49
- string
 - See string of interruptions
- supervisor-call 6-50
- interruption code 6-4
 - external 6-9
 - I/O
 - See I/O-interruption code
 - machine-check (MCIC) 3-65, 11-13
 - program 6-14
 - summary of 6-2
- interruption conditions 6-1
 - clearing of 4-53
 - floating 6-6, 11-22
 - I/O
 - See I/O-interruption condition
- interruption parameter
 - external (assigned storage locations) 3-61
 - I/O
 - See I/O-interruption parameter
- interruption-response block
 - See IRB
- interruption subclass
 - See I/O-interruption subclass
- invalid
 - access-list entry 5-53
 - address 6-17
 - bit in ASN-first-table entry 3-27
 - bit in ASN-second-table entry 3-28
 - bit in linkage-first-table entry 5-33
 - bit in linkage-second-table entry 5-33
 - bit in linkage-table entry 5-33
 - bit in page-table entry 3-44
 - bit in region-table entry 3-41
 - bit in segment-table entry 3-43
 - CBC 11-2
 - in registers 11-8
 - in storage 11-6
 - in storage keys 11-7
 - operation code 6-27
 - order (signal-processor status) 4-65
 - parameter (signal-processor status) 4-64
 - translation address 3-52
 - translation format
 - exception recognition 3-52

- invalid address specification
 - in channel-program address 16-25, 16-40
 - in IDAW 16-26
 - in MIDAW 16-26
 - of data in CCW 16-26
 - of IDAW 16-25
 - of TIC CCW 16-25, 16-40
- invalid CCW field
 - command code 16-25
 - count 16-25
 - data address 16-26
 - MIDA flag 16-27
 - suspend flag 16-27
- invalid format
 - of CCW 16-27
 - of ORB 16-27
- invalid sequence of CCWs 16-27
- INVALIDATE DAT TABLE ENTRY instruction 10-28
- INVALIDATE PAGE TABLE ENTRY instruction 10-32
 - effect of when CPU is stopped 4-2
- inverse move
 - See MOVE INVERSE instruction, move-inverse facility
- IPK (INSERT PSW KEY) instruction 10-26
- IPL (initial program loading) 4-56, 17-16
 - effect on CPU state 4-3
- IPM (INSERT PROGRAM MASK) instruction 7-213
- IPT (INVALIDATE PAGE TABLE ENTRY) instruction 10-32
- IPT-range facility 1-14
 - facility indication 4-67
- IRB (interruption-response block) 16-6
 - See also ECW, ERW, ESW, SCSW
 - storage requirements for 16-13
- ISC (I/O-interruption-subclass code) 15-2
 - enhanced 14-20
- ISKE (INSERT STORAGE KEY EXTENDED) instruction 10-26
- isolated state 16-52
- IVSK (INSERT VIRTUAL STORAGE KEY) instruction 10-27

K

- K (kilo) xxvi
- KDB (COMPARE AND SIGNAL) BFP instruction 19-17
- KDBR (COMPARE AND SIGNAL) BFP instruction 19-17
- KDTR (COMPARE AND SIGNAL) DFP instruction 20-22
- KEB (COMPARE AND SIGNAL) BFP instruction 19-17

KEBR (COMPARE AND SIGNAL) BFP instruction 19-17

key

access

See access key

manual

See manual operation

PSW

See PSW key

storage

See storage key

subchannel

See subchannel key

key check (in subchannel logout) 16-47

key-controlled protection 3-10

exception for 6-30

key mask

authorization 5-34

entry 5-35

PSW (PKM) 5-25

KIMD (COMPUTE INTERMEDIATE MESSAGE DIGEST) instruction 7-159

KLMD (COMPUTE LAST MESSAGE DIGEST) instruction 7-159

KM (CIPHER MESSAGE) instruction 7-46

KMAC (COMPUTE MESSAGE AUTHENTICATION CODE) instruction 7-171

KMC (CIPHER MESSAGE WITH CHAINING) instruction 7-46

KMCTR (CIPHER MESSAGE WITH COUNTER) instruction 7-86

KMF (CIPHER MESSAGE WITH CFB) instruction 7-71

KMO (CIPHER MESSAGE WITH OFB) instruction 7-98

KXBR (COMPARE AND SIGNAL) BFP instruction 19-17

KXTR (COMPARE AND SIGNAL) DFP instruction 20-22

L

L (LOAD) binary instruction 7-213

example A-23

L fields of instruction 5-7

LA (LOAD ADDRESS) instruction 7-215

examples A-23

LAA (LOAD AND ADD) instruction 7-217

LAAG (LOAD AND ADD) instruction 7-217

LAAL (LOAD AND ADD LOGICAL) instruction 7-217

LAALG (LOAD AND ADD LOGICAL) instruction 7-217

LAE (LOAD ADDRESS EXTENDED) instruction 7-215

LAEY (LOAD ADDRESS EXTENDED) instruction 7-215

LAM (LOAD ACCESS MULTIPLE) instruction 7-214

LAMY (LOAD ACCESS MULTIPLE) instruction 7-214

LAN (LOAD AND AND) instruction 7-218

LANG (LOAD AND AND) instruction 7-218

LAO (LOAD AND OR) instruction 7-219

LAOG (LOAD AND OR) instruction 7-219

LARL (LOAD ADDRESS RELATIVE LONG) instruction 7-216

LASP (LOAD ADDRESS SPACE PARAMETERS) instruction 10-34

last-path-used mask

See LPUM

late exception recognition 6-9

LAX (LOAD AND EXCLUSIVE OR) instruction 7-218

LAXG (LOAD AND EXCLUSIVE OR) instruction 7-218

LAY (LOAD ADDRESS) instruction 7-215

LB (LOAD BYTE) instruction 7-220

LBH (LOAD BYTE HIGH) instruction 7-221

LBR (LOAD BYTE) binary instruction 7-220

LCDBR (LOAD COMPLEMENT) BFP instruction 19-30

LCDFR (LOAD COMPLEMENT) floating-point-support instruction 9-30

LCDR (LOAD COMPLEMENT) HFP instruction 18-14

LCEBR (LOAD COMPLEMENT) BFP instruction 19-30

LCER (LOAD COMPLEMENT) HFP instruction 18-14

LCGFR (LOAD COMPLEMENT) binary instruction 7-221

LCGR (LOAD COMPLEMENT) binary instruction 7-221

LCR (LOAD COMPLEMENT) binary instruction 7-221

LCTL (LOAD CONTROL) instruction 10-44

LCTLG (LOAD CONTROL) instruction 10-44

LCXBR (LOAD COMPLEMENT) BFP instruction 19-30

LCXR (LOAD COMPLEMENT) HFP instruction 18-14

LD (LOAD) floating-point-support instruction 9-30

LDE (LOAD LENGTHENED) HFP instruction 18-15

LDEB (LOAD LENGTHENED) BFP instruction 19-33

LDEBR (LOAD LENGTHENED) BFP instruction 19-32

LDER (LOAD LENGTHENED) HFP instruction 18-15

LDETR (LOAD LENGTHENED) DFP instruction 20-40

LDGR (LOAD FPR FROM GR) floating-point-support instruction 9-33

LDR (LOAD) floating-point-support instruction 9-30

LDXBR (LOAD ROUNDED) BFP instruction 19-34

LDXBRA (LOAD ROUNDED) BFP instruction 19-34

LDXR (LOAD ROUNDED) HFP instruction 18-16

LDXTR (LOAD ROUNDED) DFP instruction 20-41

LDY (LOAD) floating-point-support instruction 9-30

LE (LOAD) floating-point-support instruction 9-30

LEDBR (LOAD ROUNDED) BFP instruction 19-34

LEDBRA (LOAD ROUNDED) BFP instruction 19-34

LEDR (LOAD ROUNDED) HFP instruction 18-16

LEDTR (LOAD ROUNDED) DFP instruction 20-41

left-to-right addressing 3-2

left-units view (LUV) of floating-point numbers 9-2

length

- field 3-2
- instruction 5-6
- of BFP data
 - decreasing 19-34
 - increasing 19-32
- of DFP data
 - decreasing 20-41
 - increasing 20-40
- of HFP data
 - decreasing 18-16
 - increasing 18-15
- register-operand 5-6
- second operand same as first 5-6
- variable (storage operand) 5-7

LER (LOAD) floating-point-support instruction 9-30

LEXBR (LOAD ROUNDED) BFP instruction 19-34

LEXBRA (LOAD ROUNDED) BFP instruction 19-34

LEXR (LOAD ROUNDED) HFP instruction 18-16

LEY (LOAD) floating-point-support instruction 9-30

LFAS (LOAD FPC AND SIGNAL) floating-point-support instruction 9-31

LFH (LOAD HIGH) instruction 7-223

LFPC (LOAD FPC) floating-point-support instruction 9-31

LFT (linkage first table) 5-33

LFTD (linkage-first-table designation) 5-32

LFTL (linkage-first-table length) 5-32

LFTO (linkage-first-table origin) 5-32

LFX (linkage first index)

- invalid bit 5-33
- translation exception 6-25

LG (LOAD) binary instruction 7-213

LGB (LOAD BYTE) instruction 7-220

LGBR (LOAD BYTE) binary instruction 7-220

LGDR (LOAD GR FROM FPR) floating-point-support instruction 9-33

LGF (LOAD) binary instruction 7-213

LGFI (LOAD IMMEDIATE) instruction 7-214

LGFR (LOAD) binary instruction 7-213

LGFRL (LOAD RELATIVE LONG) binary instruction 7-214

LGH (LOAD HALFWORD) instruction 7-222

LGHI (LOAD HALFWORD IMMEDIATE) instruction 7-222

LGHR (LOAD HALFWORD) binary instruction 7-221

LGHRL (LOAD HALFWORD RELATIVE LONG) binary instruction 7-222

LGOC (LOAD ON CONDITION) instruction 7-228

LGR (LOAD) binary instruction 7-213

LGRL (LOAD RELATIVE LONG) binary instruction 7-214

LGROC (LOAD ON CONDITION) instruction 7-228

LH (LOAD HALFWORD) instruction 7-221

- examples A-24

LHH (LOAD HALFWORD HIGH) instruction 7-222

LHI (LOAD HALFWORD IMMEDIATE) instruction 7-222

LHR (LOAD HALFWORD) binary instruction 7-221

LHRL (LOAD HALFWORD RELATIVE LONG) binary instruction 7-222

LHY (LOAD HALFWORD) instruction 7-221

LIFO (last in first out) queuing

- example for lock and unlock A-48

light

- See indicator

limit mode (I/O) 15-3

link information

- for BRANCH AND LINK instruction 7-33
- for BRANCH AND SAVE AND SET MODE instruction 7-35
- for BRANCH AND SAVE instruction 7-34

linkage first table (LFT) 5-33

- designation (LFTD) 5-32
- length (LFTL) 5-32
- origin (LFTO) 5-32

linkage for subroutines 5-11

linkage index (LX) 5-30

linkage second table (LST) 5-33

- origin (LSTO) 5-33

linkage stack 5-66, 5-74

- associated PER events 5-71
- associated trace entries 5-71
- branch state entry 10-11
- entry address 5-74
- entry descriptor 5-75
- entry-type code 5-75
- handling of information in 5-70
- header entry 5-76
- instructions 5-67
- introduction 5-72
- next-entry size 5-75
- operations 5-72

- control 5-74
- program-call state entry 10-85
- remaining free space 5-75
- section 5-72
 - identification 5-75
- state entry 5-77
- trailer entry 5-76
- linkage-stack functions 5-67
- linkage table (LT) 5-33
 - designation (LTD) 5-32
 - length (LTL) 5-32
 - origin (LTO) 5-32
- list-directed initial-program-load (IPL) function 1-14
- list-directed IPL (list-directed initial program loading) 17-19
- little endian 7-231, 7-311
- LLC (LOAD LOGICAL CHARACTER) instruction 7-224
- LLCH (LOAD LOGICAL CHARACTER HIGH) instruction 7-224
- LLCR (LOAD LOGICAL CHARACTER) binary instruction 7-223
- LLGC (LOAD LOGICAL CHARACTER) instruction 7-224
- LLGCR (LOAD LOGICAL CHARACTER) binary instruction 7-223
- LLGF (LOAD LOGICAL) instruction 7-223
- LLGFR (LOAD LOGICAL) instruction 7-223
- LLGFRL (LOAD LOGICAL RELATIVE LONG) binary instruction 7-223
- LLGH (LOAD LOGICAL HALFWORD) instruction 7-224
- LLGHR (LOAD LOGICAL HALFWORD) binary instruction 7-224
- LLGHRL (LOAD LOGICAL RELATIVE LONG) binary instruction 7-224
- LLGT (LOAD LOGICAL THIRTY ONE BITS) instruction 7-226
- LLGTR (LOAD LOGICAL THIRTY ONE BITS) instruction 7-226
- LLH (LOAD LOGICAL HALFWORD) instruction 7-224
- LLHH (LOAD LOGICAL HALFWORD HIGH) instruction 7-225
- LLHR (LOAD LOGICAL HALFWORD) binary instruction 7-224
- LLHRL (LOAD LOGICAL RELATIVE LONG) binary instruction 7-224
- LLIHF (LOAD LOGICAL IMMEDIATE) instruction 7-225
- LLIHH (LOAD LOGICAL IMMEDIATE) instruction 7-225
- LLIHL (LOAD LOGICAL IMMEDIATE) instruction 7-225
- LLILF (LOAD LOGICAL IMMEDIATE) instruction 7-225
- LLILH (LOAD LOGICAL IMMEDIATE) instruction 7-225
- LLILL (LOAD LOGICAL IMMEDIATE) instruction 7-225
- LM (LOAD MULTIPLE) instruction 7-226
- LMD (LOAD MULTIPLE DISJOINT) instruction 7-227
- LMG (LOAD MULTIPLE) instruction 7-226
- LMH (LOAD MULTIPLE HIGH) instruction 7-227
- LMY (LOAD MULTIPLE) instruction 7-226
- LNDBR (LOAD NEGATIVE) BFP instruction 19-33
- LNDFR (LOAD NEGATIVE) floating-point-support instruction 9-33
- LNDR (LOAD NEGATIVE) HFP instruction 18-15
- LNEBR (LOAD NEGATIVE) BFP instruction 19-33
- LNER (LOAD NEGATIVE) HFP instruction 18-15
- LNGFR (LOAD NEGATIVE) binary instruction 7-227
- LNGR (LOAD NEGATIVE) binary instruction 7-227
- LNR (LOAD NEGATIVE) binary instruction 7-227
- LNXR (LOAD NEGATIVE) BFP instruction 19-33
- LNXR (LOAD NEGATIVE) HFP instruction 18-15
- LOAD ACCESS MULTIPLE instructions 7-214
- LOAD ADDRESS EXTENDED instruction 7-215
- LOAD ADDRESS instructions 7-215
 - examples A-23
- LOAD ADDRESS RELATIVE LONG instruction 7-216
- LOAD ADDRESS SPACE PARAMETERS instruction 10-34
- LOAD AND ADD instructions 7-217
- LOAD AND ADD LOGICAL instructions 7-217
- LOAD AND AND instructions 7-218
- LOAD AND EXCLUSIVE OR instructions 7-218
- LOAD AND OR instructions 7-219
- LOAD AND TEST BFP instructions 19-30
- LOAD AND TEST binary instruction 7-219
- LOAD AND TEST DFP instructions 20-36
- LOAD AND TEST HFP instructions 18-13
- LOAD binary instructions 7-213
 - example A-23
- LOAD BYTE HIGH instruction 7-221
- LOAD BYTE instructions 7-220
- load-clear key 12-3
- load-clear-list-directed key 12-3
- LOAD COMPLEMENT BFP instructions 19-30
- LOAD COMPLEMENT binary instruction 7-221
- LOAD COMPLEMENT floating-point-support instruction 9-30
- LOAD COMPLEMENT HFP instructions 18-14
- LOAD CONTROL instructions 10-44
- LOAD floating-point-support instructions 9-30
- LOAD FP INTEGER BFP instructions 19-31
- LOAD FP INTEGER DFP instructions 20-37

LOAD FP INTEGER HFP instructions 18-14
 LOAD FPC AND SIGNAL floating-point-support instruction 9-31
 LOAD FPC floating-point-support instruction 9-31
 LOAD FPR FROM GR floating-point-support instruction 9-33
 LOAD GR FROM FPR floating-point-support instruction 9-33
 LOAD HALFWORD HIGH instruction 7-222
 LOAD HALFWORD IMMEDIATE instruction 7-222
 LOAD HALFWORD instruction
 examples A-24
 LOAD HALFWORD instructions 7-221
 LOAD HALFWORD RELATIVE LONG instructions 7-222
 LOAD HIGH instruction 7-223
 LOAD IMMEDIATE instruction 7-214
 load indicator 12-3
 LOAD LENGTHENED BFP instructions 19-32
 LOAD LENGTHENED DFP instructions 20-40
 LOAD LENGTHENED HFP instructions 18-15
 LOAD LOGICAL CHARACTER HIGH instruction 7-224
 LOAD LOGICAL CHARACTER instructions 7-223
 LOAD LOGICAL HALFWORD HIGH instruction 7-225
 LOAD LOGICAL HALFWORD instructions 7-224
 LOAD LOGICAL HALFWORD RELATIVE LONG instructions 7-224
 LOAD LOGICAL IMMEDIATE instructions 7-225
 LOAD LOGICAL instructions 7-223
 LOAD LOGICAL RELATIVE LONG binary instruction 7-223
 LOAD LOGICAL THIRTY ONE BITS instructions 7-226
 LOAD MULTIPLE DISJOINT instruction 7-227
 LOAD MULTIPLE HIGH instruction 7-227
 LOAD MULTIPLE instructions 7-226
 LOAD NEGATIVE BFP instructions 19-33
 LOAD NEGATIVE binary instruction 7-227
 LOAD NEGATIVE floating-point-support instruction 9-33
 LOAD NEGATIVE HFP instructions 18-15
 load-normal key 12-3
 LOAD ON CONDITION instructions 7-228
 LOAD PAGE-TABLE-ENTRY ADDRESS instruction 10-44
 LOAD PAIR DISJOINT instructions 7-219
 LOAD PAIR FROM QUADWORD instruction 7-230
 LOAD POSITIVE BFP instructions 19-33
 LOAD POSITIVE binary instruction 7-230
 LOAD POSITIVE floating-point-support instruction 9-34
 LOAD POSITIVE HFP instructions 18-16
 LOAD PSW EXTENDED instruction 10-48
 LOAD PSW instruction 10-47
 LOAD REAL ADDRESS instructions 10-49
 LOAD RELATIVE LONG instruction 7-214
 LOAD REVERSED instructions 7-230
 LOAD ROUNDED BFP instructions 19-34
 LOAD ROUNDED DFP instructions 20-41
 LOAD ROUNDED HFP instructions 18-16
 load state 4-2, 4-3
 during CCW-type IPL 4-56
 load/store-on-condition facility 1-14
 facility indication 4-68
 load-unit-address controls 12-4
 LOAD USING REAL ADDRESS instruction 10-53
 load-with-dump key 12-3
 LOAD ZERO floating-point-support instructions 9-34
 loading, initial
 See IML, IPL
 LOC (LOAD ON CONDITION) instruction 7-228
 location 3-2
 not available in configuration 6-17
 See also address
 lock A-47
 example with FIFO queuing A-49
 example with LIFO queuing A-48
 lock used by PERFORM LOCKED OPERATION instruction 7-280
 logical
 arithmetic (unsigned binary) 7-7
 comparison 7-8
 connective
 AND 7-32, 7-33
 EXCLUSIVE OR 7-206, 7-207
 OR 7-251, 7-252
 data 7-5
 logical address 3-5
 handling by DAT 3-36
 logical-path mask
 See LPM
 I/O-interruption
 See I/O-interruption subclass mask
 logical string assist 1-20
 logically partitioned (LPAR) mode 1-24, 1-26
 logout
 fixed
 assigned storage locations for 3-67
 machine-check 11-24
 subchannel (I/O) 16-47
 long binary-floating-point number 19-2
 long decimal-floating-point number 20-3
 long-displacement facility 1-14
 facility indication 4-67
 long hexadecimal-floating-point number 18-2
 long I/O block 16-24

- loop control 5-11
- loop of interruptions
 - See* string of interruptions
- low-address protection 3-13
 - control bit 3-13
 - exception for 6-30
- LPAR (logically partitioned) mode 1-24
- LPAR (logically partitioned)mode 1-26
- LPD (LOAD PAIR DISJOINT instructions 7-229
- LPDBR (LOAD POSITIVE) BFP instruction 19-33
- LPDFR (LOAD POSITIVE) floating-point-support instruction 9-34
- LPDG (LOAD PAIR DISJOINT instructions 7-229
- LPDR (LOAD POSITIVE) HFP instruction 18-16
- LPEBR (LOAD POSITIVE) BFP instruction 19-33
- LPER (LOAD POSITIVE) HFP instruction 18-16
- LPGFR (LOAD POSITIVE) binary instruction 7-230
- LPGR (LOAD POSITIVE) binary instruction 7-230
- LPM (logical-path mask) 15-5, 15-28, 15-30
 - effect on system performance of 15-11
 - used for IPL 17-17
- LPQ (LOAD PAIR FROM QUADWORD) instruction 7-230
- LPR (LOAD POSITIVE) binary instruction 7-230
- LPSW (LOAD PSW) instruction 10-47
- LPSWE (LOAD PSW EXTENDED) instruction 10-48
- LPTEA (LOAD PAGE-TABLE-ENTRY ADDRESS) instruction 10-44
- LPUM (last-path-used mask) 15-6
 - in ESW 16-49
 - field-validity flag for (in subchannel logout) 16-49
- LPXBR (LOAD POSITIVE) BFP instruction 19-33
- LPXR (LOAD POSITIVE) HFP instruction 18-16
- LR (LOAD) binary instruction 7-213
- LRA (LOAD REAL ADDRESS) instruction 10-49
- LRAG (LOAD REAL ADDRESS) instruction 10-49
- LRAY (LOAD REAL ADDRESS) instruction 10-49
- LRDR (LOAD ROUNDED) HFP instruction 18-16
- LRER (LOAD ROUNDED) HFP instruction 18-16
- LRL (LOAD RELATIVE LONG) binary instruction 7-214
- LROC (LOAD ON CONDITION) instruction 7-228
- LRV (LOAD REVERSED) instruction 7-231
- LRVG (LOAD REVERSED) instruction 7-231
- LRVGR (LOAD REVERSED) instruction 7-231
- LRVH (LOAD REVERSED) instruction 7-231
- LRVR (LOAD REVERSED) instruction 7-230
- LST (linkage second table) 5-33
- LSTO (linkage-second-table origin) 5-33
- LSX (linkage first index)
 - translation exception 6-25
- LSX (linkage second index)
 - invalid bit 5-33
- LT (linkage table) 5-33
- LT (LOAD AND TEST) instruction 7-220
- LTD (linkage-table designation) 5-32
- LTDBR (LOAD AND TEST) BFP instruction 19-30
- LTDR (LOAD AND TEST) HFP instruction 18-13
- LTDR (LOAD AND TEST) DFP instruction 20-36
- LTEBR (LOAD AND TEST) BFP instruction 19-30
- LTFR (LOAD AND TEST) HFP instruction 18-13
- LTG (LOAD AND TEST) instruction 7-220
- LTGF (LOAD AND TEST) binary instruction 7-220
- LTGFR (LOAD AND TEST) binary instruction 7-220
- LTGR (LOAD AND TEST) binary instruction 7-219
- LTG (LOAD AND TEST) instruction 7-220
- LTO (linkage-table origin) 5-32
- LTR (LOAD AND TEST) binary instruction 7-219
- LTXBR (LOAD AND TEST) BFP instruction 19-30
- LTXR (LOAD AND TEST) HFP instruction 18-13
- LTXTR (LOAD AND TEST) DFP instruction 20-36
- LURA (LOAD USING REAL ADDRESS) instruction 10-53
- LURAG (LOAD USING REAL ADDRESS) instruction 10-53
- LUV (left-units view) of floating-point numbers 9-2
- LX (linkage index) 5-30
 - invalid bit 5-33
 - translation exception 6-25
- LXD (LOAD LENGTHENED) HFP instruction 18-15
- LXDB (LOAD LENGTHENED) BFP instruction 19-33
- LXDBR (LOAD LENGTHENED) BFP instruction 19-32
- LXDR (LOAD LENGTHENED) HFP instruction 18-15
- LXDTR (LOAD LENGTHENED) DFP instruction 20-40
- LXE (LOAD LENGTHENED) HFP instruction 18-15
- LXEB (LOAD LENGTHENED) BFP instruction 19-33
- LXEBR (LOAD LENGTHENED) BFP instruction 19-32
- LXER (LOAD LENGTHENED) HFP instruction 18-15
- LXR (LOAD) floating-point-support instruction 9-30
- LY (LOAD) binary instruction 7-213
- LZDR (LOAD ZERO) floating-point-support instruction 9-34
- LZER (LOAD ZERO) floating-point-support instruction 9-34
- LZXR (LOAD ZERO) floating-point-support instruction 9-34

M

- M (mega) xxvi
- M (MULTIPLY) binary instruction 7-247, 7-248
 - example A-28
- machine check
 - See also* malfunction

- handling of malfunction detected as part of I/O 11-5
- interruption 6-14, 11-10
 - action 11-11
 - code (MCIC) 3-65, 11-13
 - floating conditions 11-23
 - machine check interruption 11-23
 - mask in the PSW 4-5
 - subclass masks in control register 11-23
- logout 11-24
- mask
 - in the PSW 4-5
- machine-check architectural-mode identification 3-63
- machine-type number (in CPU ID) 10-127
- MAD (MULTIPLY AND ADD) HFP instruction 18-19
- MADB (MULTIPLY AND ADD) BFP instruction 19-37
- MADBR (MULTIPLY AND ADD) BFP instruction 19-37
- MADR (MULTIPLY AND ADD) HFP instruction 18-18
- MAE (MULTIPLY AND ADD) HFP instruction 18-19
- MAEB (MULTIPLY AND ADD) BFP instruction 19-37
- MAEBR (MULTIPLY AND ADD) BFP instruction 19-37
- MAER (MULTIPLY AND ADD) HFP instruction 18-18
- main storage 3-2
 - See also* storage
 - effect of power-on reset on 4-55
 - See also* storage
 - shared (in multiprocessing) 4-57
- malfunction 11-1
 - at channel subsystem 16-29
 - at I/O device 16-29
 - correction of 11-2
 - effect on manual operation 12-1
 - from DIAGNOSE instruction 10-20
 - indication of 11-4
 - machine-check handling for when detected as part of I/O 11-5
- malfunction alert (external interruption) 6-12
 - when entering check-stop state 11-10
- manual indicator 12-4
 - See also* stopped state
- manual operation 12-1
 - controls
 - address-compare 12-1
 - alter-and-display 12-2
 - IML 12-3
 - load-unit-address 12-4
 - power 12-4
 - rate 12-4
 - TOD-clock 12-6
 - effect on CPU signaling 4-62
 - keys
 - interrupt 12-3
 - load-clear 12-3
 - load-clear-list-directed 12-3
 - load-normal 12-3
 - load-with-dump 12-3
 - restart 12-4
 - start 12-5
 - stop 12-5
 - store-status 12-5
 - system-reset-clear 12-5
 - system-reset-normal 12-5
- masks 6-5
 - See also* I/O interruption, interruption for IEEE exceptions 9-10
 - in BRANCH ON CONDITION instruction 7-37
 - in BRANCH RELATIVE ON CONDITION instruction 7-40
 - in COMPARE LOGICAL CHARACTERS UNDER MASK instruction 7-134
 - in INSERT CHARACTERS UNDER MASK instruction 7-212
 - in PSW 4-5
 - in STORE CHARACTERS UNDER MASK instruction 7-303
 - in TEST UNDER MASK instruction 7-317
- monitor 6-26
- path-management 15-2, 15-28, 15-30
- PER-event 4-24
- program-interruption 6-14
- subclass
 - See* subclass-mask bits
- maximum negative number 7-5
- MBA (measurement-block address) 15-9
- MC (MONITOR CALL) instruction 7-231
- MCIC (machine-check-interruption code) 3-65, 11-13
- MD (MULTIPLY) HFP instruction 18-17
- MDB (MULTIPLY) BFP instruction 19-35
- MDBR (MULTIPLY) BFP instruction 19-35
- MDE (MULTIPLY) HFP instruction 18-17
- MDEB (MULTIPLY) BFP instruction 19-35
- MDEBR (MULTIPLY) BFP instruction 19-35
- MDER (MULTIPLY) HFP instruction 18-17
- MDR (MULTIPLY) HFP instruction 18-17
 - example A-43
- MDTR (MULTIPLY) DFP instruction 20-43
- MDTRA (MULTIPLY) DFP instruction 20-43
- ME (MULTIPLY) HFP instruction 18-17
- measurement
 - block (I/O)
 - address 17-7
 - format 17-7
 - origin 17-7
 - device-connect-time 17-9

- extended-measurement-word 17-10
- measurement-block update (I/O) 17-2
- measurement block (I/O) 17-3
 - data check 16-48
 - index 15-7
 - key (MBK)
 - used as access key 3-10
 - multiple use of 15-11
 - program check 16-48
 - protection check 16-48
 - update enable 15-3
- measurement-block-format control (I/O) 15-8
- measurement data (I/O)
 - accumulated 17-3
 - effect of CSCH on 14-6
 - effect of HSCH on 14-7
- measurement-mode control (I/O) 15-3
- MEE (MULTIPLY) HFP instruction 18-17
- MEEB (MULTIPLY) BFP instruction 19-35
- MEEBR (MULTIPLY) BFP instruction 19-35
- MEER (MULTIPLY) HFP instruction 18-17
- MER (MULTIPLY) HFP instruction 18-17
- message byte (in EDIT) 8-8
- message-security assist 1-15
 - facility indication 4-67
- message-security-assist extension 1 1-15
- message-security-assist extension 2 1-15
- message-security-assist extension 3 1-15
 - facility indication 4-68
- message-security-assist extension 4 1-16
 - facility indication 4-68
- MFY (MULTIPLY) binary instruction 7-247
- MGHI (MULTIPLY HALFWORD IMMEDIATE)
 - instruction 7-248
- MH (MULTIPLY HALFWORD) instruction 7-248
 - example A-29
- MHI (MULTIPLY HALFWORD IMMEDIATE)
 - instruction 7-248
- MHY (MULTIPLY HALFWORD) binary instruction 7-248
- MIDA (modified-indirect-data addressing) 15-63
- MIDA flag in CCW 15-33
 - invalid 16-27
- MIDAW (modified indirect-data-address word)
 - invalid address specification of 16-26
- MIDAW (modified-indirect-data-address word) 15-63
 - check (in subchannel logout) 16-48
- MIDAW (modified indirect-data-address word)
 - invalid address of 16-25
- MIDAW control 15-28
- ML (MULTIPLY LOGICAL) instruction 7-249
- MLG (MULTIPLY LOGICAL) instruction 7-249
- MLGR (MULTIPLY LOGICAL) instruction 7-249
- MLR (MULTIPLY LOGICAL) instruction 7-249

- modal instructions 1-5, 7-9
- mode
 - access-register 3-36
 - addressing
 - See addressing mode
 - architectural
 - See architectural mode
 - burst (channel-path operation) 13-3
 - byte-multiplex mode (channel-path operation) 13-3
 - home-space 3-36
 - incorrect-length-indication 15-28
 - incorrect-length-suppression 15-28
 - indicator
 - architectural 12-2
 - multipath
 - See multipath mode
 - primary-space 3-36
 - real 3-36
 - requirements for semiprivileged instructions 5-24
 - secondary-space 3-36
 - single-path 15-4, 15-23
 - translation 3-36
- mode-trace-control bit 4-12
- model number (in CPU ID) 10-127
- modifiable area (in linkage-stack state entry) 5-79
- modification control 15-26
- modified-CCW-indirect-data-addressing control 15-28
- modified-CCW-indirect-data-addressing facility 1-17
- MODIFY STACKED STATE instruction 10-53
- MODIFY SUBCHANNEL instruction 14-8
- MONITOR CALL instruction 7-231
- monitor-class number 6-26
 - assigned storage locations for 3-62
- monitor code 6-26, 7-232
 - assigned storage locations for 3-65
- monitor event 6-26, 7-232
 - counting operation 7-232
 - program interruption 7-232
- monitor masks 6-26, 7-231
- monitoring
 - See also measurement
 - channel-subsystem 17-1
 - for PER events
 - See PER
 - with MONITOR CALL 6-26, 7-231
- monitor-class number 7-231
- MOVE instructions 7-233
 - examples A-21, A-24
- move-inverse facility 7-233
- MOVE INVERSE instruction 7-233
 - example A-25
- MOVE LONG EXTENDED instruction 7-238

MOVE LONG instruction 7-234
 examples A-26
 MOVE LONG UNICODE instruction 7-241
 MOVE NUMERICS instruction 7-244
 example A-26
 MOVE PAGE instruction 10-54, 10-70
 MOVE STRING instruction 7-245
 example A-27
 MOVE TO PRIMARY instruction 10-56
 MOVE TO SECONDARY instruction 10-56
 MOVE WITH DESTINATION KEY instruction 10-58
 MOVE WITH KEY instruction 10-59
 MOVE WITH OFFSET instruction 7-246
 example A-27
 move-with-optional-specifications facility 1-18
 facility indication 4-67
 MOVE WITH OPTIONAL SPECIFICATIONS
 instruction 10-60
 MOVE WITH SOURCE KEY instruction 10-63
 MOVE ZONES instruction 7-246
 example A-28
 MP (MULTIPLY DECIMAL) instruction 8-11
 example A-39
 MR (MULTIPLY) binary instruction 7-247
 example A-28
 MS (MULTIPLY SINGLE) instruction 7-249
 MSCH (MODIFY SUBCHANNEL) instruction 14-8
 MSD (MULTIPLY AND SUBTRACT) HFP instruction
 18-19
 MSDB (MULTIPLY AND SUBTRACT) BFP
 instruction 19-37
 MSDBR (MULTIPLY AND SUBTRACT) BFP
 instruction 19-37
 MSDR (MULTIPLY AND SUBTRACT) HFP
 instruction 18-19
 MSE (MULTIPLY AND SUBTRACT) HFP instruction
 18-19
 MSEB (MULTIPLY AND SUBTRACT) BFP
 instruction 19-37
 MSEBR (MULTIPLY AND SUBTRACT) BFP
 instruction 19-37
 MSER (MULTIPLY AND SUBTRACT) HFP
 instruction 18-19
 MSFI (MULTIPLY SINGLE IMMEDIATE) instruction
 7-250
 MSG (MULTIPLY SINGLE) instruction 7-249
 MSGF (MULTIPLY SINGLE) instruction 7-249
 MSGFI (MULTIPLY SINGLE IMMEDIATE) instruction
 7-250
 MSGFR (MULTIPLY SINGLE) instruction 7-249
 MSGR (MULTIPLY SINGLE) instruction 7-249
 MSR (MULTIPLY SINGLE) instruction 7-249
 MSS (multiple subchannel set facility) 17-29
 MSS (multiple-subchannel-set) facility 1-18
 MSTA (MODIFY STACKED STATE) instruction
 10-53
 MSY (MULTIPLY SINGLE) instruction 7-249
 multipath mode 15-4
 entering 15-23
 multiple-access storage references 5-96
 multiple-subchannel-set facility 1-18
 multiple subchannel set facility (MSS) 17-29
 MULTIPLY AND ADD BFP instructions 19-37
 MULTIPLY AND ADD HFP instructions 18-18
 MULTIPLY AND SUBTRACT BFP instructions 19-37
 MULTIPLY AND SUBTRACT HFP instructions 18-19
 MULTIPLY BFP instructions 19-35
 MULTIPLY binary instructions 7-247
 examples A-28
 MULTIPLY DECIMAL instruction 8-11
 example A-39
 MULTIPLY DFP instructions 20-43
 MULTIPLY HALFWORD IMMEDIATE instruction
 7-248
 MULTIPLY HALFWORD instruction 7-248
 example A-29
 MULTIPLY HFP instructions 18-17
 example A-43
 MULTIPLY LOGICAL instructions 7-249
 MULTIPLY SINGLE IMMEDIATE instructions 7-250
 MULTIPLY SINGLE instructions 7-249
 multiprocessing 4-57
 manual operations for 12-6
 programming considerations for 8-4, A-45
 programming examples A-45
 timing-facility interruptions for 4-43
 TOD clock for 4-38
 multiprogramming examples A-45
 MVC (MOVE) instruction 7-232
 examples A-21, A-24
 MVCDK (MOVE WITH DESTINATION KEY)
 instruction 10-58
 MVCIN (MOVE INVERSE) instruction 7-233
 example A-25
 MVCK (MOVE WITH KEY) instruction 10-59
 MVCL (MOVE LONG) instruction 7-234
 examples A-26
 MVCLE (MOVE LONG EXTENDED) instruction
 7-238
 MVCLU (MOVE LONG UNICODE) instruction 7-241
 MVCOS (MOVE WITH OPTIONAL
 SPECIFICATIONS) instruction 10-60
 MVCP (MOVE TO PRIMARY) instruction 10-56
 MVCS (MOVE TO SECONDARY) instruction 10-56
 MVCSK (MOVE WITH SOURCE KEY) instruction
 10-63
 MVGHI (MOVE) instruction 7-233
 MVHHI (MOVE) instruction 7-232

- MVHI (MOVE) instruction 7-233
- MVI (MOVE) instruction 7-233
 - example A-25
- MVIY (MOVE) instruction 7-233
- MVN (MOVE NUMERICS) instruction 7-244
 - example A-26
- MVO (MOVE WITH OFFSET) instruction 7-246
 - example A-27
- MVPG (MOVE PAGE) instruction 10-54, 10-70
- MVST (MOVE STRING) instruction 7-245
 - example A-27
- MVZ (MOVE ZONES) instruction 7-246
 - example A-28
- MXBR (MULTIPLY) BFP instruction 19-35
- MXD (MULTIPLY) HFP instruction 18-17
- MXDB (MULTIPLY) BFP instruction 19-35
- MXDBR (MULTIPLY) BFP instruction 19-35
- MXDR (MULTIPLY) HFP instruction 18-17
- MXR (MULTIPLY) HFP instruction 18-17
- MXTR (MULTIPLY) DFP instruction 20-43
- MXTRA (MULTIPLY) DFP instruction 20-43

N

- N (AND) instruction 7-31
- N condition (I/O) 16-13
- NaN (not-a-number) 9-2, 19-4, 20-7
- NC (AND) instruction 7-32
- NDNR (AND) binary instruction 7-31
- NDXGR (EXCLUSIVE OR) binary instruction 7-205
- near-valid CBC 11-2
 - in storage 11-5
- negative zero
 - binary 7-5
 - decimal 8-3
 - example A-5
- new PSW 4-3
 - assigned storage locations for 3-66
 - fetches during interruption 6-2
- next-entry size (in linkage stack) 5-75
- NG (AND) instruction 7-32
- NGR (AND) instruction 7-31
- NGRK (AND) binary instruction 7-31
- NI (AND) instruction 7-32
 - example A-8
- NIHF (AND IMMEDIATE) instruction 7-32
- NIHH (AND IMMEDIATE) instruction 7-32
- NIHL (AND IMMEDIATE) instruction 7-33
- NILF (AND IMMEDIATE) instruction 7-33
- NILH (AND IMMEDIATE) instruction 7-33
- NILL (AND IMMEDIATE) instruction 7-33
- NIY (AND) instruction 7-32
- no-operation
 - instruction (BRANCH ON CONDITION) 7-37

- instruction (BRANCH RELATIVE ON CONDITION) 7-40
- node (of tree structure) 7-334
- noninterlocked-update storage reference 5-94
- nonnumeric entities
 - decimal 20-7
 - IEEE 9-2
- nonquiescing key-setting facility 1-18
- nontrap action
 - IEEE 9-17
- nonvolatile storage 3-2
- normal numbers
 - class of BFP data 19-4
 - class of DFP data 20-7
- normalization
 - of HFP numbers 18-2
- not-a-number (NaN) 9-2, 19-4, 20-7
- not operational
 - as channel-path state 16-13
 - See path-not-operational bit
 - as CPU state 4-62
 - as TOD-clock state 4-39
- not set (TOD-clock state) 4-38
- NR (AND) instruction 7-31
- nullification
 - exceptions to 5-23
 - for exigent machine-check conditions 11-10
 - of instruction execution 5-20
 - of unit of operation 5-21
- numbering
 - of addresses (byte locations) 3-2
 - of bits 3-3
- numbers
 - binary 7-5
 - examples A-2
 - binary-floating-point 19-2
 - CPU-model 10-127
 - decimal 8-1
 - examples A-4
 - device 13-6
 - hexadecimal 5-6
 - hexadecimal-floating-point 18-2
 - examples A-5
 - machine-type 10-127
- numeric bits 8-1
 - moving of 7-244
- Numeric part of PC number 5-79
- NY (AND) instruction 7-32

O

- O (OR) instruction 7-250
- OC (OR) instruction 7-251

- OEMI (original equipment manufacturers information)
 - for I/O interface
 - publication referenced xxvii
- offset of symbol-translation table 7-150
- OG (OR) instruction 7-250
- OGR (OR) instruction 7-250
- OGRK (OR) binary instruction 7-250
- OI (OR) instruction 7-251
 - example A-29
 - example of problem with A-45
- OIHF (OR IMMEDIATE) instruction 7-251
- OIHH (OR IMMEDIATE) instruction 7-251
- OIHL (OR IMMEDIATE) instruction 7-251
- OILF (OR IMMEDIATE) instruction 7-251
- OILH (OR IMMEDIATE) instruction 7-252
- OILL (OR IMMEDIATE) instruction 7-252
- OIY (OR) instruction 7-251
- old PSW 6-2
 - assigned storage locations for 3-66
- one's complement binary notation 7-5
 - used for SUBTRACT LOGICAL instruction 7-314
 - used for SUBTRACT LOGICAL WITH BORROW instruction 7-315
- op code
 - See operation code
- operand 5-3
 - access identification 3-63
 - access of 5-93
 - for I/O instructions 14-2
 - address generation for 5-8
 - exception 6-27
 - immediate 5-6
 - length of 5-3
 - overlap of
 - for decimal instructions 8-3
 - for general instructions 7-5
 - register for 5-6
 - sequence of references for 5-93
 - storage 5-7
 - types of (fetch, store, update) 5-93
 - used for result 5-3
- operating state 4-2
- operation
 - I/O
 - See I/O operations
 - unit of 5-20
- operation code (op code) 5-2
 - invalid 6-27
- operation exception 6-27
- operation-request block
 - See ORB
- operator facilities 2-6, 12-1
 - basic 12-1
- operator intervening (signal-processor status) 4-65
- OR (OR) instruction 7-250
- OR IMMEDIATE instructions 7-251
- OR instructions 7-250
 - example of problem with OR immediate A-45
 - examples A-29
- ORB (operation-request block) 15-24
 - channel-program-type control 15-27
 - CSS priority 15-29
 - designating a CCW channel program 15-25
 - extension control in 15-28, 15-30
 - interruption parameter in 15-25
 - invalid 16-27
 - logical-path mask (LPM) in 15-28, 15-30
- orders (I/O) 13-8
- orders (signal-processor) 4-58
 - conditional emergency signal 4-61
 - conditions precluding response to 4-62
 - CPU reset 4-59
 - emergency signal 4-58
 - emergency signal, conditional 4-61
 - external call 4-58
 - initial CPU reset 4-59
 - restart 4-59
 - sense 4-58
 - set architecture 4-60
 - set prefix 4-59
 - start 4-58
 - stop 4-58
 - stop and store status 4-59
 - store status at address 4-59
- ORK (OR) binary instruction 7-250
- overflow
 - binary 7-6
 - example A-2
 - decimal 6-22
 - exponent
 - See exponent overflow
 - fixed-point 6-24, 7-7
 - in CRW 17-26
- overlap
 - destructive 7-234, 7-238, 7-242
 - operand 5-87
 - for decimal instructions 8-3
 - for general instructions 7-5
 - operation 5-85
- OY (OR) instruction 7-250

P

- P (peta) xxvi
- PACK ASCII instruction 7-253
- PACK instruction 7-252
 - example A-29
- PACK UNICODE instruction 7-254

- packed decimal numbers 8-1
 - conversion of to zoned format 7-331
 - conversion to from zoned format 7-252
- padding byte
 - for COMPARE LOGICAL LONG EXTENDED instruction 7-138
 - for COMPARE LOGICAL LONG instruction 7-135
 - for MOVE LONG EXTENDED instruction 7-238
 - for MOVE LONG instruction 7-234
- page 3-35
- page-frame real address (PFRA) 3-44
- PAGE IN instruction 10-64
- page index (PX) 3-35
- page-invalid bit (in page-table entry) 3-44
- PAGE OUT instruction 10-65
- page protection
 - exception for 6-30
- page swapping 3-34
- page table 3-44
 - lookup 3-51
 - origin (PTO) 3-42
- page-translation exception 6-27
 - as an access exception 6-40, 6-46
- PALB (PURGE ALB) instruction 10-107
- PAM (path-available mask) 15-7
 - effect of reconfiguration on 15-11
 - effect of resetting on 15-11
 - effect on allegiance of 15-12
- parallel-I/O channel-to-channel adapter
 - publication referenced xxvii
- parallel-I/O interface 13-3
 - OEMI publication referenced xxvii
- parameter
 - external-interruption 6-9
 - assigned storage locations for 3-61
 - I/O-interruption
 - See I/O-interruption parameter
 - register for SIGNAL PROCESSOR 4-59, 10-124
 - translation 3-36
- parity bit 11-2
- parsing-enhancement facility 1-18
 - facility indication 4-67
- partial completion of instruction execution 5-20
- PASCE (primary address-space-control element) 3-38
- PASN (primary address-space number) 3-20
 - in trace entry 4-21
- PASTE (primary AST entry) 5-32
- PASTEIN (primary-AST-entry instance number) 3-22
- PASTEO (primary-AST-entry origin) 5-32, 5-50
- path
 - See channel path
- path available for selection 15-13
- path management 13-9
 - for clear function 15-14, 15-22
 - for halt function 15-16
 - for start function and resume function 15-20
- path-management-control word
 - See PMCW
- path-management masks
 - last-path-used mask
 - See LPUM
 - logical-path mask
 - See LPM
 - path-available mask
 - See PAM
 - path-installed mask
 - See PIM
 - path-not-operational mask
 - See PNOM
 - path-operational mask
 - See POM
- path-not-operational bit (N) in SCSW 16-13
- path-not-operational condition 15-5
- path verification required
 - indicator for (in ERW) 16-52
- pattern (in EDIT) 8-7
- PC-cp (PROGRAM CALL instruction, to current primary) 10-86
- PC number 10-83
 - in linkage-stack state entry 5-79
 - in trace entry 4-20
 - translation 5-30
- PC-ss (PROGRAM CALL instruction, with space switching) 10-86
- PC-translation-specification exception 6-28
- PC-type bit 5-35
- PCC (PERFORM CRYPTOGRAPHIC COMPUTATION) instruction 7-255
- PCI (program-controlled interruption) 15-59
 - as flag in CCW 15-32
 - intermediate interruption condition for 16-18
 - subchannel status for 16-24
- PCKMO (PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATION) instruction 10-66
- pending channel reports
 - effect of I/O-system reset on 17-15
- pending interruption
 - See interruption pending
- PER (program-event recording)
 - access identification 3-62, 4-27
 - address 4-27
 - assigned storage locations for 3-62
 - address-space-control element (ASCE)
 - identification 4-27
 - ASCE (address-space-control element)
 - identification 4-27

ATMD (addressing-and-translation-mode identification) 4-26
 code 4-26
 assigned storage locations for 3-62
 events 4-23
 extensions 1-24
 instruction-fetching event 4-31
 masks
 bit in the PSW 4-5
 PER-event 4-24
 priority of indication 4-29
 program-interruption condition 6-28
 storage-alteration event 4-32
 storage-area designation 4-30
 ending address 4-24
 starting address 4-24
 wraparound 4-30
 store-using-real-address event 4-32
 successful-branching event 4-31
 PER-3 facility 1-18
 breaking-event-address recording 4-36
 PER instruction-fetching nullification 4-23
 PERFORM CRYPTOGRAPHIC COMPUTATION instruction 7-255
 PERFORM CRYPTOGRAPHIC KEY MANAGEMENT OPERATION instruction 10-66
 PERFORM FLOATING-POINT OPERATION floating-point-support instruction 9-34
 PERFORM FRAME MANAGEMENT FUNCTION instruction 10-70
 PERFORM LOCKED OPERATION instruction 7-270
 example A-51
 PERFORM TIMING FACILITY FUNCTION instruction 10-73
 PERFORM TOPOLOGY FUNCTION instruction 10-80
 permissible set 9-13
 PFD (PREFETCH DATA) instruction 7-285
 PFDRL (PREFETCH DATA RELATIVE LONG) instruction 7-285
 PFPO (PERFORM FLOATING-POINT OPERATION) floating-point-support instruction 9-34
 PFPO facility 1-18
 facility indication 4-68
 PFRA (page-frame real address) 3-44
 PGIN (PAGE IN) instruction 10-64
 PGOUT (PAGE OUT) instruction 10-65
 piecemeal steps of instruction execution 5-86
 PIM (path-installed mask) 15-6
 PKA (PACK ASCII) instruction 7-253
 PKM (PSW-key mask) 5-25
 PKU (PACK UNICODE) instruction 7-254
 PLO (PERFORM LOCKED OPERATION) instruction 7-270
 example A-51
 PMCW (path-management-control word) 15-2
 channel-path identifiers (CHPID) 15-8
 PNOM (path-not-operational mask) 15-5
 effect on POM of 15-11
 indicated in SCSW 16-13
 point of damage 11-13
 point of interruption 5-20
 for machine check 11-13
 POM (path-operational mask) 15-7
 effect on PNOM of 15-11
 POPCNT (POPULATION COUNT) instruction 7-284
 population-count facility 1-18
 facility indication 4-68
 POPULATION COUNT instruction 7-284
 postnormalization 18-2
 power controls 12-4
 power-on reset 4-55
 powers of 2
 table of G-1
 PR (PROGRAM CALL instruction 10-81
 PR (PROGRAM RETURN) instruction 10-94
 PR-cp (PROGRAM RETURN instruction, to current primary) 10-95
 PR-ss (PROGRAM RETURN instruction, with space switching) 10-95
 PR/SM (Processor Resource/Systems Manager) 1-24, 1-26
 precise intermediate value 9-12
 precision (floating-point) 9-2
 precision-rounded value 9-12
 preferred quantum 20-2
 preferred sign codes 8-2
 prefetch control 15-27
 PREFETCH DATA instruction 7-285
 PREFETCH DATA RELATIVE LONG instruction 7-285
 prefetching
 access exceptions not recognized for 6-40
 channel-control check during 16-29, 16-43
 channel-data check during 16-28, 16-43
 handling of invalid CBC in storage during 11-7
 of ART-table and DAT-table entries 5-90
 of data for I/O 15-53
 of instructions 5-89
 of operands 5-93
 See CCW prefetch control
 prefix 3-18
 set by signal-processor order 4-59
 store-status save area for 3-67
 prefix area 3-18
 prefix register 3-18
 prenormalization 18-2
 primary address space 3-19

- primary ASN (PASN) 3-20
 - in linkage-stack state entry 5-78
- primary-AST entry (PASTE) 5-32
 - origin (PASTE0) 5-32, 5-50
- primary ASTEIN (PASTEIN)
 - in linkage-stack state entry 5-79
- primary authority 3-32
 - exception 6-28
- primary designation-type-control bits 3-39
- primary interruption condition (I/O) 16-4
- primary-list bit 5-51
- primary private-space-control bit 3-38
- primary real-space-control bit 3-39
- primary real-space token origin (PRSTKO) 3-39
- primary region table
 - designation (PRTD) 3-38
 - length (PRTL) 3-39
 - origin (PRTO) 3-38
- primary segment table
 - designation (PSTD) 3-38
 - length (PSTL) 3-39
 - origin (PSTO) 3-38
- primary-space access-list designation (PSALD) 5-52
- primary-space mode 3-36
- primary space-switch-event-control bit 3-38
- primary-status bit (I/O) 16-19
- primary storage-alteration-event-control bit 3-38
- primary subspace-group-control bit 3-38
- primary virtual address 3-5
 - effective address-space-control element for 3-45
- priority
 - of access exceptions 6-46
 - of ASN-translation exceptions 6-49
 - of data exceptions 6-15
 - of external-interruption conditions 6-10
 - of I/O interruptions 16-5
 - of interruptions (CPU) 6-50
 - of PER events 4-29
 - of program-interruption conditions 6-44
 - of subspace-replacement exceptions 6-49
 - of trace exceptions 6-49
- private bit 5-53
- private-space control
 - effect on
 - fetch-protection override 3-12
 - low-address protection 3-13
 - use of common segments 3-43
- private-space-control bit 3-38
 - home 3-40
 - primary 3-38
 - secondary 3-40
- privileged instructions 4-5
 - control 10-1
 - I/O 14-1
- privileged-operation exception 6-29
- problem state 4-5
 - bit in entry-table entry 5-34
 - bit in the PSW 4-5
 - compatibility 1-27
- processing backup (synchronous machine-check condition) 11-17
- processing damage (synchronous machine-check condition) 11-17
- processor
 - See CPU 2-2
- processor-availability facility 1-24
- Processor Resource/Systems Manager (PR/SM) 1-24, 1-26
- program 5-45
 - channel
 - See channel program
 - exceptions 6-14
 - execution of 5-1
 - fields of SCHIB modifiable by 15-10
 - initial loading by CCW-type IPL 17-17
 - initial loading by list-directed IPL 17-19
 - initial loading of 4-56, 17-16
 - interruption 6-14
 - priority of 6-15, 6-44
- PROGRAM CALL instruction 10-81
 - trace entry for 4-20
 - type of 5-35
- program-call state entry 5-77, 10-85
- program check
 - as subchannel status 16-25, 16-40
 - measurement-block 16-48
- program-controlled interruption (I/O)
 - See PCI
- program-event recording
 - See PER
- program events
 - See PER events
- program mask
 - in the PSW 4-6, 6-14
 - inserted by INSERT PROGRAM MASK 7-213
 - preserved by BRANCH AND LINK 5-12, 7-34
 - set by SET PROGRAM MASK 7-296
 - validity bit for 11-19
- PROGRAM RETURN instruction 10-94
- program-status word
 - See PSW
- PROGRAM TRANSFER instruction 10-98
 - trace entry for 4-20
- PROGRAM TRANSFER WITH INSTANCE instruction 10-98
- programmable field of TOD clock 4-41
- protection (storage) 3-10
 - access-list-controlled

- See access-list-controlled protection
- DAT
 - See DAT protection
- during tracing 4-22
- fetch
 - See fetch protection
- key-controlled
 - See key-controlled protection
- low-address
 - See low-address protection
- page
 - See page protection
- protection check
 - as subchannel status 16-28, 16-42
 - measurement-block 16-48
- protection exception 6-30
 - as an access exception 6-40, 6-46
- PRSD (primary real-space designation) 3-38
- PRSTKO (primary real-space token origin) 3-39
- PRTD (primary region-table designation) 3-38
- PRTL (primary region-table length) 3-39
- PRTD (primary region-table origin) 3-38
- PSALD (primary-space access-list designation) 5-52
- pseudo AST entry 3-20
- PSMF(PERFORM FRAME MANAGEMENT FUNCTION) instruction 10-70
- PSTD (primary segment-table designation) 3-38
- PSTL (primary segment-table length) 3-39
- PSTO (primary segment-table origin) 3-38
- PSW (program-status word) 2-3, 4-3
 - current 4-3, 5-11
 - stored during interruption 6-2
 - exceptions associated with 6-8
 - format error 6-8
 - in linkage-stack state entry 5-78
 - in program execution 5-11
 - saved 4-53
 - store-status save area for 3-67
 - validity bits for 11-19
- PSW key 4-5
 - control bit 5-71
 - in entry-table entry 5-72
 - in trace entry 4-20
 - used as access key 3-10
 - validity bit for 11-19
- PSW-key mask (PKM) 5-25
 - control bit 5-71
 - in linkage-stack state entry 5-78
- PT (PROGRAM TRANSFER) instruction 10-98
- PT-cp (PROGRAM TRANSFER instruction, to current primary) 10-99
- PT-ss or PTI-ss (PROGRAM TRANSFER (WITH INSTANCE) instruction, with space switching) 10-99

- PTF (PERFORM TOPOLOGY FUNCTION)
 - instruction 10-80
- PTFF (PERFORM TIMING FACILITY FUNCTION)
 - instruction 10-73
- PTI (PROGRAM TRANSFER WITH INSTANCE)
 - instruction 10-98
- PTLB (PURGE TLB) instruction 10-107
- PTO (page-table origin) 3-42
- publications
 - other related documents xxvii
- PURGE ALB instruction 10-107
- PURGE TLB instruction 10-107
- PX (page index) 3-35

Q

- QADTR (QUANTIZE) DFP instruction 20-45
- QAXTR (QUANTIZE) DFP instruction 20-45
- QNaN (quiet NaN) 9-2, 19-4, 20-7
- quadword 3-3
- quadword-concurrent storage references 5-97
- QUANTIZE DFP instructions 20-45
- quantum 20-2
- queuing
 - FIFO
 - example for lock and unlock A-49
 - LIFO
 - example for lock and unlock A-48
- quiescing 5-102
- quiet NaN (QNaN) 9-2, 19-4, 20-7

R

- R field of instruction 5-6
- radix
 - binary 9-4
 - decimal 9-4
 - hexadecimal 9-3
- rate control 12-4
- RCHP (RESET CHANNEL PATH) instruction 14-9
- real address 3-4
- real mode 3-36
- real space
 - token origin (RSTKO) 3-38
- real-space-control bit
 - home 3-40
 - primary 3-39
 - secondary 3-40
- real-space designation (RSD)
 - home 3-40
 - primary 3-38
 - secondary 3-40
- real storage 3-5
- receiver check (signal-processor status) 4-65
- reconfiguration of I/O system 17-22

- recovery
 - as class of machine-check condition 11-10
 - channel-subsystem 17-24
 - system 11-14
 - subclass-mask bit for 11-23
- reduced-authority state 10-7
- redundancy 11-2
- reference
 - bit in storage key 3-9
 - multiple-access 5-96
 - recording 3-16
 - sequence for storage 5-85
 - See also* sequence
 - single-access 5-95
- region 3-35
- region first index (RFX) 3-35
- region-first-translation exception 6-30
 - as an access exception 6-46
- region index (RX)
 - in virtual address 3-35
- region-invalid bit (in region-table entry) 3-41
- region second index (RSX) 3-35
- region-second-translation exception 6-31
 - as an access exception 6-46
- region table
 - origin (RTO) 3-38
- region-table designation (RTD) 3-38
 - home 3-40
 - primary 3-38
 - secondary 3-39
- region-table entry (RTE) 3-41
- region third index (RTX) 3-35
- region-third-translation exception 6-31
 - as an access exception 6-46
- region-translation exception 6-30
- register
 - access 2-5
 - base-address 2-3
 - control 2-5
 - designation of 5-6
 - floating-point 2-3, 9-8
 - floating-point-control 9-9
 - general 2-3
 - index 2-3
 - prefix 3-18
 - save areas for 3-67, 11-21
 - validation of 11-8
 - validity bits for 11-20
- relative branching 5-10
- remainder 19-5
 - result of DIVIDE TO INTEGER 19-27
- remaining free space (in linkage stack) 5-75
- remote operating stations 12-1
- reporting-source code (RSC) 17-26
- reporting-source ID (RSID) 17-28
- repressible machine-check conditions 11-10
- request-logging-only 16-52
 - indicator for (in ERW) 16-52
- REROUND DFP instructions 20-47
- reset 17-12
 - channel-path 17-12
 - clear 4-54
 - CPU 4-53
 - effect on CPU state 4-2
 - effect on TOD clock 4-38
 - I/O-system 17-13
 - as part of subsystem reset 4-54
 - initial CPU 4-54
 - power on 4-55
 - subsystem 4-54
 - summary of functions 4-52
 - summary of functions performed by manual initiation of 4-51
 - system-reset-clear key 12-5
 - system-reset-normal key 12-5
- RESET CHANNEL PATH instruction 14-9
 - function initiated by 15-75
 - See* channel-path-reset function
- RESET REFERENCE BIT EXTENDED instruction 10-107
- RESET REFERENCE BIT MULTIPLE instruction 10-108
- reset-reference-bits-multiple facility 1-19
 - facility indication 4-68
- reset signal (I/O) 17-12
 - in I/O-system reset 17-13, 17-14
 - issued as part of RCHP 15-76
 - in channel-path reset 17-12
- resolution
 - of clock comparator 4-48
 - of CPU timer 4-49
 - of TOD clock 4-38
- restart
 - interruption 6-49
 - key 12-4
 - signal-processor order 4-59
- result operand 5-3
- resume function 13-11, 15-20
 - initiated by RESUME SUBCHANNEL 14-10
 - path management for 15-20
 - pending 16-14, 16-36
 - See also* start function
- RESUME PROGRAM instruction 10-108
- RESUME SUBCHANNEL instruction 14-10
 - channel-program requirements for 14-11
 - count of in measurement block 17-4
 - function initiated by 15-20
 - See* resume function

- retry
 - CPU 11-2
 - I/O command
 - See command retry
- reusable-ASN bit 3-22
- RFX (region first index) 3-35
- RI instruction format 5-5
- RIE instruction format 5-5
- right-units view (RUV) of floating-point numbers 9-2
- RIL instruction format 5-5
- RIS instruction format 5-5
- RISBG (ROTATE THEN INSERT SELECTED BITS) instruction 7-288
- RISBHG (ROTATE THEN INSERT SELECTED BITS HIGH) instruction 7-290
- RISBLG (ROTATE THEN INSERT SELECTED BITS LOW) instruction 7-290
- RLL (ROTATE LEFT SINGLE LOGICAL) instruction 7-286
- RLLG (ROTATE LEFT SINGLE LOGICAL) instruction 7-286
- RNSBG (ROTATE THEN AND SELECTED BITS) instruction 7-287
- ROTATE LEFT SINGLE LOGICAL instruction 7-286
- ROTATE THEN AND SELECTED BITS instruction 7-287
- ROTATE THEN EXCLUSIVE OR SELECTED BITS instruction 7-287
 - example A-30
- ROTATE THEN INSERT SELECTED BITS HIGH instruction 7-290
- ROTATE THEN INSERT SELECTED BITS instruction 7-288
 - example A-30
- ROTATE THEN INSERT SELECTED BITS LOW instruction 7-290
- ROTATE THEN OR SELECTED BITS instruction 7-287
 - example A-30
- rounded intermediate value 9-12
- rounding (decimal) 8-12
 - example A-40
- rounding (floating-point)
 - IEEE 9-13
- rounding action
 - summary of 9-17
- rounding methods 9-13
- RP (RESUME PROGRAM) instruction 10-108
- RR instruction format 5-5
- RRBE (RESET REFERENCE BIT EXTENDED) instruction 10-107
- RRBM (RESET REFERENCE BIT MULTIPLE) instruction 10-108
- RRD instruction format 5-5
- RRDTR (REROUND) DFP instruction 20-47
- RRE instruction format 5-5
- RRF instruction format 5-5
- RRS instruction format 5-5
- RRXTR (REROUND) DFP instruction 20-47
- RS instruction format 5-5
- RSC (reporting-source code) 17-26
- RSCH (RESUME SUBCHANNEL) instruction 14-10
- RSI instruction format 5-5
- RSID (reporting-source ID) 17-28
- RSL instruction format 5-5
- RSTKO (real-space token origin) 3-38
- RSX (region second index) 3-35
- RSY instruction format 5-5
- RTE (region-table entry) 3-41
- RTO (region-table origin) 3-38
- RTX (region third index) 3-35
- running (state of TOD clock) 4-38
- RUV (right-units view) of floating-point numbers 9-2
- RX (region index) 3-35
- RX instruction format 5-5
- RXE instruction format 5-5
- RXF instruction format 5-5
- RXSBG (ROTATE THEN EXCLUSIVE OR SELECTED BITS) instruction 7-287
- RXY instruction format 5-5

S

- S (SUBTRACT) binary instruction 7-311
- S instruction format 5-5
- SAC (SET ADDRESS SPACE CONTROL) instruction 10-110
- SACF (SET ADDRESS SPACE CONTROL FAST) instruction 10-111
- SAL (SET ADDRESS LIMIT) instruction 14-12
- SAM24 (SET ADDRESSING MODE) instruction 7-295
- SAM31 (SET ADDRESSING MODE) instruction 7-295
- SAM64 (SET ADDRESSING MODE) instruction 7-295
- sample count (in ESW) 17-4
- SAR (SET ACCESS) instruction 7-295
- SASCE (secondary address-space-control element) 3-39
- SASN (secondary address-space number) 3-20
 - in trace entry 4-21
- SASTEIN (secondary-AST-entry instance number) 3-22
- save areas for registers 3-67, 11-21
- saved PSW 4-53
- scale factor (Ψ) 9-12
- scaled preferred quantum 20-3

- scaled value 9-12
- scaling exponent
 - signed (Ω) 9-12
 - unsigned (α) 9-12
- SCHIB (subchannel-information block) 15-2
 - as operand of
 - MODIFY SUBCHANNEL 14-8
 - STORE SUBCHANNEL 14-18
 - Measurement-Block Address (MBA) 15-9
 - model-dependent area in 15-9
 - path-management-control word (PMCW) in 15-2
 - subchannel-status word (SCSW) in 15-9
 - summary of modifiable fields in 15-10
- SCHM (SET CHANNEL MONITOR) instruction 14-13
- SCK (SET CLOCK) instruction 10-112
- SCKC (SET CLOCK COMPARATOR) instruction 10-113
- SCKPF (SET CLOCK PROGRAMMABLE FIELD) instruction 10-113
- SCP-initiated reset 1-24
- SCSW (subchannel-status word) 16-7
 - activity-control field in 16-14, 16-36
 - CCW address in 16-19
 - count in 16-31, 16-44, 16-45
 - device-status field in 16-24
 - function-control field in 16-13
 - in IRB 16-7
 - in SCHIB 15-9
 - status-control field in 16-17
 - subchannel-control field in 16-12, 16-35
 - subchannel-status field in 16-24, 16-40
- SD (SUBTRACT NORMALIZED) HFP instruction 18-24
- SDB (SUBTRACT) BFP instruction 19-39
- SDBR (SUBTRACT) BFP instruction 19-39
- SDR (SUBTRACT NORMALIZED) HFP instruction 18-24
- SDTR (SUBTRACT) DFP instruction 20-51
- SDTRA (SUBTRACT) DFP instruction 20-51
- SE (SUBTRACT NORMALIZED) HFP instruction 18-24
- SEARCH STRING instruction 7-291
 - examples A-31
- SEARCH STRING UNICODE instruction 7-293
- SEB (SUBTRACT) BFP instruction 19-39
- SEBR (SUBTRACT) BFP instruction 19-39
- secondary address space 3-19
- secondary ASN (SASN) 3-20
 - control bit 5-72
 - in linkage-stack state entry 5-78
- secondary ASTEIN (SASTEIN)
 - in linkage-stack state entry 5-79
- secondary authority 3-32
 - exception 6-32
- secondary-CCW address validity (in ERW) 16-52
- secondary designation-type-control bits 3-40
- secondary error (in subchannel logout) 16-50
- secondary interruption condition (I/O) 16-5
- secondary private-space-control bit 3-40
- secondary real-space-control bit 3-40
- secondary real-space token origin (SRSTKO) 3-40
- secondary region table
 - designation (SRTD) 3-39
 - length (SRTL) 3-40
 - origin (SRTO) 3-40
- secondary segment table
 - designation (SSTD) 3-39
 - length (SSTL) 3-40
 - origin (SSTO) 3-40
- secondary-space-control bit 3-37, 5-25
- secondary-space mode 3-36
- secondary-status bit (I/O) 16-19
- secondary storage-alteration-event-control bit 3-40
- secondary subspace-group-control bit 3-40
- secondary virtual address 3-5
 - effective address-space-control element for 3-45
- segment 3-35
 - segment-frame absolute address 3-42
 - segment index (SX) 3-35
 - segment-invalid bit (in segment-table entry) 3-43
- segment table 3-42
 - length (STL) 3-38
 - origin (STO) 3-38
- segment-table designation (STD)
 - home 3-40
 - obtaining of in access-register translation 5-44
 - primary 3-38
 - secondary 3-39
- segment-translation exception 6-32
 - as an access exception 6-40, 6-46
- self-describing block of I/O data 15-57
- semiprivileged
 - instructions 4-5
 - descriptions of 10-2
 - program authorization 5-24
 - summary of 5-27
 - programs 4-5, 5-24
- sense
 - as signal-processor order 4-58
- sense-running-status facility
 - facility indication 4-67
- sequence
 - conceptual 5-85
 - instruction-execution 5-2
 - of CCWs that is invalid 16-27
 - of storage references 5-85
 - ART-table and DAT-table entries 5-90

- for floating-point data 9-8
 - instructions 5-89
 - operands 5-93
 - storage keys 5-91
- sequence code (in subchannel logout) 16-50
 - field-validity flag 16-49
- SER (SUBTRACT NORMALIZED) HFP instruction 18-24
- serial-I/O channel-to-channel adapter
 - publication referenced xxvii
- serial-I/O interface 13-3
 - publication referenced xxvii
- serialization 5-99
 - caused by I/O Instructions 14-2
 - channel-program 5-102
 - CPU 5-100
 - in completion of store operations 5-94
 - quiescing 5-102
 - specific operand 5-101
- server-time-protocol facility 1-19
- service-call-logical-processor (SCLP) facility 1-25
- service-processor damage 11-16
- service processor inoperative (signal-processor status) 4-65
- service-signal external interruption 6-12
 - subclass-mask bit for 6-12
- SET ACCESS instruction 7-295
- SET ADDRESS LIMIT instruction 14-12
- SET ADDRESS SPACE CONTROL FAST instruction 10-111
- SET ADDRESS SPACE CONTROL instruction 10-110
- SET ADDRESSING MODE instruction 7-295
- SET ADDRESSING MODE instructions 7-295
- set architecture
 - signal-processor order 4-60
- SET BFP ROUNDING MODE floating-point-support instructions 9-45
- SET CHANNEL MONITOR instruction 14-13
 - effect on measurement modes of 17-1
- SET CLOCK COMPARATOR instruction 10-113
- SET CLOCK instruction 10-112
- SET CLOCK PROGRAMMABLE FIELD instruction 10-113
- SET CPU TIMER instruction 10-114
- SET DFP ROUNDING MODE (SRNMT) floating-point-support instruction 9-45
- SET FPC AND SIGNAL floating-point-support instruction 9-46
- SET FPC floating-point-support instruction 9-46
- set prefix (signal-processor order) 4-59
- SET PREFIX instruction 10-114
- SET PROGRAM MASK instruction 7-296
- set-program-parameters facility
 - facility indication 4-68
- SET PSW KEY FROM ADDRESS instruction 10-115
- SET SECONDARY ASN instruction 10-115
 - access registers 5-48
- SET SECONDARY ASN WITH INSTANCE instruction 10-115
- set state (of TOD clock) 4-38
- SET STORAGE KEY EXTENDED instruction 10-121
- SET SYSTEM MASK instruction 10-124
- SFAA (segment-frame absolute address) 3-42
- SFASR (SET FPC AND SIGNAL) floating-point-support instruction 9-46
- SFPC (SET FPC) floating-point-support instruction 9-46
- SG (SUBTRACT) binary instruction 7-311
- SGF (SUBTRACT) binary instruction 7-311
- SGFR (SUBTRACT) binary instruction 7-311
- SGR (SUBTRACT) binary instruction 7-311
- SGRK (SUBTRACT) binary instruction 7-311
- SH (SUBTRACT HALFWORD) instruction 7-312
- shared storage
 - See storage sharing
- shared TOD clock 4-38
- SHHHR (SUBTRACT HIGH) instruction 7-312
- SHHLR (SUBTRACT HIGH) instruction 7-312
- SHIFT AND ROUND DECIMAL instruction 8-12
 - examples A-39
- SHIFT LEFT DOUBLE instruction 7-296
 - example A-31
- SHIFT LEFT DOUBLE LOGICAL instruction 7-297
- SHIFT LEFT SINGLE instruction 7-297
 - example A-32
- SHIFT LEFT SINGLE LOGICAL instruction 7-298
- SHIFT RIGHT DOUBLE instruction 7-299
- SHIFT RIGHT DOUBLE LOGICAL instruction 7-299
- SHIFT RIGHT SINGLE instruction 7-300
- SHIFT RIGHT SINGLE LOGICAL instruction 7-301
- SHIFT SIGNIFICAND LEFT DFP instructions 20-49
- SHIFT SIGNIFICAND RIGHT DFP instructions 20-50
- shifting
 - floating-point
 - See normalization
- short binary-floating-point number 19-2
- short decimal-floating-point number 20-3
- short hexadecimal-floating-point number 18-2
- short I/O block 16-24
- SHY (SUBTRACT HALFWORD) instruction 7-312
- SI instruction format 5-5
- SID
 - See subsystem-identification word
- sign bit
 - binary 7-5
 - floating point 9-2
- sign codes (decimal) 8-2

- signal (I/O) 17-11
 - clear
 - See clear signal
 - halt
 - See halt signal
 - reset
 - See reset signal
- SIGNAL PROCESSOR instruction 10-124
 - orders 4-58
 - status 4-63
- signaling NaN (SNaN) 9-2, 19-4, 20-7
- signed binary
 - arithmetic 7-6
 - comparison 7-8
 - integer 7-5
 - examples A-2
- signed exponent 9-2
- signed-packed-decimal format 8-1
 - used in DFP formatting instructions 20-8
- signed-packed-decimal numbers
 - examples A-4
- signed scaling exponent (Ω) 9-12
- significance
 - loss 18-2
 - in HFP addition 18-9
 - mask (in PSW) 4-6
 - requested in REROUND 20-47
 - See also significant digits
 - starter (in EDIT) 8-8
- significand
 - in BFP data 19-3
 - in DFP data 20-5
- significant digits 20-5
- SIGP
 - See SIGNAL PROCESSOR instruction
- SIGP (SIGNAL PROCESSOR) instruction 10-124
- SIL instruction format 5-5
- simulated IEEE-exception data exception 6-22
- single-access reference 5-95
- single-path mode 15-4, 15-23
- SIY instruction format 5-5
- size notation xxvi
- size of address
 - controlled by addressing mode 5-8
 - in CCW 15-32
- skip flag in CCW 15-32
 - effect on data transfer of 15-59
- SL (SUBTRACT LOGICAL) instruction 7-313
- SLA (SHIFT LEFT SINGLE) instruction 7-297
 - example A-32
- SLAG (SHIFT LEFT SINGLE) instruction 7-297
- SLAK (SHIFT LEFT SINGLE) instruction 7-297
- SLB (SUBTRACT LOGICAL WITH BORROW)
 - instruction 7-315
- SLBG (SUBTRACT LOGICAL WITH BORROW)
 - instruction 7-315
- SLBGR (SUBTRACT LOGICAL WITH BORROW)
 - instruction 7-314
- SLBR (SUBTRACT LOGICAL WITH BORROW)
 - instruction 7-314
- SLDA (SHIFT LEFT DOUBLE) instruction 7-296
 - example A-31
- SLDL (SHIFT LEFT DOUBLE LOGICAL) instruction 7-297
- SLDT (SHIFT SIGNIFICAND LEFT) DFP instruction 20-49
- SLFI (SUBTRACT LOGICAL IMMEDIATE)
 - instruction 7-313
- SLGFI (SUBTRACT LOGICAL IMMEDIATE)
 - instruction 7-313
- SLGFR (SUBTRACT LOGICAL) instruction 7-313
- SLGR (SUBTRACT LOGICAL) instruction 7-313
- SLGRK (SUBTRACT LOGICAL) binary instruction 7-313
- SLHHHR (SUBTRACT LOGICAL HIGH) instruction 7-314
- SLHHLR (SUBTRACT LOGICAL HIGH) instruction 7-314
- SLI (suppress-length-indication) flag in CCW 15-32
 - for immediate operations 15-54
- SLL (SHIFT LEFT SINGLE LOGICAL) instruction 7-298
- SLLG (SHIFT LEFT SINGLE LOGICAL) instruction 7-298
- SLLK (SHIFT LEFT SINGLE LOGICAL) instruction 7-298
- SLR (SUBTRACT LOGICAL) instruction 7-313
- SLRK (SUBTRACT LOGICAL) binary instruction 7-313
- SLXT (SHIFT SIGNIFICAND LEFT) DFP instruction 20-49
- SLY (SUBTRACT LOGICAL) instruction 7-313
- SNaN (signaling NaN) 9-2, 19-4, 20-7
- solicited interruption condition (I/O) 16-3
- solid errors 11-5
- sorting
 - extended 1-24
- sorting instructions
 - example A-53
 - See also COMPARE AND FORM CODEWORD instruction, UPDATE TREE instruction
- source of interruption
 - identified by interruption code 6-4
- SP (SUBTRACT DECIMAL) instruction 8-13
- space-switch event 6-32
 - control bit
 - in ASTE 3-29
- space-switch-event-control bit

- home 3-40
- primary 3-38
- special-operation exception 6-33
- special QNaN 9-3, 19-4, 20-7
- special quantum-handling operations 20-3
- specific-operand serialization 5-101
- specification exception 6-35
- SPKA (SET PSW KEY FROM ADDRESS) instruction 10-115
- SPM (SET PROGRAM MASK) instruction 7-296
- SPT (SET CPU TIMER) instruction 10-114
- SPX (SET PREFIX) instruction 10-114
- SQD (SQUARE ROOT) HFP instruction 18-23
- SQDB (SQUARE ROOT) BFP instruction 19-38
- SQDBR (SQUARE ROOT) BFP instruction 19-38
- SQDR (SQUARE ROOT) HFP instruction 18-23
- SQE (SQUARE ROOT) HFP instruction 18-23
- SQEB (SQUARE ROOT) BFP instruction 19-38
- SQEBR (SQUARE ROOT) BFP instruction 19-38
- SQER (SQUARE ROOT) HFP instruction 18-23
- SQUARE ROOT BFP instructions 19-38
- SQUARE ROOT HFP instructions 18-20
- SQXBR (SQUARE ROOT) BFP instruction 19-38
- SQXR (SQUARE ROOT) HFP instruction 18-23
- SR (SUBTRACT) binary instruction 7-311
- SRA (SHIFT RIGHT SINGLE) instruction 7-300
- SRAG (SHIFT RIGHT SINGLE) instruction 7-300
- SRAK (SHIFT RIGHT SINGLE) instruction 7-300
- SRDA (SHIFT RIGHT DOUBLE) instruction 7-299
- SRDL (SHIFT RIGHT DOUBLE LOGICAL) instruction 7-299
- SRDT (SHIFT SIGNIFICAND RIGHT) DFP instruction 20-50
- SRK (SUBTRACT) binary instruction 7-311
- SRL (SHIFT RIGHT SINGLE LOGICAL) instruction 7-301
- SRLG (SHIFT RIGHT SINGLE LOGICAL) instruction 7-301
- SRLK (SHIFT RIGHT SINGLE LOGICAL) instruction 7-301
- SRNM (SET BFP ROUNDING MODE) floating-point-support instruction 9-45
- SRNMB (SET BFP ROUNDING MODE) floating-point-support instruction 9-45
- SRNMT (SET DFP ROUNDING MODE) floating-point-support instruction 9-45
- SRP (SHIFT AND ROUND DECIMAL) instruction 8-12
 - examples A-39
- SRSD (secondary real-space designation) 3-40
- SRST (SEARCH STRING) instruction 7-291
 - examples A-31
- SRSTKO (secondary real-space token origin) 3-40
- SRSTU (SEARCH STRING UNICODE) instruction 7-293
- SRTD (secondary region-table designation) 3-39
- SRTL (secondary region-table length) 3-40
- SRTO (secondary region-table origin) 3-40
- SRXT (SHIFT SIGNIFICAND RIGHT) DFP instruction 20-50
- SS instruction format 5-5
- SSAIR (SET SECONDARY ASN WITH INSTANCE) instruction 10-115
- SSAR (SET SECONDARY ASN) instruction 10-115
 - access registers 5-48
- SSAR-cp or SSAIR-cp (SET SECONDARY ASN (WITH INSTANCE) instruction, to current primary) 10-116
- SSAR-ss or SSAIR-ss (SET SECONDARY ASN (WITH INSTANCE) instruction, with space switching) 10-116
- SSASTEO (subspace-AST-entry origin) 5-63
- SSASTESN (subspace-AST-entry sequence number) 5-63
- SSCH (START SUBCHANNEL) instruction 14-15
- SSE instruction format 5-5
- SSF instruction format 5-5
- SSID (subchannel-set identifier) 13-5
- SSKE (SET STORAGE KEY EXTENDED) instruction 10-121
- SSM (SET SYSTEM MASK) instruction 10-124
- SSM-suppression-control bit 6-33, 10-124
- SSTD (secondary segment-table designation) 3-39
- SSTL (secondary segment-table length) 3-40
- SSTO (secondary segment-table origin) 3-40
- ST (STORE) binary instruction 7-301
- stack-empty exception 6-38
- stack-full exception 6-38
- stack-operation exception 6-39
- stack-specification exception 6-39
- stack-type exception 6-39
- stacking process 5-80
- stacking PROGRAM CALL 5-67
- STAM (STORE ACCESS MULTIPLE) instruction 7-302
- STAMY (STORE ACCESS MULTIPLE) instruction 7-302
- standalone dump 12-5
- standard epoch (for TOD clock) 4-41
- STAP (STORE CPU ADDRESS) instruction 10-126
- start (CPU)
 - function 4-2
 - key 12-5
 - signal-processor order 4-58
- start function (I/O) 13-8, 15-20
 - bit in SCSW for 16-13
 - initiated by START SUBCHANNEL 14-15

- path management for 15-20
- pending 16-15, 16-36
- START SUBCHANNEL instruction 14-15
 - See also* start function for I/O
 - count of in measurement block 17-4
 - deferred condition code for (in SCSW) 16-9
 - function initiated by 15-20
 - operation-request block (ORB) used by 15-24
- state
 - CPU
 - See* CPU state
 - TOD-clock 4-38
- state entry 5-77
- status
 - alert 16-17
 - device 16-24
 - effect of clear function on 15-16
 - field-validity flag for (in subchannel logout) 16-49
 - with inappropriate bit combination 16-50
 - device-status check 16-50
 - for SIGNAL PROCESSOR 4-58, 10-125
 - initial-status interruption
 - See* initial-status-interruption control
 - intermediate 16-18, 16-36
 - primary 16-19
 - program
 - See* PSW
 - resulting from signal-processor orders 4-63
 - secondary 16-19
 - storing of 4-56
 - manual key for 12-5
 - subchannel 16-24
- status-control field (in SCSW) 16-17
- status modifier (device status)
 - effect of in command chaining 15-58
- status pending 16-19
- status-verification facility 17-22
- STC (STORE CHARACTER) instruction 7-302
- STCH (STORE CHARACTER HIGH) instruction 7-303
- STCK (STORE CLOCK) instruction 7-304
- STCKC (STORE CLOCK COMPARATOR) instruction 10-126
- STCKE (STORE CLOCK EXTENDED) instruction 7-305
- STCKF (STORE CLOCK FAST) instruction 7-304
- STCM (STORE CHARACTERS UNDER MASK) instruction 7-303
 - examples A-32
- STCMH (STORE CHARACTERS UNDER MASK) instruction 7-303
- STCMY (STORE CHARACTERS UNDER MASK) instruction 7-303

- STCPS (STORE CHANNEL PATH STATUS) instruction 14-17
- STCRW (STORE CHANNEL REPORT WORD) instruction 14-17
- STCTG (STORE CONTROL) instruction 10-126
- STCTL (STORE CONTROL) instruction 10-126
- STCY (STORE CHARACTER) instruction 7-302
- STD
 - See* segment-table designation
- STD (STORE) floating-point-support instruction 9-47
- STDY (STORE) floating-point-support instruction 9-47
- STE (STORE) floating-point-support instruction 9-47
- STEY (STORE) floating-point-support instruction 9-47
- STFH (STORE HIGH) instruction 7-308
- STFL (STORE FACILITY LIST) instruction 10-128
- STFL facility list 3-65
- STFLE (STORE FACILITY LIST EXTENDED) instruction 7-306
- STFPC (STORE FPC) floating-point-support instruction 9-47
- STG (STORE) binary instruction 7-301
- STGOC (STORE ON CONDITION) instruction 7-309
- STGRL (STORE RELATIVE LONG) instruction 7-301
- STH (STORE HALFWORD) instruction 7-307
- STHH (STORE HALFWORD HIGH) instruction 7-308
- STHHRL (STORE HALFWORD RELATIVE LONG) instruction 7-307
- STHY (STORE HALFWORD) instruction 7-307
- STIDP (STORE CPU ID) instruction 10-127
- STL (segment-table length) 3-38
- STM (STORE MULTIPLE) instruction 7-308
 - example A-32
- STMG (STORE MULTIPLE) instruction 7-308
- STMH (STORE MULTIPLE HIGH) instruction 7-309
- STMY (STORE MULTIPLE) instruction 7-308
- STNSM (STORE THEN AND SYSTEM MASK) instruction 10-149
- STO (segment-table origin) 3-38
- STOC (STORE ON CONDITION) instruction 7-309
- stop
 - function 4-2
 - key 12-5
 - signal-processor order 4-58
- stop and store status (signal-processor order) 4-59
- stopped (signal-processor status) 4-65
- stopped state
 - of CPU 4-2
 - effect on completion of store operations 5-94
 - of TOD clock 4-38
- storage 3-1, 3-38

- absolute 3-4
- address wraparound
 - See wraparound
- addressing 3-2
 - See also address
- alteration
 - space-control bit 4-24
- alteration manual controls 12-2
- alteration PER event 4-32
 - bits for 3-38
 - mask for 4-24
- assigned locations in 3-61
- auxiliary 3-2, 3-34
- block 3-4
 - testing for usability of 10-153
- buffer (cache) 3-2
- clearing of
 - See clearing operation
- concurrency of access for references to 5-97
- configuration of 3-4
- direct-access 3-2
- display 12-2
- error 11-18
 - indirect 11-19
- expanded 2-2
- failing address in
 - See failing-storage address
- interlocked update 5-93, 5-94
- interlocks for virtual references 5-86
- main 3-2
- noninterlocked update of 5-94
- nonvolatile 3-2
- operand 5-7
 - reference to (fetch, store, update) 5-93
 - update reference 5-94
- operand consistency 5-95
 - examples A-49, A-51
- prefixing for 3-18
- real 3-5
- sequence of references to 5-85
 - for floating-point data 9-8
- size
 - notation for xxvi
- validation of 11-6
- virtual 3-34
- volatile 3-2
 - effect of power-on reset on 4-55
- storage-access code (in subchannel logout) 16-50
- storage-alteration-event bit 4-24
- storage-alteration-event-control bit 3-38
 - home 3-40
 - primary 3-38
 - secondary 3-40
- storage-area designation
 - for PER events 4-30
- storage degradation (machine-check condition) 11-18
- storage key 3-8
 - error in 11-18
 - sequence of references to 5-91
 - testing for usability of 10-153
 - validation of 11-7
- storage-key function 1-24
- storage-logical-validity bit 11-20
- storage protection 3-10
 - during tracing 4-22
- storage-protection-override-control bit 3-11
- storage reconfiguration 1-24
- storage sharing
 - by address spaces 3-34
 - by CPUs and the channel subsystem 3-4
 - examples A-45
 - in multiprocessing 4-57
- STORE ACCESS MULTIPLE instructions 7-302
- STORE binary instructions 7-301
- STORE CHANNEL PATH STATUS instruction 14-17
- STORE CHANNEL REPORT WORD instruction 14-17
 - channel-report word (CRW) stored by 17-26
- STORE CHARACTER HIGH instruction 7-303, 7-308
- STORE CHARACTER instructions 7-302
- STORE CHARACTERS UNDER MASK instruction 7-303
 - examples A-32
- STORE CLOCK COMPARATOR instruction 10-126
- STORE CLOCK EXTENDED instruction 7-305
- store-clock-fast facility 1-19, 4-21
 - facility indication 4-67
- STORE CLOCK FAST instruction 7-304
- STORE CLOCK instruction 7-304
- STORE CONTROL instruction 10-126
- STORE CPU ADDRESS instruction 10-126
- STORE CPU ID instruction 10-127
- STORE CPU TIMER instruction 10-128
- store-facility-list-extended facility 1-19
 - facility indication 4-67
- STORE FACILITY LIST EXTENDED instruction 7-306
- STORE FACILITY LIST instruction 10-128
- STORE floating-point-support instructions 9-47
- STORE FPC floating-point-support instruction 9-47
- STORE HALFWORD HIGH instruction 7-308
- STORE HALFWORD instructions 7-307
- STORE HALFWORD RELATIVE LONG instruction 7-307
- STORE HIGH instruction 7-308
- STORE MULTIPLE HIGH instruction 7-309

- STORE MULTIPLE instruction
 - example A-32
- STORE MULTIPLE instructions 7-308
- STORE ON CONDITION instructions 7-309
- STORE PAIR TO QUADWORD instruction 7-310
- STORE PREFIX instruction 10-128
- STORE REAL ADDRESS instruction 10-129
- store reference 5-94
 - access exceptions for 6-43
- STORE RELATIVE LONG instructions 7-301
- STORE REVERSED instructions 7-310
- store status 4-56
 - key 12-5
 - signal-processor order for 4-59
- store-status architectural-mode identification 3-63
- store status at address (signal-processor order) 4-59
- STORE SUBCHANNEL instruction 14-18
- STORE SYSTEM INFORMATION instruction 10-130
- STORE THEN AND SYSTEM MASK instruction 10-149
- STORE THEN OR SYSTEM MASK instruction 10-149
- store using real address (PER event) 4-32
- store-using-real-address-event mask 4-24
- STORE USING REAL ADDRESS instruction 10-150
- STOSM (STORE THEN OR SYSTEM MASK) instruction 10-149
- STP clock source error (machine-check condition) 11-22
- STP sync check (machine-check condition) 11-21
- STP-timing-alert condition
 - See timing alert
- STPQ (STORE PAIR TO QUADWORD) instruction 7-310
- STPT (STORE CPU TIMER) instruction 10-128
- STPX (STORE PREFIX) instruction 10-128
- STRAG (STORE REAL ADDRESS) instruction 10-129
- streaming-mode control 15-25
- string of interruptions 4-3, 6-51
 - caused by clock comparator 4-49
 - caused by CPU timer 4-50
- STRV (STORE REVERSED) instruction 7-310
- STRVG (STORE REVERSED) instruction 7-311
- STRVH (STORE REVERSED) instruction 7-310
- STSCH (STORE SUBCHANNEL) instruction 14-18
- STSI (STORE SYSTEM INFORMATION) instruction 10-130
- STURA (STORE USING REAL ADDRESS) instruction 10-150
- STURG (STORE USING REAL ADDRESS) instruction 10-150
- STY (STORE) binary instruction 7-301
- SU (SUBTRACT UNNORMALIZED) HFP instruction 18-24
- subchannel
 - active allegiance for 15-13
 - dedicated allegiance for 15-13
 - effect of I/O-system reset on 17-14
 - idle 16-14
 - working allegiance for 15-12
- subchannel-active bit 16-16, 16-36
- subchannel addressing 13-5
- subchannel control information in SCSW 16-12, 16-35
- subchannel enabled bit in PMCW 15-3
- subchannel-information block
 - See SCHIB
- subchannel key 15-25, 15-30, 16-9
 - used as access key 3-10
 - used for IPL 17-17
- subchannel key check (in subchannel logout) 16-47
- subchannel logout 16-47
- subchannel number 13-5
- subchannel operation mode
 - command mode 13-9
 - transport mode 13-9
- subchannel-set identifier
 - See SSID
- Subchannel sets 13-2
- subchannel status 16-24, 16-40
- subchannel-status word
 - See SCSW
- subchannels 13-2
- subclass-mask bits
 - I/O-interruption
 - See I/O-interruption subclass mask
 - external-interruption 6-10
 - machine-check 11-23
- subnormal numbers
 - class of DFP data 20-6
- subroutine linkage 5-11
- subspace-active bit 5-63
- subspace-AST-entry origin (SSASTEO) 5-63
- subspace-AST-entry sequence number (SSASTESN) 5-63
- subspace-group control 3-38
- subspace-group-control bit
 - primary 3-38
 - secondary 3-40
- subspace groups
 - introduction to 5-14
- subspace-replacement
 - exceptions 6-49
 - operations 5-65
- subsystem-identification word 14-1
- subsystem-identification word (SID)

- assigned storage locations for 3-65
- subsystem-linkage-control bit 5-26, 5-32
- subsystem reset 4-54
- SUBTRACT BFP instructions 19-39
- SUBTRACT binary instructions 7-311
- SUBTRACT DECIMAL instruction 8-13
- SUBTRACT DFP instructions 20-51
- SUBTRACT HALFWORD instructions 7-312
- SUBTRACT HIGH instruction 7-312
- SUBTRACT LOGICAL HIGH instructions 7-314
- SUBTRACT LOGICAL IMMEDIATE instructions 7-313
- SUBTRACT LOGICAL instructions 7-313
- SUBTRACT LOGICAL WITH BORROW instructions 7-314
- SUBTRACT NORMALIZED
 - See* SUBTRACT BFP instructions
- SUBTRACT NORMALIZED HFP instructions 18-24
- SUBTRACT UNNORMALIZED HFP instructions 18-24
- successful-branching PER event 4-31
 - mask for 4-24
- SUPERVISOR CALL instruction 7-315
- supervisor-call interruption 6-50
- supervisor state 4-5
- support functions (I/O) 17-1
- suppress-length-indication flag in CCW
 - See* SLI
- suppress-suspended-interruption control (I/O) 15-27, 16-12
 - used for IPL 17-17
- suppression
 - exceptions to 5-23
 - of instruction execution 5-19
 - of unit of operation 5-21
- suppression on protection 3-14
- SUR (SUBTRACT UNNORMALIZED) HFP instruction 18-24
- suspend control 15-25
- suspend-control bit 16-9
 - used for IPL 17-17
- suspend flag in CCW 15-32
 - invalid 16-27
- suspend function 13-11
- suspended bit (in SCSW) 16-17, 16-36
- suspension of channel-program execution 15-68
 - effect on DCTI of 15-70
 - intermediate interruption condition for 16-18
- SVC (SUPERVISOR CALL) instruction 7-315
- SW (SUBTRACT UNNORMALIZED) HFP instruction 18-24
- swapping
 - by COMPARE (DOUBLE) AND SWAP instructions 7-122
 - by EXCLUSIVE OR instruction 7-206
- SWR (SUBTRACT UNNORMALIZED) HFP instruction 18-24
- SX (segment index) 3-35
- SXBR (SUBTRACT) BFP instruction 19-39
- SXR (SUBTRACT NORMALIZED) HFP instruction 18-24
- SXTR (SUBTRACT) DFP instruction 20-51
- SXTRA (SUBTRACT) DFP instruction 20-51
- SY (SUBTRACT) binary instruction 7-311
- symbol translation
 - use by VTAM 7-155
- symbol-translation-option bit 7-150
- synchronization
 - checkpoint 11-3
 - of CPU timer with TOD clock 4-49
 - of TOD clocks 4-38, 4-43
- synchronization control 15-26
- synchronous machine-check-interruption conditions 11-17
- system
 - manual control of 12-1
 - organization of 2-1
- system check stop 11-10
- system damage 11-14
- system mask (in PSW) 4-3
 - validity bit for 11-19
- system recovery 11-14
- system reset
 - I/O
 - See* I/O-system reset
 - See* reset
- system-reset-clear key 12-5
- system-reset-normal key 12-5
- System/360 and System/370 I/O interface
 - See* parallel-I/O interface

T

- T (tera) xxvi
- table of powers of 2 G-1
- table-type bits 3-41, 3-43
 - in region-table entry 3-41
 - in segment-table entry 3-43
- tables
 - ASN
 - See* ASN first table, ASN second table
 - authority
 - See* authority table
 - DAT
 - See* page table, segment table, region table
 - entry
 - See* entry table
 - linkage

- See linkage tables
- page
 - See page table
- region
 - See region table
- segment
 - See segment table
- translation 3-40
- TAI (International Atomic Time) related to UTC 4-41
- TAM (TEST ADDRESSING MODE) instruction 7-315
- TAR (TEST ACCESS) instruction 10-150
- target instruction 7-207
- target-precision-constrained rounding 9-12
- TB (TEST BLOCK) instruction 10-152
- TBDR (CONVERT HFP TO BFP) floating-point-support instruction 9-28
- TBDER (CONVERT HFP TO BFP) floating-point-support instruction 9-28
- TCCB (transport-command-control block)
 - address of 15-35
 - contents of 15-36
- TCDB (TEST DATA CLASS) BFP instruction 19-39
- TCEB (TEST DATA CLASS) BFP instruction 19-39
- DCW (device-command word) 15-37
- TCW (transport-command word) 15-33
 - contents of 15-33
- TCW channel program 13-7
- TCW (transport-command word)
 - chaining 15-58
- TCXB (TEST DATA CLASS) BFP instruction 19-39
- TDCDT (TEST DATA CLASS) DFP instruction 20-51
- TDCET (TEST DATA CLASS) DFP instruction 20-51
- TDCXT (TEST DATA CLASS) DFP instruction 20-51
- TDGDT (TEST DATA GROUP) DFP instruction 20-52
- TDGET (TEST DATA GROUP) DFP instruction 20-52
- TDGXT (TEST DATA GROUP) DFP instruction 20-52
- termination
 - of I/O operations
 - See conclusion of I/O operations
 - of instruction execution 5-20
 - for exigent machine-check conditions 11-10
 - of unit of operation 5-21
 - for exigent machine-check conditions 11-10
- termination code (in subchannel logout) 16-50
 - field-validity flag for 16-49
- TEST ACCESS instruction 10-150
- TEST ADDRESSING MODE instruction 7-315
- TEST AND SET instruction 7-316
- TEST BLOCK instruction 10-152
- TEST DATA CLASS BFP instructions 19-39
- TEST DATA CLASS DFP instructions 20-51
- TEST DATA GROUP DFP instructions 20-52
- TEST DECIMAL instruction 8-13
- test indicator 12-5
- TEST PENDING INTERRUPTION instruction 14-19
 - interruption code stored by 14-20
- TEST PROTECTION instruction 10-155
- TEST SUBCHANNEL instruction 14-20
 - interruption-response block (IRB) used by 16-6
- TEST UNDER MASK HIGH instruction 7-316
- TEST UNDER MASK instruction 7-316
 - examples A-33
- TEST UNDER MASK LOW instruction 7-316
- testing for storage-block and storage-key usability 10-153
- THDER (CONVERT BFP TO HFP) floating-point-support instruction 9-27
- THDR (CONVERT BFP TO HFP) floating-point-support instruction 9-27
- TIC (transfer in channel) 15-72
 - invalid sequence of 16-27
- ties 9-13
- time-of-day clock
 - See TOD clock
- timer
 - See CPU timer
- timing
 - channel-subsystem 17-2
- timing alert
 - external interruption 6-12
- timing facilities 4-38
- timing-facility bit (in PMCW) 15-4
- timing-facility damage 11-15
 - for TOD clock 4-39
- timing-alert condition
 - ETR-timing-alert condition 6-12
 - STP-timing-alert condition 6-12
- TLB (translation-lookaside buffer) 3-52
 - entries 3-53
 - attachment of 3-53
 - clearing of 3-56
 - effect of translation changes on 3-55
 - usable state 3-54
- TM (TEST UNDER MASK) instruction 7-316
 - examples A-33
- TMH (TEST UNDER MASK HIGH) instruction 7-316
- TMHH (TEST UNDER MASK) instruction 7-316
- TMHL (TEST UNDER MASK) instruction 7-316
- TML (TEST UNDER MASK LOW) instruction 7-316
- TMLH (TEST UNDER MASK) instruction 7-316
- TMLL (TEST UNDER MASK) instruction 7-316
- TMY (TEST UNDER MASK) instruction 7-316
- TOD clock 4-38
 - effect of power-on reset on 4-55
 - effect on clock-comparator interruption 6-10

- effect on CPU-timer decrementing 4-50
- effect on CPU-timer interruption 6-11
- manual control of 4-38, 12-6
- steering 4-44
 - facility indication 4-67
 - overview 4-45
 - registers 4-47
- unique values of 4-40
- validation of 11-8
- value in trace entry 4-21
- TOD-clock-control-override control 4-38
- TOD-clock programmable field 4-41
- TOD-clock programmable register 4-41
 - save areas for 3-67
 - validity bit for 11-20
- TOD-clock-steering facility 1-19
- TOD-clock-sync-control bit 4-38, 4-43
- TOD-clock-synchronization facility 4-43
- TP (TEST DECIMAL) instruction 8-13
- TPI (TEST PENDING INTERRUPTION) instruction 14-19
 - interruption code stored by 14-20
- TPROT (TEST PROTECTION) instruction 10-155
- TR (TRANSLATE) instruction 7-317
 - example A-33
- trace 4-10
 - entries 4-12
 - entry address 4-12
 - exceptions 6-49
 - table exception 6-39
- TRACE instruction 10-157
 - trace entry for 4-21
- TRACG (TRACE) instruction 10-157
- trailer entry 5-76
- transfer in channel
 - See* TIC
- transferring program control 5-67
- TRANSLATE AND TEST instruction 7-318
 - example A-34
- TRANSLATE AND TEST REVERSE instruction 7-323
- TRANSLATE EXTENDED instruction 7-324
- TRANSLATE instruction 7-317
 - example A-33
- TRANSLATE ONE TO ONE instruction 7-327
- TRANSLATE ONE TO TWO instruction 7-327
- TRANSLATE TWO TO ONE instruction 7-327
- TRANSLATE TWO TO TWO instruction 7-327
- translation
 - address 3-34
 - See* dynamic address translation
 - exception identification 3-63
 - lookaside buffer
 - See* TLB
 - PC-number 5-30
 - specification exception 6-40
 - tables for 3-40
 - translation modes 3-36
 - translation parameters 3-36
 - translation path 3-53
 - TCCB (transport-command-control block) 15-36
 - transport-command-control block
 - See* TCCB
 - transport-command word
 - See* TCW
 - transport-status block
 - See* TSB
 - trap action
 - IEEE 9-17
 - trap control block 10-159
 - TRAP instruction 10-159
 - trap save area 10-160
 - TRAP2 (TRAP) instruction 10-159
 - TRAP4 (TRAP) instruction 10-159
 - TRE (TRANSLATE EXTENDED) instruction 7-324
 - tree structure for sorting 7-334
 - example A-53
 - trial execution
 - for editing instructions and TRANSLATE instruction 5-24
 - for PER 4-26
 - trimodal addressing 5-8, 7-9
 - TROO (TRANSLATE ONE TO ONE) instruction 7-327
 - TROT (TRANSLATE ONE TO TWO) instruction 7-327
 - TRT (TRANSLATE AND TEST) instruction 7-318
 - example A-34
 - TRTO (TRANSLATE TWO TO ONE) instruction 7-327
 - TRTR (TRANSLATE AND TEST REVERSE) instruction 7-323
 - TRTT (TRANSLATE TWO TO TWO) instruction 7-327
 - true zero (HFP number) 18-1
 - TS (TEST AND SET) instruction 7-316
 - TSB (transport-status block) 15-43
 - address of 15-35
 - contents of 15-44, 15-50
 - TSCH (TEST SUBCHANNEL) instruction 14-20
 - two's complement binary notation 7-5
 - examples A-2
 - type of PROGRAM CALL 5-35

U

- ulp (unit in the last place) 9-7, 9-8, 19-3
- underflow

- See exponent underflow
- unit check (device status)
 - in establishing dedicated allegiance 15-13
- unit of operation 5-20
- unlock A-47
 - example with FIFO queuing A-50
 - example with LIFO queuing A-49
- unnormalized floating-point number 18-2
 - HFP data only 9-3
- unordered (comparison to a NaN) 9-24
- UNPACK ASCII instruction 7-332
- UNPACK instruction 7-331
 - example A-35
- UNPACK UNICODE instruction 7-333
- UNPK (UNPACK) instruction 7-331
 - example A-35
- UNPKA (PACK ASCII) instruction 7-332
- UNPKU (PACK UNICODE) instruction 7-333
- unprivileged instructions 4-5, 7-5
- unsigned binary
 - arithmetic 7-7
 - integer 7-5
 - examples A-3
 - in address generation 5-9
- unsigned binary integer
 - conversion to floating point 19-20
- unsigned-packed-decimal format 8-2
 - used in DFP formatting instructions 20-8
- unsigned scaling exponent (α) 9-12
- unsolicited interruption condition (I/O) 16-3
- unstack-suppression bit 5-75
- unstacking process 5-82
- update reference 5-94
- UPDATE TREE instruction 7-334
 - example A-53
- UPT (UPDATE TREE) instruction 7-334
 - example A-53
- usable ALB entry 5-61
- UTC (Coordinated Universal Time) used in TOD
 - epoch 4-41

V

- valid ART-table entry 5-61
- valid CBC 11-2
- valid floating-point-register numbers 9-9
- valid region-table, segment-table, or page-table entry 3-53
- validation 11-5
 - of registers 11-8
 - of storage 11-6
 - of storage key 11-7
 - of TOD clock 11-8
- validity bit for backward stack-entry address 5-76

- validity bit for forward-section-header address 5-76
- validity bits
 - in machine-check-interruption code 11-19
 - in subchannel logout 16-49
- variable-length field 3-3
- version code 10-127
- view (of floating-point numbers) 9-2
 - common-rounding-point 9-13
 - fraction 9-2
 - left-units view 9-2
 - right-units view 9-2
- virtual address 3-5
- virtual machine
 - extensions for 1-24
- virtual storage 3-34
- VM-data-space facility 1-24
- volatile storage 3-2
 - effect of power-on reset on 4-55
- voting digit 9-13
- VTAM use of symbol translation 7-155

W

- wait indicator 12-6
- wait-state bit
 - in the PSW 4-5
- warning (machine-check condition) 11-16
 - subclass-mask bit for 11-24
- word 3-3
- word-concurrent storage references 5-97
- working allegiance (I/O) 15-12
- wraparound
 - of instruction addresses 5-8
 - of PER addresses 4-30
 - of register numbers
 - for LOAD MULTIPLE instruction 7-226
 - for STORE MULTIPLE instruction 7-309
 - of storage addresses 3-7
 - controlled by addressing mode 3-6
 - for MOVE INVERSE instruction 7-233
 - for MOVE LONG EXTENDED instruction 7-238
 - for MOVE LONG instruction 7-234
 - for MOVE LONG UNICODE instruction 7-242
 - for MOVE WITH OPTIONAL SPECIFICATIONS instruction 10-61
 - of TOD clock 4-38

X

- X (EXCLUSIVE OR) instruction 7-205
- X field of instruction 5-9
- XA (extended architecture)
 - See 370-XA architecture
- XC (EXCLUSIVE OR) instruction 7-206

- examples A-20
- XG (EXCLUSIVE OR) instruction 7-205
- XGR (EXCLUSIVE OR) instruction 7-205
- XI (EXCLUSIVE OR) instruction 7-205
 - example A-21
- XIHF (EXCLUSIVE OR IMMEDIATE) instruction 7-206
- XILF (EXCLUSIVE OR IMMEDIATE) instruction 7-206
- XIY (EXCLUSIVE OR) instruction 7-205
- XR (EXCLUSIVE OR) instruction 7-205
- XRK (EXCLUSIVE OR) binary instruction 7-205
- XSCH (CANCEL SUBCHANNEL) instruction 14-4
- XY (EXCLUSIVE OR) instruction 7-205

Z

- Z bit (zero condition-code bit) 16-12
 - as cause of intermediate interruption condition 16-18
- z/Architecture architecture

- additions to 1-2, 1-7
- original highlights of 1-2
- ZAP (ZERO AND ADD) instruction 8-13
 - example A-41
- zero
 - instruction-length code 6-6
 - negative
 - See negative zero
 - normal meaning for byte value xxvii
 - setting floating-point register to 9-34
 - true (HFP number) 18-1
- ZERO AND ADD instruction 8-13
 - example A-41
- zero condition code (Z bit in SCSW) 16-12
- zero-padding control 7-150
- zone bits 8-1
 - moving of 7-246
- zoned decimal numbers 8-1
- zoned-decimal numbers
 - examples A-4

Communicating Your Comments to IBM

z/Architecture Principles of Operation: z/Architecture Principles of Operation

Publication No: SA22-7832-08

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a reader's comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
 - FAX: (International Access Code)+1+845+432-9405
- If you prefer to send comments electronically, use one of these network IDs:
 - Internet e-mail: mhvrcfs@us.ibm.com
 - World Wide Web: <http://www.ibm.com/servers/eserver/zseries/zos/webqs.html>

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

Reader's Comments — We'd Like to Hear from You

Title: z/Architecture Principles of Operation

Publication No: SA22-7832-08

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: _____

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

☐ As an introduction

☐ As a text (student)

☐ As a reference manual

☐ As a text (instructor)

☐ For another purpose (explain)

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number:

Comment:

Name

Address

Company or Organization

Phone No.

Cut or Fold
Along Line

Reader's Comments — We'd Like to Hear from You
SA22-7832-08



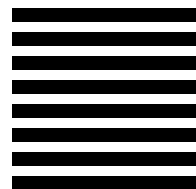
Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400



Fold and Tape

Please do not staple

Fold and Tape

SA22-7832-08

Cut or Fold
Along Line



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SA22-7832-08

