

GA22-7070-0  
File No. 4300-01

**Systems**

**IBM 4300 Processors  
Principles of Operation  
for ECPS:VSE Mode**

**IBM**

GA22-7070-0  
File No. 4300-01

**Systems**

**IBM 4300 Processors  
Principles of Operation  
for ECPS:VSE Mode**



**First Edition (January 1979)**

Changes are continually made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Product Publications, Department B98, PO Box 390, Poughkeepsie, NY, U.S.A. 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

## Preface

This publication provides, for reference purposes, a detailed definition of the architecture of IBM 4300 Processors when operating in the Extended Control Program Support: Virtual Storage Extended (ECPS:VSE) mode.

The publication describes each function of the architecture to the level of detail that must be understood in order 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.

The information in this publication is provided principally for use by assembler-language programmers, although anyone concerned with the functional details of the IBM 4300 Processors will find it useful.

This publication is written as a reference document and should not be considered an introduction or a textbook.

All facilities discussed in this publication are not necessarily available on every processor. Furthermore, in some instances the definitions have been structured to allow for some degree of extensibility, and therefore certain capabilities may be described or implied that are not offered on any model. An example of such capabilities is the provision for the number of channel-mask bits in the control register. The allowance for this type of extensibility should not be construed as implying any intention by IBM to provide such capabilities. For information about the characteristics and availability of features on a specific processor, use the functional characteristics manual for that processor. The availability of features on processors is summarized in the *IBM*

*4300 Processors Summary and Input/Output & Data Communications Configurator*, GA33-1523.

This publication applies only to the ECPS:VSE mode of operation. The *IBM System/370 Principles of Operation*, GA22-7000, should be consulted regarding the functions of the architecture which apply when the processor operates in the System/370 mode.

### **Size Notation**

The letters K, M, and G denote the multipliers  $2^{10}$ ,  $2^{20}$ , and  $2^{30}$ , respectively. It should be noted that although the letters are borrowed from the decimal system and stand for kilo ( $10^3$ ), mega ( $10^6$ ), and giga ( $10^9$ ), 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:

<i>Symbol</i>	<i>Value</i>
K (kilo)	$1,024 = 2^{10}$
M (mega)	$1,048,576 = 2^{20}$
G (giga)	$1,073,741,824 = 2^{30}$

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

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.

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

**This page left blank intentionally.**

# Contents

<b>Chapter 1. Introduction</b>	1-1
The IBM 4300 Processors	1-1
Compatibility	1-2
Compatibility Among 4300 Processors	1-2
Compatibility Between 4300 Processors and System/370	1-2
Control-Program Compatibility	1-2
Problem-State Compatibility	1-2
<b>Chapter 2. Organization</b>	2-1
Main Storage	2-1
Central Processing Unit	2-1
General Registers	2-2
Floating-Point Registers	2-2
Control Registers	2-3
Input and Output	2-3
Channels	2-4
Input/Output Devices and Control Units	2-4
Operator Facilities	2-4
<b>Chapter 3. Storage</b>	3-1
Information Formats and Addressing	3-1
Information Formats	3-1
Storage Addressing	3-2
Integral Boundaries	3-2
One-Level Addressing	3-2
Storage Size	3-3
Pages	3-3
Page Frames	3-3
Page Description	3-3
Storage Key	3-3
Page Bits	3-4
Page States	3-4
Frame Index	3-5
Page and Frame Control	3-5
Capacity Counts	3-5
Storage-Control Instructions	3-6
Key-Controlled Protection	3-6
Protection Action	3-6
Accesses Protected	3-7
Reference and Change Recording	3-7
Assigned Storage Locations	3-8
Storage While CPU Is in Operating State	3-8
Storage While CPU Is in Load State	3-10
<b>Chapter 4. Control</b>	4-1
CPU States	4-1
Wait State	4-2
Supervisor and Problem States	4-2
Stopped, Operating, Load, and Check-Stop States	4-2
EC and BC Modes	4-3
Program-Status Word	4-4
Program-Status-Word Format in EC Mode	4-4
Program-Status-Word Format in BC Mode	4-6
Control Registers	4-7
Monitoring	4-8
Program-Event Recording	4-9
Control-Register Allocation	4-9
Operation	4-10
Identification of Cause	4-10
Priority of Indication	4-11
Storage-Area Designation	4-11
PER Events	4-12
Successful Branching	4-12
Instruction Fetching	4-12
Storage Alteration	4-12
General-Register Alteration	4-12
Indication of Events Concurrently with Other Interruption Conditions	4-13
Timing	4-15
Time-of-Day Clock	4-15
Format	4-15
States	4-16
Setting and Inspecting the Value	4-16
Clock Comparator	4-18
CPU Timer	4-18
Interval Timer	4-19
Externally Initiated Functions	4-20
Resets	4-20
Program Reset	4-21
Initial Program Reset	4-23
Clear Reset	4-23
Power-On Reset	4-23
Initial Program Loading	4-24
Machine Save	4-25
<b>Chapter 5. Program Execution</b>	5-1
Instructions	5-1
Operands	5-1
Instruction Format	5-2
Register Operands	5-3
Immediate Operands	5-3
Storage Operands	5-3
Operand-Address Generation	5-4
Instruction Execution	5-4
Sequential Instruction Execution	5-4
Branching	5-4
Interruptions	5-5
Sequence of Storage References	5-5
Instruction Fetch	5-6
Page-Description Accesses	5-6
Storage-Operand References	5-7
Storage-Operand Fetch References	5-7
Storage-Operand Store References	5-7
Storage-Operand Update References	5-7
Storage-Operand Consistency	5-8

Relation Between Operand Accesses	5-8	Input/Output Interruption	6-20
Other Storage References	5-9	Restart Interruption	6-21
Serialization	5-9	Priority of Interruptions	6-21
CPU Serialization	5-9		
Channel Serialization	5-10		
<b>Chapter 6. Interruptions</b>	6-1	<b>Chapter 7. General Instructions</b>	7-1
Interruption Action	6-1	Data Format	7-1
Source Identification	6-2	Binary-Integer Representation	7-2
Enabling and Disabling	6-4	Signed and Unsigned Binary Arithmetic	7-3
Instruction-Length Code	6-4	Signed and Logical Comparison	7-3
Zero ILC	6-5	Instructions	7-4
ILC on Instruction-Fetch Exceptions	6-5	ADD	7-7
Exceptions Associated with the PSW	6-6	ADD HALFWORD	7-7
Early Exception Recognition	6-6	ADD LOGICAL	7-7
Late Exception Recognition	6-6	AND	7-7
Types of Instruction Ending	6-6	BRANCH AND LINK	7-8
Interruptible Instructions	6-7	BRANCH ON CONDITION	7-9
Point of Interruption	6-7	BRANCH ON COUNT	7-10
Ending of Interruptible Instructions	6-7	BRANCH ON INDEX HIGH	7-10
Machine-Check Interruption	6-8	BRANCH ON INDEX LOW OR EQUAL	7-10
Supervisor-Call Interruption	6-8	COMPARE	7-11
Program Interruption	6-8	COMPARE AND SWAP	7-11
Program-Interruption Conditions	6-9	COMPARE DOUBLE AND SWAP	7-11
Addressing Exception	6-9	COMPARE HALFWORD	7-13
Data Exception	6-9	COMPARE LOGICAL	7-13
Decimal-Divide Exception	6-10	COMPARE LOGICAL CHARACTERS	
Decimal-Overflow Exception	6-10	UNDER MASK	7-13
Execute Exception	6-10	COMPARE LOGICAL LONG	7-14
Exponent-Overflow Exception	6-10	CONVERT TO BINARY	7-15
Exponent-Underflow Exception	6-10	CONVERT TO DECIMAL	7-16
Fixed-Point-Divide Exception	6-10	DIVIDE	7-16
Fixed-Point-Overflow Exception	6-11	EXCLUSIVE OR	7-17
Floating-Point-Divide Exception	6-11	EXECUTE	7-17
Monitor Event	6-11	INSERT CHARACTER	7-18
Operation Exception	6-11	INSERT CHARACTERS UNDER MASK	7-18
Page-Access Exception	6-11	LOAD	7-19
Page-State Exception	6-12	LOAD ADDRESS	7-19
Page-Transition Exception	6-12	LOAD AND TEST	7-19
PER Event	6-12	LOAD COMPLEMENT	7-20
Privileged-Operation Exception	6-12	LOAD HALFWORD	7-20
Protection Exception	6-12	LOAD MULTIPLE	7-20
Significance Exception	6-13	LOAD NEGATIVE	7-21
Special-Operation Exception	6-13	LOAD POSITIVE	7-21
Specification Exception	6-13	MONITOR CALL	7-21
Recognition of Access Exceptions	6-13	MOVE	7-22
Nontransparent Nullification	6-16	MOVE INVERSE	7-22
Multiple Program-Interruption Conditions	6-17	MOVE LONG	7-23
External Interruption	6-19	MOVE NUMERICS	7-25
Clock Comparator	6-19	MOVE WITH OFFSET	7-25
CPU Timer	6-19	MOVE ZONES	7-26
External Signal	6-20	MULTIPLY	7-26
Interrupt Key	6-20	MULTIPLY HALFWORD	7-27
Interval Timer	6-20	OR	7-27
		PACK	7-28
		SET PROGRAM MASK	7-28
		SHIFT LEFT DOUBLE	7-29

SHIFT LEFT DOUBLE LOGICAL 7-29  
 SHIFT LEFT SINGLE 7-30  
 SHIFT LEFT SINGLE LOGICAL 7-30  
 SHIFT RIGHT DOUBLE 7-30  
 SHIFT RIGHT DOUBLE LOGICAL 7-31  
 SHIFT RIGHT SINGLE 7-31  
 SHIFT RIGHT SINGLE LOGICAL 7-32  
 STORE 7-32  
 STORE CHARACTER 7-32  
 STORE CHARACTERS UNDER MASK 7-32  
 STORE CLOCK 7-33  
 STORE HALFWORD 7-33  
 STORE MULTIPLE 7-33  
 SUBTRACT 7-34  
 SUBTRACT HALFWORD 7-34  
 SUBTRACT LOGICAL 7-34  
 SUPERVISOR CALL 7-35  
 TEST AND SET 7-35  
 TEST UNDER MASK 7-36  
 TRANSLATE 7-36  
 TRANSLATE AND TEST 7-37  
 UNPACK 7-38

**Chapter 8. Decimal Instructions 8-1**

Data Format 8-1  
     Zoned Format 8-1  
     Packed Format 8-1  
 Number Representation 8-2  
 Instructions 8-2  
     ADD DECIMAL 8-3  
     COMPARE DECIMAL 8-4  
     DIVIDE DECIMAL 8-4  
     EDIT 8-5  
     EDIT AND MARK 8-8  
     MULTIPLY DECIMAL 8-8  
     SHIFT AND ROUND DECIMAL 8-8  
     SUBTRACT DECIMAL 8-10  
     ZERO AND ADD 8-10

**Chapter 9. Floating-Point Instructions 9-1**

Data Format 9-1  
 Guard Digit 9-2  
 Number Representation 9-3  
 Normalization 9-3  
 Instructions 9-4  
     ADD NORMALIZED 9-6  
     ADD UNNORMALIZED 9-7  
     COMPARE 9-8  
     DIVIDE 9-8  
     HALVE 9-9  
     LOAD 9-10  
     LOAD AND TEST 9-10  
     LOAD COMPLEMENT 9-11  
     LOAD NEGATIVE 9-11  
     LOAD POSITIVE 9-11

LOAD ROUNDED 9-12  
 MULTIPLY 9-12  
 STORE 9-14  
 SUBTRACT NORMALIZED 9-14  
 SUBTRACT UNNORMALIZED 9-15

**Chapter 10. Control Instructions 10-1**

CLEAR PAGE 10-3  
 CONNECT PAGE 10-3  
 DECONFIGURE PAGE 10-4  
 DIAGNOSE 10-4  
 DISCONNECT PAGE 10-5  
 INSERT PAGE BITS 10-5  
 INSERT PSW KEY 10-5  
 INSERT STORAGE KEY 10-6  
 LOAD CONTROL 10-6  
 LOAD FRAME INDEX 10-6  
 LOAD PSW 10-7  
 MAKE ADDRESSABLE 10-7  
 MAKE UNADDRESSABLE 10-8  
 RESET REFERENCE BIT 10-8  
 RETRIEVE STATUS AND PAGE 10-8  
 SET CLOCK 10-9  
 SET CLOCK COMPARATOR 10-9  
 SET CPU TIMER 10-10  
 SET PAGE BITS 10-10  
 SET PSW KEY FROM ADDRESS 10-10  
 SET STORAGE KEY 10-11  
 SET SYSTEM MASK 10-11  
 STORE CAPACITY COUNTS 10-11  
 STORE CLOCK COMPARATOR 10-12  
 STORE CONTROL 10-12  
 STORE CPU ID 10-12  
 STORE CPU TIMER 10-13  
 STORE THEN AND SYSTEM MASK 10-13  
 STORE THEN OR SYSTEM MASK 10-14

**Chapter 11. Machine-Check Handling 11-1**

Machine-Check Detection 11-1  
 Correction of Machine Malfunctions 11-1  
 Handling of Machine Checks 11-2  
     Invalid CBC in Storage 11-2  
     Invalid CBC in Page Descriptions 11-2  
     Invalid CBC in Registers 11-3  
     Usage of Validation 11-3  
 Check-Stop State 11-3  
 Machine-Check Interruption 11-4  
     Exigent Conditions 11-4  
     Repressible Conditions 11-4  
     Interruption Action 11-4  
     Point of Interruption 11-5  
 Machine-Check-Interruption Code 11-6  
     Subclass 11-7  
     Auxiliary Bits 11-8  
     Machine-Check Interruption-Code Validity Bits 11-8

Machine-Check Extended Interruption Information	11-9	Conclusion at Operation Initiation	12-43
Register-Save Areas	11-9	Immediate Operations	12-44
Failing-Storage Address	11-9	Conclusion of Data Transfer	12-44
Machine-Check Masking	11-9	Termination by HALT I/O or HALT DEVICE	12-45
<b>Chapter 12. Input/Output Operations</b>	12-1	Termination by CLEAR I/O	12-46
Attachment of Input/Output Devices	12-2	Termination Due to Equipment Malfunction	12-47
Input/Output Devices	12-2	Input/Output Interruptions	12-47
Control Units	12-2	Interruption Conditions	12-47
Channels	12-3	Channel-Available Interruption	12-48
Modes of Operation	12-3	Priority of Interruptions	12-48
Types of Channels	12-4	Interruption Action	12-49
I/O-System Operation	12-5	Channel-Status Word	12-49
Compatibility of Operation	12-7	Unit Status	12-50
Control of Input/Output Devices	12-7	Attention	12-50
Input/Output Device Addressing	12-7	Status Modifier	12-50
States of the Input/Output System	12-8	Control-Unit End	12-51
Resetting of the Input/Output System	12-11	Busy	12-51
I/O System Reset	12-11	Channel End	12-53
I/O Selective Reset	12-11	Device End	12-53
Effect of Reset on a Working Device	12-11	Unit Check	12-53
Reset Upon Malfunction	12-12	Unit Exception	12-54
Condition Code	12-12	Channel Status	12-55
Instruction Formats	12-14	Program-Controlled Interruption	12-55
Instructions	12-15	Incorrect Length	12-55
CLEAR I/O	12-16	Program Check	12-55
HALT DEVICE	12-18	Protection Check	12-56
HALT I/O	12-21	Channel-Data Check	12-56
START I/O	12-23	Channel-Control Check	12-56
START I/O FAST RELEASE	12-23	Interface-Control Check	12-57
STORE CHANNEL ID	12-26	Chaining Check	12-57
TEST CHANNEL	12-27	Contents Of Channel-Status Word	12-57
TEST I/O	12-27	Information Provided by	
Input/Output-Instruction-Exception Handling	12-30	Channel-Status Word	12-57
Execution of Input/Output Operations	12-30	Subchannel Key	12-58
Blocking of Data	12-30	CCW Address	12-58
Channel-Address Word	12-30	Count	12-59
Channel-Command Word	12-31	Status	12-60
Command Code	12-32	Channel Logout	12-63
Designation of Storage Area	12-32	I/O Communications Area	12-63
Chaining	12-33		
Data Chaining	12-35	<b>Chapter 13. Operator Facilities</b>	13-1
Command Chaining	12-36	Basic Operator Facilities	13-1
Skipping	12-36	Address-Compare Controls	13-1
Program-Controlled Interruption	12-36	Alter-and-Display Controls	13-2
Commands	12-37	Check Control	13-2
Write	12-39	Check-Stop Indicator	13-2
Read	12-39	IML Controls	13-2
Read Backward	12-39	Interrupt Key	13-2
Control	12-40	Interval-Timer Control	13-3
Sense	12-40	Load Indicator	13-3
Transfer in Channel	12-42	Load-Clear Key	13-3
Command Retry	12-42	Load-Normal Key	13-3
Conclusion of Input/Output Operations	12-43	Load-Unit-Address Controls	13-3
Types of Conclusion	12-43	Machine-Save Key	13-3
		Manual Indicator	13-3

Mode Indicator	13-3
Power Controls	13-4
Rate Control	13-4
Restart Key	13-4
Save Indicator	13-4
Start Key	13-4
Stop Key	13-4
Storage-Size Control	13-4
System Indicator	13-5
System-Reset-Clear Key	13-5
System-Reset-Normal Key	13-5
Test Indicator	13-5
TOD-Clock Control	13-5
Wait Indicator	13-5

## Appendix A. Number Representation and Instruction-Use

Examples	A-1
Number Representation	A-2
Binary Integers	A-2
Signed Binary Integers	A-2
Unsigned Binary Integers	A-3
Decimal Integers	A-3
Floating-Point Numbers	A-4
Conversion Example	A-5
Instruction-Use Examples	A-5
Machine Format	A-5
Assembler-Language Format	A-5
General Instructions	A-6
ADD HALFWORD (AH)	A-6
AND (N, NR, NI, NC)	A-6
AND (NI)	A-6
BRANCH AND LINK (BAL, BALR)	A-7
BRANCH ON CONDITION (BC, BCR)	A-7
BRANCH ON COUNT (BCT, BCTR)	A-8
BRANCH ON INDEX HIGH (BXH)	A-8
BRANCH ON INDEX LOW OR EQUAL (BXLE)	A-9
COMPARE HALFWORD (CH)	A-9
COMPARE LOGICAL (CL, CLR, CLI, CLC)	A-9
Compare Logical (CLR)	A-9
Compare Logical (CLI)	A-9
Compare Logical (CLC)	A-10
COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)	A-10
COMPARE LOGICAL LONG (CLCL)	A-11
CONVERT TO BINARY (CVB)	A-12
CONVERT TO DECIMAL (CVD)	A-12
DIVIDE (D, DR)	A-13
EXCLUSIVE OR (X, XR, XI, XC)	A-13
Exclusive OR (XI)	A-13
Exclusive OR (XC)	A-13
EXECUTE (EX)	A-14
INSERT CHARACTERS UNDER MASK (ICM)	A-15
LOAD (L, LR)	A-16
LOAD ADDRESS (LA)	A-16

LOAD HALFWORD (LH)	A-16
MOVE (MVI)	A-17
MOVE (MVC)	A-17
MOVE LONG (MVCL)	A-18
MOVE NUMERICS (MVN)	A-18
MOVE WITH OFFSET (MVO)	A-19
MOVE ZONES (MVZ)	A-19
MULTIPLY (M, MR)	A-20
MULTIPLY HALFWORD (MH)	A-20
OR (O, OR, OI, OC)	A-21
OR (OI)	A-21
PACK (PACK)	A-21
SHIFT LEFT DOUBLE (SLDA)	A-21
SHIFT LEFT SINGLE (SLA)	A-22
STORE CHARACTERS UNDER MASK (STCM)	A-22
STORE MULTIPLE (STM)	A-23
TEST UNDER MASK (TM)	A-23
TRANSLATE (TR)	A-23
TRANSLATE AND TEST (TRT)	A-24
UNPACK (UNPK)	A-25
Decimal Instructions	A-25
ADD DECIMAL (AP)	A-26
COMPARE DECIMAL (CP)	A-26
DIVIDE DECIMAL (DP)	A-26
EDIT (ED)	A-27
EDIT AND MARK (EDMK)	A-28
MULTIPLY DECIMAL (MP)	A-28
SHIFT AND ROUND DECIMAL (SRP)	A-29
Decimal Left Shift	A-29
Decimal Right Shift	A-29
Decimal Right Shift and Round	A-29
Multiplying by a Variable Power of 10	A-30
ZERO AND ADD (ZAP)	A-30
Floating-Point Instructions	A-30
ADD NORMALIZED (AE, AER, AD, ADR)	A-30
ADD UNNORMALIZED (AU, AUR, AW, AWR)	A-31
COMPARE (CE, CER, CD, CDR)	A-31
Multiprogramming and Multiprocessing Examples	A-32
Example of a Program Failure Using OR Immediate	A-32
COMPARE AND SWAP (CS, CDS)	A-32
Setting a Single Bit	A-33
Updating Counters	A-33

## Appendix B. Lists of Instructions B-1

## Appendix C. Condition-Code Settings C-1

## Appendix D. Table of Powers of 2 D-1

## Appendix E. Hexadecimal Tables E-1

## Index X-1

# Chapter 1. Introduction

## Contents

The IBM 4300 Processors	1-1
Compatibility	1-2
Compatibility Among 4300 Processors	1-2
Compatibility Between 4300 Processors and System/370	1-2
Control-Program Compatibility	1-2
Problem-State Compatibility	1-2

## The IBM 4300 Processors

The IBM 4300 Processors are small and moderately sized processors that have evolved from System/370. They may be used in one of two architectural modes of operation. When operating in the Extended Control Program Support: Virtual Storage Extended (ECPS:VSE) mode, a processor provides new facilities that are designed specifically to enhance the DOS/VSE control program. To run control programs such as VM/370 and OS/VS1, which do not use these facilities, a processor is placed in the System/370 mode. This publication describes the architecture of the 4300 Processors when operating in the ECPS:VSE mode.

The architecture of a machine 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 programs running on different machine implementations produce the results that are defined by a single architecture, the implementations are considered to be compatible.

The ECPS:VSE mode includes a new storage-control facility, called one-level addressing, for creating a single virtual storage of up to 16,777,216 bytes, which both the CPU and the channels address directly using one uniform set of virtual addresses. Mapping the virtual storage onto the real storage is performed internal to the machine.

The one-level-addressing facility provides new instructions and interruptions which the control

program uses to determine which parts of virtual storage currently are mapped onto real storage and thereby are made addressable. These instructions and interruptions, and the associated internal address-mapping functions, take the place of dynamic address translation (DAT) and channel indirect data addressing in System/370.

The ECPS:VSE mode also includes a new status-saving function, called machine save, which preserves the entire CPU state and the first 2,048 (2K) bytes of storage. The operator uses machine save in preparation for a complete storage dump. Machine save replaces the store-status function of System/370, which necessarily alters some of the storage to be dumped.

If multiple virtual storages are not required, the ECPS:VSE mode affords the following advantages when compared to System/370:

- Simpler storage-mapping function, with more of the function performed automatically by the machine
- Improved control-program performance, because the control program need not translate the virtual addresses of channel programs

Programming of the machine has been simplified, relative to System/370, by omitting the following functions:

- Multiprocessing and associated instructions
- Machine-check logout and full channel logout

These model-dependent logouts are replaced by internal facilities for diagnosing machine malfunctions. This removes model-dependent error-handling procedures from the control program and improves serviceability.

## Compatibility

### *Compatibility Among 4300 Processors*

Although models of the 4300 Processors 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 will give identical results on any model, provided that it:

1. Is not time-dependent.
2. Does not depend on system facilities (such as storage capacity, I/O equipment, or optional features) 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 should not depend on interruptions caused by the use of operation codes or command codes that in some models are not assigned or not installed. 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 logout. 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 in this publication to be unpredictable or model-dependent, or on special-purpose functions (such as emulators and assists) that are not described in this publication. This includes the requirement that the program should not depend on the assignment of I/O 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 this publication.

### *Compatibility Between 4300 Processors and System/370*

#### Control-Program Compatibility

If the preceding compatibility restrictions are observed, a program written for the 4300 Processors or System/370 will run on the other system. However, because of the compatibility restrictions, control programs cannot be transferred between these systems if they take advantage of facilities that are available on one system but not the other. In particular, the 4300 Processors do not offer the System/370 dynamic-address-translation facility in the ECPS:VSE mode and, hence, cannot execute programs which rely on this particular facility.

To provide full control-program compatibility between System/370 and the 4300 Processors, the 4300 Processors offer an alternate microprogram that causes the machine to assume the characteristics of a System/370 model. When the machine is in this mode, the operation of the machine is as described in the *IBM System/370 Principles of Operation*, GA22-7000.

#### Problem-State Compatibility

A high degree of compatibility exists at the problem-state level between 4300 Processors operating in the ECPS:VSE mode and System/370. Because the majority of a user's applications are written for the problem state, this problem-state compatibility is useful in many installations.

A program written to run in the problem state on 4300 Processors or System/370 will run on the other system, provided that it:

1. Observes the limitations described in the section "Compatibility Among 4300 Processors."
2. Is not dependent on results defined in this publication or in the *IBM System/370 Principles of Operation*, as appropriate, to be unpredictable or model-dependent (an extension of the fourth rule in the section "Compatibility Among 4300 Processors").
3. Is not dependent on control-program facilities which are unavailable on the system.

To allow the problem programmer to guard against the effects of facilities that are available on System/370 but not on 4300 Processors, this publication in several places describes the results of such effects. For example, when a program is written which shares storage in a multiprogramming environment on a single-CPU configuration,

precautions should be taken to allow such a program to run correctly on a multiple-CPU (multiprocessing) configuration.

Specifically, COMPARE AND SWAP, COMPARE DOUBLE AND SWAP, and TEST AND SET are the only instructions which should be used to create interlocks between concurrent programs. These are the only instructions that do not, between fetching and storing of the storage operand, permit another CPU to access the operand location. The instructions AND (NI or NC), EXCLUSIVE OR (XI or XC), and OR (OI or OC) should not be used for such interlocks.

Serialization of CPU operations, which is performed by all interruptions and by the execution of certain instructions, affects the sequence of events as observed by other CPUs in a multiprocessing configuration as well as by channels. (See the section "Serialization" in Chapter 5, "Program Execution.")

### **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 of storage (addresses 0-511). Other operation codes, bit positions, and low-storage locations are specifically noted as being available for programming use. The remaining ones are unassigned and reserved for future assignment to new facilities and other extensions of the architecture.

To ensure that existing programs run 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 this location to be changed.



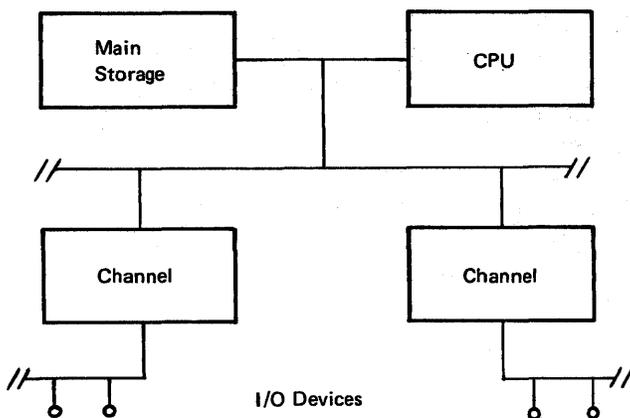
## Chapter 2. Organization

### Contents

Main Storage	2-1
Central Processing Unit	2-1
General Registers	2-2
Floating-Point Registers	2-2
Control Registers	2-3
Input and Output	2-3
Channels	2-4
Input/Output Devices and Control Units	2-4
Operator Facilities	2-4

Logically, IBM 4300 Processors consist of main storage, a central processing unit (CPU), operator facilities, and channels. The channels allow input/output (I/O) devices to be attached, usually through control units (see the figure "Logical Structure").

Specific processors may differ in their internal characteristics, the number and types of channels, the size of main storage, and the representation of the operator facilities. The differences in internal characteristics are apparent to the observer only as differences in machine performance.



Logical Structure

### Main Storage

The 4300 Processors provide fast-access main storage and storage-control functions to permit high-speed processing of data by the CPU and channels. The storage-control functions permit main storage to be controlled at two levels: real storage and virtual storage.

Real storage is the storage where data and instructions actually reside at the time they are accessed by the CPU and channels, but neither CPU programs nor channel programs can address real storage directly. The size of real storage depends on the model.

Virtual storage allows both CPU programs and channel programs to address an apparent main storage of up to 16,777,216 (16M) bytes. Virtual storage may be larger than the underlying real storage. If the virtual storage is larger than the real storage, a supervisory control program using the storage-control functions of the machine is required for controlling which parts of virtual storage are currently mapped onto real storage. This control is dynamic and transparent to the other programs except for the time delay.

### Central Processing Unit

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

The physical makeup of the CPU in the various models of the machine may be different, but the

logical function remains the same. The result of executing a valid instruction is the same for each model.

The CPU, in executing instructions, can process binary integers and floating-point numbers 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 CPU to another without affecting the logical results.

Instructions which the CPU executes fall into five classes: general, decimal, floating-point, control, and input/output instructions. The general instructions are used in performing fixed-point arithmetic operations and logical, branching, and other nonarithmetic operations. The decimal instructions operate on data in the decimal format, and the floating-point instructions on data in the floating-point format. The control instructions and the input/output instructions are privileged instructions that can be executed only when the CPU is in the supervisor state.

To perform its functions, the CPU may use a certain amount of internal storage. Examples of such functions include the mapping of virtual storage to real storage and the arithmetic and logical functions. This internal storage is not considered part of main storage and is not addressable by programs.

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, the control registers, and registers associated with the timing facilities. The current PSW contains information used to control instruction sequencing and to hold and indicate the states of the machine in relation to the program currently being executed. Registers associated with the timing facilities are the time-of-day (TOD) clock, the clock comparator, and the CPU timer.

Some models may use the same physical storage for both addressable main storage and unaddressable internal storage, and internal storage may actually contain the CPU registers. Such internal storage is not considered to be part of main storage, because the program cannot access the contents of internal storage by means of storage addresses.

The general, floating-point, and control registers are discussed separately in the following sections. (See also the figure "General, Floating-Point, and Control

Registers.") The instruction operation code determines which type of register is to be used in an operation.

### ***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 32 bits. 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 operations, two adjacent general registers are coupled, providing a 64-bit format. In these operations, the program must designate an even-numbered register, which contains the leftmost (high-order) 32 bits. The next higher-numbered register contains the rightmost (low-order) 32 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 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***

Four floating-point registers are available for floating-point operations. They are identified by the numbers 0, 2, 4, and 6. 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 operand occupies the leftmost bit positions of a floating-point register. The rightmost portion of the register is ignored and remains unchanged in arithmetic operations that call for short operands. Two pairs of adjacent floating-point registers can be used for extended operands: registers 0 and 2, and registers 4 and 6. Each of these pairs provides a 128-bit format.



## ***Channels***

A channel relieves the CPU of the burden of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O operations. A channel connects with the CPU, with main storage, and with control units.

A channel may be an independent unit, complete with the necessary logical and internal-storage capabilities, or it may time-share CPU facilities and be physically integrated with the CPU. In either case, channel functions are identical. Channels may be implemented, however, to have different maximum data-transfer capabilities.

There are three types of channels: byte-multiplexer, block-multiplexer, and selector channels.

## ***Input/Output Devices and Control Units***

Input/output devices include such equipment as card readers and punches, magnetic-tape units, direct-access storage, displays, keyboards, printers, tele-processing devices, and sensor-based equipment. Many I/O devices function with an external medium, such as punched cards or magnetic tape. Some I/O

devices handle only electrical signals, such as those found in sensor-based networks. In either case, I/O-device operation is regulated by a control unit. The control-unit function may be housed with the I/O device or in the CPU, or a separate control unit may be used. In all cases, the control-unit function 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.

## ***Operator Facilities***

The operator facilities provide the functions necessary for operator control of the machine. It may have an associated operator-console device which may also be used as an I/O device for communicating with the control program and problem programs.

The main functions provided by the operator facilities are system reset, clearing, initial program loading, start, stop, alter, and display.

## Chapter 3. Storage

### Contents

Information Formats and Addressing	3-1	Frame Index	3-5
Information Formats	3-1	Page and Frame Control	3-5
Storage Addressing	3-2	Capacity Counts	3-5
Integral Boundaries	3-2	Storage-Control Instructions	3-6
One-Level Addressing	3-2	Key-Controlled Protection	3-6
Storage Size	3-3	Protection Action	3-6
Pages	3-3	Accesses Protected	3-7
Page Frames	3-3	Reference and Change Recording	3-7
Page Description	3-3	Assigned Storage Locations	3-8
Storage Key	3-3	Storage While CPU Is in Operating State	3-8
Page Bits	3-4	Storage While CPU Is in Load State	3-10
Page States	3-4		

This chapter discusses the representation of information in storage, how information is addressed, and the one-level-addressing facility for controlling virtual and real storage. The chapter also contains a list of permanently assigned storage locations.

The term "main storage" is used generically to describe both virtual and real storage, in order to distinguish this fast-access storage from auxiliary storage, such as direct-access storage devices. Because, in this publication, most references to main storage apply to virtual storage, the abbreviated term "storage" is generally used in place of "virtual storage" when the meaning is clear.

All addresses of storage locations are virtual addresses, because they always refer to virtual storage. Hence, when applied to main storage, address means virtual address in this publication.

### Information Formats and Addressing

#### *Information Formats*

Information is transmitted between storage and the CPU or a channel in units of eight bits, or a multiple of eight bits, at a time. Each eight-bit unit of information is called a byte, the basic building block of all formats. The bits in a byte are numbered 0 through 7, from left to right.

The bits in an address, which is 24 bits long, are numbered 8 through 31. Within any other fixed-length format of multiple bytes, the bits making up the format are consecutively numbered from left to right, starting with the number 0. Leftmost bits

are sometimes referred to as the "high-order" bits and rightmost bits as the "low-order" bits.

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 manual to the length of data fields and registers exclude mention of the associated check bits. All storage capacities are expressed in number of bytes.

Bytes may be handled separately or grouped together in fields. A halfword is a group of two consecutive bytes and is the basic building block of instructions. A word or fullword is a group of four consecutive bytes; a doubleword is a group of eight bytes. The location of any field or group of bytes is specified by the address of its leftmost byte.

The length of fields is either implied by the operation to be performed or stated explicitly as part of the instruction. When the length is implied, the information is said to have a fixed length, which can be one, two, four, or eight bytes.

When the length of a field is not implied by the operation code, but is stated explicitly, the information is said to have variable field length. Variable-length operands are variable 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.

### Storage Addressing

Byte locations in storage are consecutively numbered, left to right, starting with 0; each number is considered the address of the corresponding byte. A group of bytes in storage is addressed by the leftmost byte of the group. The number of bytes in the group is either implied or explicitly defined by the operation. Addresses are 24-bit binary numbers, which provide 16,777,216 ( $2^{24}$  or 16M) byte addresses.

Storage addressing wraps around from the maximum byte address, 16,777,215, to address 0. In a 16M-byte storage, information may be located partially in the last and partially in the first locations of storage and is processed without any special indication of crossing the maximum-address boundary.

### Integral Boundaries

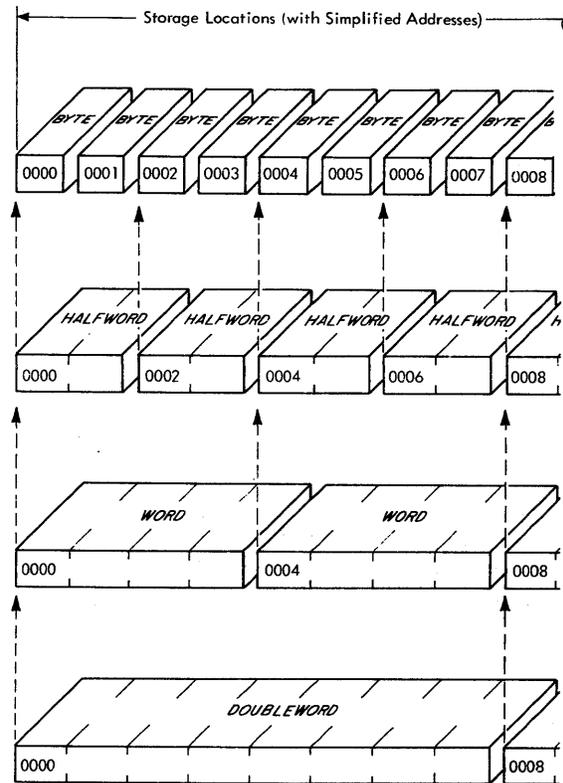
Certain units of information must be located in storage on an integral boundary. A boundary is called integral for a unit of information when its storage address is a multiple of the length of the unit in bytes. Special names are given to fields of two, four, and eight bytes when they are located on 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.

Thus, a word is on an integral boundary when it is located in storage so that its address (that is, the address of the leftmost of the four bytes) is a multiple of the number 4. Similarly, a halfword is on an integral boundary when it has an address that is a multiple of the number 2, and a doubleword is on an integral boundary when it has an address that is a multiple of the number 8. (See the figure "Integral Boundaries for Halfwords, Words, and Doublewords.")

When storage addresses designate halfwords, words, and doublewords on integral boundaries, the binary representation of the address contains one, two, or three low-order zero bits, respectively.

Instructions must appear on halfword integral boundaries, and channel-command words and the storage operands of certain instructions must appear on other integral boundaries. The storage operands

of most instructions do not have boundary-alignment requirements.



Integral Boundaries for Halfwords, Words, and Doublewords

### Programming Note

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.

### One-Level Addressing

The one-level-addressing facility may be used by the control program to create virtual storage that is larger than the actual capacity of the underlying real storage. Other programs and channels address this virtual storage directly as if all data and instructions actually resided in virtual storage.

Main storage is volatile; that is, the contents are not preserved when power is off.

## Storage Size

The storage size is the number of addressable byte locations provided in virtual storage. A model may allow one or more storage sizes. If more than one storage size is provided, the current storage size is determined by the manual storage-size control during an initial microprogram loading (IML) operation. The storage size cannot be changed by programming.

The storage size is always a multiple of 2,048 (2K) bytes, up to a maximum of 16,777,216 (16M) bytes.

When the storage size exceeds the size of real storage, the parts of virtual storage which are currently not directly accessible may be kept on auxiliary storage, such as direct-access storage devices (DASD). The transfer of the contents of virtual storage to and from auxiliary storage may be controlled by a supervisory control program using I/O instructions in such a way that the remaining CPU programs and channel programs can address any part of virtual storage as if it were all directly accessible.

Storage addresses range from zero to one less than the storage size. If the CPU attempts to refer to a storage location that is not provided or to the corresponding page description (see below), that attempt is indicated by an addressing exception or, in the case of the LOAD FRAME INDEX instruction, by the condition code. If an I/O operation attempts to access a storage location that is not provided, the operation is terminated by a channel program check.

Normally, the indication that a location is not provided is given only when the information associated with that location is actually required, and not when the operation can be completed without that information.

When the storage size is set to the maximum of 16M bytes, all storage locations are provided; addressing exceptions or channel program checks for CCW or data locations cannot occur.

## Pages

Virtual storage is divided into pages, each page consisting of 2,048 ( $2^{11}$ ) consecutive bytes on a 2,048-byte address boundary. Virtual storage has up to 8,192 ( $2^{13}$ ) pages of storage. The size of virtual storage and, hence, the number of pages provided depend on the model and on the setting of the manual storage-size control, if one is provided.

Storage-control instructions, except for INSERT STORAGE KEY and SET STORAGE KEY, refer to a page by the address of any byte in that page. The low-order 11 bits of an operand address referring to

a whole page are ignored. The INSERT STORAGE KEY and SET STORAGE KEY instructions also use a page address, but the low-order four bits of their operand address must be zeros.

## Page Frames

Real storage is divided into page frames, each capable of containing the data for one page of virtual storage. The size of real storage and, hence, the number of page frames present in the machine depend on the model. Real storage is not explicitly addressable by CPU programs and channel programs.

A virtual-storage page, to be accessible to CPU programs and channel programs, must be associated with a real-storage page frame. An instruction is provided which assigns to a page a free page frame selected by the machine. This instruction is said to connect the page to its assigned frame. Thereafter, the page frame is referred to by the address of the corresponding page. When any previous contents of the page have been retrieved from external storage and the page is ready for accessing by a CPU program, another instruction is used to make the page addressable.

As the supply of free page frames diminishes, the control program may make a page not addressable and, if any bytes in the page have been changed, write the contents of the page on auxiliary storage. An instruction may then be issued to disconnect the page, thus freeing its frame.

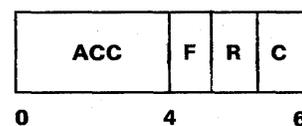
A page frame cannot be assigned to more than one page at a time.

## Page Description

Associated with each virtual-storage page which is provided are a seven-bit storage key, three programmable page bits, a page state, and the frame index of the page frame currently assigned to the page, if any. This information, called the page description, is maintained in internal storage.

## Storage Key

A control field, called storage key, is associated with each page that is provided.



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

**Access-Control Bits (ACC):** The four access-control bits, bits 0-3, are matched with the four-bit access key whenever information is stored, or whenever information is fetched from a location that is protected against fetching.

**Fetch-Protection Bit (F):** The fetch-protection bit, bit 4, controls whether 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 protection applies both to 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 page 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 page.

The entire storage key is set by SET STORAGE KEY and inspected by INSERT STORAGE KEY. The reference and change bits are also set by SET PAGE BITS and inspected by INSERT PAGE BITS. Additionally, the instruction RESET REFERENCE BIT provides a means of inspecting the reference and change bits and of setting the reference bit to zero.

### Page Bits

The three programmable page bits associated with each page may be set by the instruction SET PAGE BITS and inspected by INSERT PAGE BITS. The page bits are disregarded by the machine during other operations.

### Programming Note

The page bits may be used by the program to assist in managing pages on auxiliary page storage. For example, one of the bits may indicate whether a version of the corresponding page exists on auxiliary storage.

### Page States

A page may be in one of three states:

- Disconnected
- Connected
- Addressable

If disconnected, the page does not have a page frame assigned to it. Any attempt by the CPU to access a disconnected page causes a page-access exception or, when a CLEAR PAGE instruction is being executed, a page-state exception. Any attempt by a channel to access a disconnected page, in order to fetch a CCW or to access a data area designated during the execution of a CCW, creates an I/O-interruption condition indicating protection check. However, if a CCW is prefetched, a protection check is not indicated until the CCW is due to be executed.

If connected, the page has a page frame assigned to it. A connected page may be accessed, if the protection mechanism permits:

1. By I/O channels
2. By the CPU as the operand of a CLEAR PAGE instruction

Except when executing CLEAR PAGE, an attempt by the CPU to access a connected page causes a page-access exception.

If addressable, the page has a page frame assigned to it, and the CPU and I/O channels may access the page if the protection mechanism permits.

Although the addressable state implies that the page is also connected, the term "connected state" refers specifically to the state of a page that is neither addressable nor disconnected.

The page state is checked for all storage accesses to locations that are explicitly or implicitly accessed by the CPU or by a channel.

The page state is changed by instructions, which may make the transition from the disconnected to the connected state and from the connected to the addressable state, or vice versa. The instructions are CONNECT PAGE, DECONFIGURE PAGE, DISCONNECT PAGE, MAKE ADDRESSABLE, and MAKE UNADDRESSABLE. Most of these instructions may also be applied to pages which are already in the desired state. An attempt to change directly from disconnected to addressable, or vice versa, causes a page-transition exception. A page-transition exception is also caused by DECONFIGURE PAGE when applied to a disconnected page.

The first page, page 0, containing byte locations 0 to 2047, is always addressable. It cannot be placed in the connected or disconnected state.

The clear-reset function causes a number of consecutive pages, starting with page 0, to be assigned to page frames, cleared, and placed in the addressable state. (See the section "Clear Reset" in Chapter 4, "Control.")

### Programming Notes

1. The three page states permit programs to distinguish pages in the disconnected state, which cannot be accessed at all, from pages in the connected state, which are undergoing I/O operations to or from external page storage, and from pages in the addressable state, which are ready for normal storage access.
2. The storage-control instructions other than CLEAR PAGE operate on page descriptions, not pages. Instructions which operate on page descriptions do not make storage accesses and do not cause page-access or protection exceptions.
3. All channel accesses to storage appear as if they referred to pages and not to the associated page frames. If a page that is being accessed by a channel becomes disconnected and another channel access is attempted, protection check is indicated, and the I/O operation terminates. If the page becomes disconnected and then becomes reconnected before protection check is indicated, protection check may be indicated subsequently, or accesses may continue using the newly assigned page frame.

### Frame Index

A unique 16-bit binary integer is assigned to each page frame existing in the machine. This integer is the frame index of the page frame. The value of the frame index ranges from zero up to, but not including, the existing-frame-capacity count (EFCC) (see the section "Capacity Counts" in this chapter).

When a CONNECT PAGE instruction connects a page to a frame, the frame index of the connected frame is returned by the instruction. The frame index remains associated with that page until the page is disconnected. When the same page is connected again, the new frame index is, in general, different and unpredictable unless there was only one free frame remaining.

The frame index for an already connected or addressable page may be displayed by LOAD FRAME INDEX.

When DECONFIGURE PAGE makes a page frame unavailable, the frame index of that frame will not recur until a clear-reset operation is performed.

The frame index currently associated with a page is part of its page description. A disconnected page has no frame index, and LOAD FRAME INDEX returns no frame-index value for a disconnected page.

### Programming Notes

1. The frame index assists the control program in maintaining compact tables of connected or addressable pages. The frame index is not, and should not be interpreted as, the address of a frame in real storage. The algorithm for assigning a frame index to a page frame is implementation-dependent. Programming should not depend on a particular algorithm.
2. The set of frame indexes is dense if the EFCC equals the AFCC, that is, if there are no unavailable page frames. It becomes nonsense to the extent that frames are made unavailable by DECONFIGURE PAGE or by maintenance intervention.
3. DECONFIGURE PAGE removes a page frame from contention for connection when a machine check has indicated damage to a page. This can be done only while the frame is connected to a page, because a frame cannot be addressed directly.

### Page and Frame Control

#### Capacity Counts

Four internally maintained counts are defined to assist the program in managing pages and page frames. Each count is a 16-bit unsigned binary integer. The counts are set or updated by the machine. They are displayed by the STORE CAPACITY COUNTS instruction, which stores each count as a 32-bit integer with 16 high-order zero bits.

The page-capacity count (PCC) is the number of virtual-storage pages provided by the machine. The pages have page addresses from 0 to PCC minus one. The value of the PCC is equal to the storage size divided by 2,048; it is set during clear reset according to the current setting of the manual storage-size control, if one is provided.

The existing-frame-capacity count (EFCC) is the number of page frames existing in a particular implementation of the machine. The EFCC reflects the total capacity of real storage. The value of the EFCC is set during clear reset.

The available-frame-capacity count (AFCC) is the number of page frames connected or available for connection to pages. The value of AFCC may be equal to or less than the EFCC. During CPU operation, the AFCC may be decreased by the instruction DECONFIGURE PAGE. The clear-reset function initializes the AFCC to the value of the EFCC less the number of frames that are kept unavailable for connection by maintenance intervention.

The free-frame-capacity count (FFCC) is the number of available page frames that are currently not connected to pages. The value of the FFCC may range from zero to the AFCC minus one. During CPU operation, the value of the FFCC may be changed by the instructions CONNECT PAGE and DISCONNECT PAGE. The clear-reset operation initializes the FFCC to zero or to the value of AFCC minus PCC, depending on whether the AFCC is less than the PCC or not.

Since page 0 must always be addressable, the frame connected to page 0 is considered available but not free. Hence, the minimum value of the AFCC is one, and the maximum value of the FFCC is the AFCC minus one.

### Storage-Control Instructions

CONNECT PAGE is used to change a page from the disconnected to the connected state. MAKE ADDRESSABLE changes a page from connected to addressable. MAKE UNADDRESSABLE changes the page state from addressable to connected. DISCONNECT PAGE changes the page state from connected to disconnected. DECONFIGURE PAGE disconnects a connected page and makes the corresponding page frame and its frame index unavailable. LOAD FRAME INDEX tests the page state of a page and displays its frame index, if any. These six instructions do not change or check the storage key of the specified pages.

CLEAR PAGE sets the contents of a page to zero and validates the page.

SET STORAGE KEY replaces the storage key of a page. INSERT STORAGE KEY retrieves the storage key of a page except, in the BC mode, for the reference and change bits. RESET REFERENCE BIT tests the reference and change bits and resets the reference bit to zero.

SET PAGE BITS tests the reference and change bits of a page and then explicitly sets them along with the three programmable page bits of that page. INSERT PAGE BITS retrieves the values of the three

page bits, the reference bit, and the change bit of a page.

All storage-control instructions are privileged.

## Key-Controlled Protection

Key-controlled protection is provided to protect the contents of storage from destruction or misuse caused by erroneous or unauthorized storing or fetching by the program. It affords protection against improper storing or against both improper storing and fetching, but not against improper fetching alone.

### Protection Action

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 the figure "Summary of Protection Action."

Fetch-Protection Bit of Storage Key	Conditions Key Relation	Is access to storage permitted?	
		Fetch	Store
0	Match	Yes	Yes
0	Mismatch	Yes	No
1	Match	Yes	Yes
1	Mismatch	No	No

#### 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.

#### Summary of Protection Action

When the access to storage is initiated by the CPU, and protection applies, the PSW key is the access key which is used as the comparand. The PSW key occupies bit positions 8-11 of the current PSW.

When the reference is made by a channel, and protection applies, the subchannel key associated with the I/O operation is the access key which is used as the comparand. The subchannel key is specified for

an I/O operation in bit positions 0-3 of the channel-address word (CAW); the subchannel key is later placed in bit positions 0-3 of the channel-status word (CSW) that is stored as a result of the I/O operation.

When a CPU access is prohibited because of protection, the operation is suppressed or terminated, and a program interruption for a protection exception takes place. When a channel access is prohibited, protection check is indicated in the CSW stored as a result of the operation.

When a store access is prohibited because of 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.

Key-controlled protection is always active, regardless of whether the CPU is in the problem or supervisor state, and regardless of the type of CPU instruction or channel-command word being executed.

### ***Accesses Protected***

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 protection.

All storage accesses by a channel to fetch a CCW or to access a data area designated during the execution of a CCW are subject to protection. However, if a CCW or output data is prefetched, a protection check is not indicated until the CCW is due to be executed or the data is due to be written.

Protection is not applied to accesses that are implicitly made by the CPU or channel for such sequences as:

- Interruptions,
- Updating the interval timer,
- Fetching the CAW during the execution of an I/O instruction,
- Storing the CSW by an I/O instruction or interruption,
- Storing channel identification during the execution of STORE CHANNEL ID,
- Limited channel logout, or
- 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.

## **Reference and Change Recording**

Reference recording provides information for use in selecting pages for replacement. Change recording provides information as to which pages have to have their contents saved when their connected page frames are to be reused.

The reference bit normally is set to one each time a location in the corresponding page is referred to either for storing or fetching of information. The change bit is set to one each time information is stored in the corresponding page.

Reference and change recording takes place for any storage access and applies to accesses made by the CPU, as well as accesses due to I/O operations. References to a storage location associated with interruptions and I/O instructions, such as references to the CAW, CSW, or PSW locations, are included. It is unpredictable whether updating of the interval timer causes the reference and change bits for location 80 to be set to ones.

References to the page operand of the following instructions do not cause storage access and do not cause reference and change bits to be set to ones implicitly:

CONNECT PAGE  
DECONFIGURE PAGE (both bits are set to zeros)  
DISCONNECT PAGE (both bits are set to zeros)  
INSERT PAGE BITS  
INSERT STORAGE KEY  
LOAD FRAME INDEX  
MAKE ADDRESSABLE  
MAKE UNADDRESSABLE  
RESET REFERENCE BIT (reference bit is set to zero)  
SET PAGE BITS (both bits are set explicitly)  
SET STORAGE KEY (both bits are set explicitly)

The change bit is not set to one for an attempt to store if the storage reference is not permitted, regardless of whether the CPU instruction responsible for the reference is suppressed or terminated. In particular, a CPU reference causing a protection or page-access exception, and an I/O reference causing a protection check, do not cause the change bit to be set to one.

The instructions SET PAGE BITS and SET STORAGE KEY may be used to set the reference and change bits explicitly to either zero or one. SET PAGE BITS also sets the condition code to indicate the previous state of the reference and change bits. The instruction RESET REFERENCE BIT tests both the reference and change bits and sets the condition code; it also sets the reference bit to zero. The

instructions DECONFIGURE PAGE and DISCONNECT PAGE set both the reference and change bits to zeros.

The record of references 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 conditions, a reference bit may be set to zero by other than explicit program action.

The record of changes provided by the change bit is also substantially accurate. Under certain conditions, the change bit may be set to one when storing is permitted but no storing has actually taken place. However, the change bit is always set to one when storing occurs and is never set to zero without explicit program action.

#### Programming Note

The accuracy of reference and change recording is sufficient to allow effective operation of page-replacement algorithms.

### Assigned Storage Locations

Assigned locations in storage have different uses when the CPU is in the operating state or in the load state. This section is summarized in the figure "Assigned Storage Locations."

#### Programming Note

In the BC mode, there is no implicit storing in locations 128 and above if all of the following conditions are met:

1. The manual check control is set to stop.
2. The MONITOR CALL and STORE CHANNEL ID instructions are not issued.
3. The page-capacity count is equal to or less than the available-frame-capacity count and all pages are addressable.

#### Storage While CPU Is in Operating State

This section shows the format and extent of the assigned storage locations while the CPU is in the operating state. Unless specifically noted, the usage applies to both the EC and BC modes.

0-7 *Restart New PSW:* The new PSW is fetched from locations 0-7 during a restart interruption.

- 8-15 *Restart Old PSW:* The current PSW is stored as the old PSW at locations 8-15 during a restart interruption.
- 24-31 *External Old PSW:* The current PSW is stored as the old PSW at locations 24-31 during an external interruption.
- 32-39 *Supervisor-Call Old PSW:* The current PSW is stored as the old PSW at locations 32-39 during a supervisor-call interruption.
- 40-47 *Program Old PSW:* The current PSW is stored as the old PSW at locations 40-47 during a program interruption.
- 48-55 *Machine-Check Old PSW:* The current PSW is stored as the old PSW at locations 48-55 during a machine-check interruption.
- 56-63 *Input/Output Old PSW:* The current PSW is stored as the old PSW at locations 56-63 during an I/O interruption.
- 64-71 *CSW:* The channel-status word (CSW) is stored at locations 64-71 during an I/O interruption. It, or a portion thereof, may be stored during the execution of START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, or HALT DEVICE, in which case condition code 1 is set.
- 72-75 *CAW:* The channel-address word (CAW) is fetched from locations 72-75 during the execution of START I/O and START I/O FAST RELEASE.
- 80-83 *Interval Timer:* Locations 80-83 contain the interval timer. The interval timer is updated whenever the CPU is in the operating state and the manual interval-timer control is set to enable.
- 88-95 *External New PSW:* The new PSW is fetched from locations 88-95 during an external interruption.
- 96-103 *Supervisor-Call New PSW:* The new PSW is fetched from locations 96-103 during a supervisor-call interruption.
- 104-111 *Program New PSW:* The new PSW is fetched from locations 104-111 during a program interruption.
- 112-119 *Machine-Check New PSW:* The new PSW is fetched from locations 112-119 during a machine-check interruption.

- 120-127 *Input/Output New PSW*: The new PSW is fetched from locations 120-127 during an I/O interruption.
- 132-135 *External-Interruption Code*: During an external interruption in the EC mode, the interruption code is stored at locations 134-135, and zeros are stored at locations 132-133.
- 136-139 *Supervisor-Call-Interruption Identification*: During a supervisor-call interruption in the EC mode, 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 137.
- 140-143 *Program-Interruption Identification*: During a program interruption in the EC mode, 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 141.
- 144-147 *Access-Exception Address*: During a program interruption due to a page-access exception, the address for which the exception is being indicated is stored at locations 145-147, and zeros are stored at location 144.
- 148-149 *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 148.
- 150-151 *PER Code*: During a program interruption due to a program event, the program-event-recording (PER) code is stored in bit positions 0-3 of location 150, and zeros are stored in bit positions 4-7 and at location 151. This field can be stored only when the instruction causing the PER condition was executed under the control of a PSW specifying the EC mode.
- 152-155 *PER Address*: During a program interruption due to a program event, the program-event-recording (PER) address is stored at locations 153-155, and zeros are stored at location 152. This field can be stored only when the instruction causing the PER condition was executed under the control of a PSW specifying the EC mode.
- 156-159 *Monitor Code*: During a program interruption due to a monitor event, the monitor code is stored at locations 157-159, and zeros are stored at location 156.
- 168-171 *Channel ID*: The four-byte channel-identification information is stored at locations 168-171 during the execution of STORE CHANNEL ID.
- 176-179 *Limited Channel Logout*: The limited-channel-logout information is stored at locations 176-179. This field may be stored only when the CSW or a portion of the CSW is stored.
- 185-187 *I/O Address*: During an I/O interruption in the EC mode, the two-byte I/O address is stored at locations 186-187, and zeros are stored at location 185.
- 216-223 *CPU-Timer Save Area*: During a machine-check interruption, the contents of the CPU timer are stored at locations 216-223.
- 224-231 *Clock-Comparator Save Area*: During a machine-check interruption, the contents of the clock comparator are stored at ~~locations~~ 224-231.
- 232-239 *Machine-Check-Interruption Code*: During a machine-check interruption, the machine-check-interruption code is stored at locations 232-239.
- 248-251 *Failing-Storage Address*: During a machine-check interruption, a failing-storage address, if any, is stored at locations 249-251, and zeros are stored at location 248.
- 352-383 *Floating-Point-Register Save Area*: During a machine-check interruption, the contents of the floating-point registers are stored at locations 352-383.
- 384-447 *General-Register Save Area*: During a machine-check interruption, the contents of the general registers are stored at locations 384-447.
- 448-511 *Control-Register Save Area*: During a machine-check interruption, the contents of the control registers are stored at locations 448-511.

### ***Storage While CPU Is in Load State***

0-7 ***IPL PSW:*** The first eight bytes read during the IPL initial read operation are stored at locations 0-7. The contents of these locations are used as the new PSW at the completion of the IPL operation. These locations may also be used for temporary storage at the initiation of the IPL operation.

8-15 ***IPL CCW1:*** Bytes 8-15 read during the IPL initial-read operation are stored at

locations 8-15. The contents of these locations are ordinarily used as the next CCW in an IPL CCW chain after completion of the IPL initial-read operation.

16-23 ***IPL CCW2:*** Bytes 16-23 read during the IPL initial-read operation are stored at locations 16-23. The contents of these locations may be used as another CCW in the IPL CCW chain to follow IPL CCW1.

Hex	Dec	
0	0	Restart New PSW
4	4	
8	8	Restart Old PSW
C	12	
10	16	
14	20	
18	24	External Old PSW
1C	28	
20	32	Supervisor-Call Old PSW
24	36	
28	40	Program Old PSW
2C	44	
30	48	Machine-Check Old PSW
34	52	
38	56	Input/Output Old PSW
3C	60	
40	64	Channel Status Word
44	68	
48	72	Channel Address Word
4C	76	
50	80	Interval Timer
54	84	
58	88	External New PSW
5C	92	
60	96	Supervisor-Call New PSW
64	100	
68	104	Program New PSW
6C	108	
70	112	Machine-Check New PSW
74	116	
78	120	Input/Output New PSW
7C	124	
80	128	
84	132	0000000000000000 External-Interrupt Code
88	136	0000000000000000 ILC0 Superv -Call-Irptn. Code
8C	140	0000000000000000 ILC0 Program-Interrupt Code
90	144	00000000 Access-Exception Address
94	148	00000000 Monitor CI # Per C 000000000000
98	152	00000000 PER Address
9C	156	00000000 Monitor Code
A0	160	
A4	164	
A8	168	Channel ID
AC	172	
B0	176	Limited Channel Logout
B4	180	
B8	184	00000000 I/O Address

Assigned Storage Locations

Hex	Dec	
BC	188	
C0	192	
C4	196	
C8	200	
CC	204	
D0	208	
D4	212	
D8	216	CPU-Timer Save Area
DC	220	
E0	224	Clock-Comparator Save Area
E4	228	
E8	232	Machine-Check Interruption Code
EC	236	
F0	240	
F4	244	
F8	248	00000000 Failing-Storage Address
FC	252	
100	256	
104	260	
108	264	
~ ~		
154	340	
158	344	
15C	348	
160	352	Floating-Point Register Save Area
164	356	
168	360	
16C	364	
170	368	
174	372	
178	376	
17C	380	
180	384	General-Register Save Area
184	388	
188	392	
18C	396	
~ ~		
1B4	436	
1B8	440	
1BC	444	
1C0	448	Control-Register Save Area
1C4	452	
1C8	456	
1CC	460	
~ ~		
1F4	500	
1F8	504	
1FC	508	

## Chapter 4. Control

### Contents

CPU States	4-1	General-Register Alteration	4-12
Wait State	4-2	Indication of Events Concurrently with Other Interruption Conditions	4-13
Supervisor and Problem States	4-2	Timing	4-15
Stopped, Operating, Load, and Check-Stop States	4-2	Time-of-Day Clock	4-15
EC and BC Modes	4-3	Format	4-15
Program-Status Word	4-4	States	4-16
Program-Status-Word Format in EC Mode	4-4	Setting and Inspecting the Value	4-16
Program-Status-Word Format in BC Mode	4-6	Clock Comparator	4-18
Control Registers	4-7	CPU Timer	4-18
Monitoring	4-8	Interval Timer	4-19
Program-Event Recording	4-9	Externally Initiated Functions	4-20
Control-Register Allocation	4-9	Resets	4-20
Operation	4-10	Program Reset	4-21
Identification of Cause	4-10	Initial Program Reset	4-23
Priority of Indication	4-11	Clear Reset	4-23
Storage-Area Designation	4-11	Power-On Reset	4-23
PER Events	4-12	Initial Program Loading	4-24
Successful Branching	4-12	Machine Save	4-25
Instruction Fetching	4-12		
Storage Alteration	4-12		

This chapter describes in detail a number of facilities for:

- Changing and interrogating the state of the CPU
- Measuring time
- Initiating certain operations externally
- In general, enhancing the efficiency, utility, and programmability of the machine

The information determining the state and controlling the operation of the CPU resides in the program-status word (PSW) and in control registers. Additional status and control information appears in permanently assigned storage locations which are listed in Chapter 3, "Storage." Supervisor state and a set of instructions that are valid only in supervisor state provide a means for avoiding unauthorized or inadvertent change of the machine state.

There are four facilities for measuring time:

- The time-of-day clock permits indication of calendar time.
- The clock comparator permits a program to be alerted at a particular value of the time-of-day clock.
- The CPU timer and the interval timer provide means for a program to be alerted after a specified time interval has elapsed.

There is a set of externally initiated functions for initializing the machine or for inspecting its status. These operations include resets, initial program loading, and machine save.

Two facilities enhance the usability of the machine. Monitoring is useful for performing various measurement functions. Program-event recording provides a means to assist in debugging programs.

### CPU States

Excluding states that may exist only during maintenance, the CPU has three sets of alternative states:

- Wait and nonwait
  - Supervisor and problem
  - Stopped, operating, load, and check stop
- These states differ in the way they affect CPU functions and in the way they are indicated and changed.

## ***Wait State***

In the wait state, no instructions are processed, whereas in the nonwait state, instruction processing proceeds in the normal manner. The CPU is interruptible in the wait state, provided it is enabled for the interruption source.

The CPU is in the wait state when bit 14 of the PSW is one. When bit 14 is zero, the CPU is in the nonwait state.

When the CPU is in the wait state, the wait indicator is on.

The updating of timing facilities is not affected by whether the CPU is in the wait or nonwait state.

## **Programming Note**

The wait state may be used to halt instruction processing until an I/O or external interruption occurs. To leave the wait state without manual intervention, the CPU must be enabled for the interruption source.

## ***Supervisor and Problem States***

The choice between supervisor and problem state determines whether the full set of instructions is valid.

In the problem state, only those instructions are valid that cannot be used to affect system integrity and that do not pertain to maintenance functions. In the supervisor state, all instructions are valid.

The instructions that are valid only when the CPU is in the supervisor state are called privileged instructions. They include those which modify or inspect storage keys and other parts of the page descriptions, those which modify or inspect the control fields in the PSW and in control registers, and those which pertain to timing facilities and input/output. A privileged instruction encountered in the problem state constitutes a privileged-operation exception and causes a program interruption.

The CPU is in the problem state when bit 15 of the PSW is one. When bit 15 is zero, the CPU is in the supervisor state.

The updating of timing facilities is not affected by whether the CPU is in the problem or supervisor state.

## **Programming Note**

The CPU may be switched between the wait and nonwait states and between the supervisor and problem states only by introducing an entire new PSW. This may be performed by an interruption or by initial program loading.

The instruction LOAD PSW may be used to switch from the supervisor to the problem state and from the nonwait to the wait state, but not vice versa. To allow the return from an interruption-handling routine by LOAD PSW, the PSW for the interruption-handling routine must specify the supervisor state.

## ***Stopped, Operating, Load, and Check-Stop States***

The stopped, operating, load, and check-stop states are four mutually exclusive states. 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 interruptions, subject to the control of the wait bit and of bits for masking interruptions, and in the manner specified by the setting of the manual rate control. The CPU is in the load state during the initial-program-loading operation. The CPU may enter the check-stop state only as the result of machine malfunctions.

A change between these four CPU states cannot be effected by the program. The states are not controlled or identified by bits in the PSW.

The state of the CPU is manually changed from stopped to operating when the start function is performed or when the restart key is activated and a restart interruption occurs.

The start function is performed when the start key is activated. If the rate control is set to instruction step, the start function causes one unit of operation to be executed, after which the stop function is performed. The effect of the start function is unpredictable when the stopped state has been entered by means of a reset.

The state of the CPU is changed from operating to stopped by the stop function. The stop function is performed when:

- The stop key is activated while the CPU is in the operating state
- The CPU has finished the execution of a unit of operation with the rate control set to instruction step

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 CPU is in the wait state, the transition takes place immediately. 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, all pending, allowed interruptions are taken 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. When the CPU is in the stopped state, interruption conditions remain pending.

When the CPU is in the wait state and the rate control is set to instruction step, initiating the start function causes no instruction to be executed, but all pending, allowed interruptions are taken before the CPU returns to the stopped state.

The CPU is also placed in the stopped state:

- When a reset is completed, except when the reset operation is performed as part of initial program loading, and
- When an address comparison indicates equality and stopping on the match is specified

The execution of resets is described in the section "Resets" in this chapter, and address comparison is described in the section "Address-Compare Controls" in Chapter 13, "Operator Facilities."

If the CPU is in the stopped state when the start key is activated, the CPU executes at least one instruction before taking an interruption for which the CPU is enabled.

When the CPU is in the stopped state, the manual indicator is on.

The CPU enters the load state when the load-normal or load-clear key is activated (see the section "Initial Program Loading" in this chapter). While the CPU is in the load state, the load indicator is on. When the initial-program-loading operation is completed successfully, the CPU state changes from load to operating, provided the rate control is set to process; if the rate control is set to instruction step, the CPU state changes from load to stopped.

The check-stop state, which the CPU enters on certain types of machine malfunction, is described in Chapter 11, "Machine-Check Handling." The check-stop indicator is on when the CPU is in the check-stop state.

The interval timer is updated only 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 time-of-day clock is updated whenever power is on.

### 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 interruption sequence results from a program new PSW with a PSW-format error of the type that is recognized early or from a continuous interruption condition, such as one due to the CPU timer.
3. Input/output operations continue to completion after the CPU enters the stopped state. The interruption conditions due to completion of I/O operations remain pending when the CPU is in the stopped state.

### EC and BC Modes

Two control modes are provided for the formatting and use of control and status information: the extended-control (EC) mode and the basic-control (BC) mode. Certain functions available in the EC mode are not available, or are available in a restricted form, in the BC mode. The mode currently in effect is specified by PSW bit 12. Bit 12 is one for the EC mode and zero for the BC mode.

Program-event recording can be specified only in the EC mode, because the PSW bit to turn this function on is not available in the BC mode.

In the EC mode, I/O interruptions can be controlled individually for up to 32 channels using the correspondingly numbered 32 mask bits in control register 2; there is also a summary-mask bit for I/O interruptions, bit 6 of the PSW. The BC mode provides these capabilities only for channels 6 and up: these channels are individually controlled by the corresponding bits of control register 2, as well as the summary-mask bit, bit 6 of the PSW; channels 0-5 are controlled separately by bits 0-5 of the PSW and are not subject to the summary mask or to mask bits in control register 2.

When interruptions occur while in the EC mode, the interruption code and instruction-length code are stored at various permanently assigned storage locations according to the class of interruptions. In the BC mode, the interruption code and instruction-length code for all except machine-check interruptions are placed in the PSW.

The program-mask and condition-code fields in the PSW are allocated to different bit positions in the two control modes. INSERT STORAGE KEY provides the reference and change bits when in the EC mode but produces zeros in the corresponding bit positions when in the BC mode.

**Programming Notes**

1. The BC mode provides a PSW format that is compatible with the PSW of System/360.
2. The choice between EC and BC modes affects only those aspects of operation that are specifically defined to be different for the two modes. It does not affect the operation of any functions that are not associated with the control bits in the PSW provided only in the EC mode, and it does not affect the validity of any instructions. The instructions SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK perform the specified function on the leftmost byte of the PSW regardless of the mode specified by the current PSW. The instruction SET PROGRAM MASK introduces a new program mask regardless of the PSW bit positions occupied by the mask.

**Program-Status Word**

The current program-status word (PSW) contains information required for the proper execution of the currently active program. The PSW is 64 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.

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. The status of the CPU can be changed by loading a new PSW or part of a PSW.

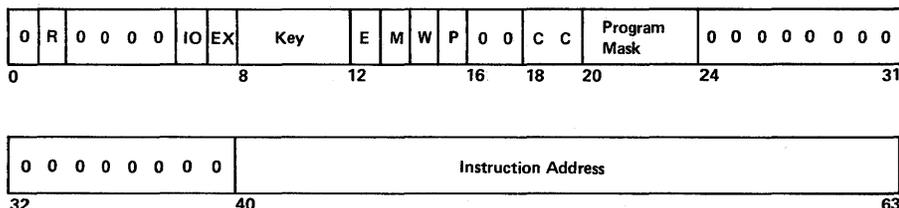
The instruction LOAD PSW introduces a new PSW. Other instructions operate on only part of the PSW. SET PROGRAM MASK introduces a new condition code and the four program-mask bits; SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK change bits in the leftmost byte of the PSW; SET PSW KEY FROM ADDRESS introduces a new PSW key; and the instruction address is updated by sequential instruction execution and replaced by successful branches.

The instruction INSERT PSW KEY places the PSW key in a register; STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK store the leftmost byte of the PSW; and BRANCH AND LINK loads the condition code, program mask, and instruction address in a register, as well as the instruction-length code, which in the BC mode is part of the PSW.

A new or modified PSW becomes active (that is, the information introduced into the current PSW assumes control over the CPU) when an interruption or the execution of an instruction is completed. The interruption for program-event recording 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.

***Program-Status-Word Format in EC Mode***



PSW Format in EC Mode

The following is a summary of the functions of the PSW fields in the EC mode.

**PER Mask (R):** Bit 1 controls whether the CPU is enabled for interruption by program events associated with program-event recording (PER). When the bit is zero, no program event can cause an interruption. When the bit is one, interruptions are permitted subject to the PER-event-mask bits in control register 9.

**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 channel-mask bits in control register 2: when a channel-mask bit is zero, the associated channel cannot cause an I/O interruption; when the channel-mask bit is one, an interruption condition at the channel can cause an interruption.

**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.

**PSW Key:** Bits 8-11 form the access key for storage references by the CPU. This PSW key is matched with a storage key whenever information is stored, or whenever information is fetched from a location that is protected against fetching.

**EC Mode (E):** Bit 12, which controls the format of the PSW and the mode of operation of the CPU, is one in the extended-control (EC) mode.

**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 in the wait state. When bit 14 is zero, the CPU is not in the wait state.

**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.

**Condition Code (CC):** Bits 18 and 19 are the two bits of the condition code.

**Program Mask:** Bits 20-23 are the four program-mask bits. Each bit is associated with a program exception, as follows:

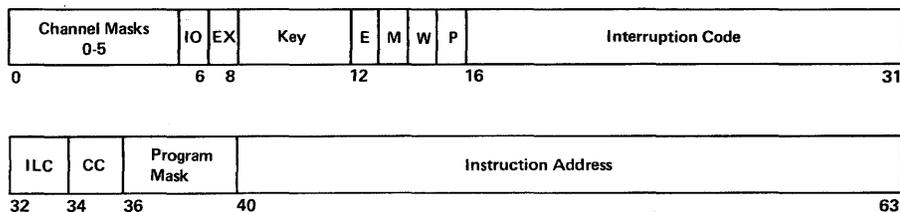
Program Mask Bit	Program Exception
20	Fixed-point overflow
21	Decimal overflow
22	Exponent underflow
23	Significance

When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The significance-mask bit also determines the manner in which floating-point addition and subtraction are completed.

**Instruction Address:** Bits 40-63 form the instruction address. This address designates the location of the leftmost byte of the next instruction.

Bit positions 0, 2-5, 16, 17, and 24-39 are unassigned and must contain zeros. A specification exception is recognized when these bit positions do not contain zeros.

## Program-Status-Word Format in BC Mode



PSW Format in BC Mode

The following is a summary of the functions of the PSW fields in the BC mode:

**Channel Masks 0-5:** Bits 0-5 control whether the CPU is enabled for I/O interruptions from channels 0-5, respectively. When a bit is zero, the associated channel cannot cause an I/O interruption. When the bit is one, an interruption condition at the channel can cause an I/O interruption.

**I/O Mask (IO):** Bit 6 controls whether the CPU is enabled for I/O interruptions from channels 6 and higher. When the bit is zero, these channels cannot cause I/O interruptions. When the bit is one, I/O interruptions are subject to the channel-mask bits of the corresponding channels in control register 2: when a channel-mask bit is zero, the associated channel cannot cause an I/O interruption; when the channel-mask bit is one, an interruption condition at the channel can cause an interruption.

**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.

**PSW Key:** Bits 8-11 form the access key for storage references by the CPU. This PSW key is matched with a storage key whenever information is stored, or whenever information is fetched from a location that is protected against fetching.

**EC Mode (E):** Bit 12, which controls the format of the PSW and the mode of operation of the CPU, is zero in the basic-control (BC) mode.

**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 in the wait state. When bit 14 is zero, the CPU is not in the wait state.

**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.

**Interruption Code:** Bits 16-31 in the old PSW, which is stored during a program, supervisor-call, external, or I/O interruption, identify the cause of the interruption. When a new PSW is introduced, the contents of this field are ignored.

**Instruction-Length Code (ILC):** The code in bit positions 32 and 33 indicates the length of the last-interpreted instruction when a program or supervisor-call interruption occurs or when BRANCH AND LINK is executed. When a new PSW is introduced, the contents of this field are ignored.

**Condition Code (CC):** Bits 34 and 35 are the two bits of the condition code.

**Program Mask:** Bits 36-39 are the four program-mask bits. Each bit is associated with a program exception, as follows:

<u>Program Mask Bit</u>	<u>Program Exception</u>
36	Fixed-point overflow
37	Decimal overflow
38	Exponent underflow
39	Significance

When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The significance-mask bit also determines the manner in which floating-point addition and subtraction are completed.

**Instruction Address:** Bits 40-63 form the instruction address. This address designates the location of the leftmost byte of the next instruction.

## Control Registers

The control registers provide a means for maintaining and manipulating control information that resides outside the PSW.

The addressing structure provides for sixteen 32-bit control registers. The contents of these registers are part of the CPU state. The instruction LOAD CONTROL loads control information from storage into control registers, whereas STORE CONTROL transfers information from control registers to storage.

One or more specific bit positions in control registers are assigned to each function requiring such register space. The bits control the indicated function.

STORE CONTROL returns the information placed in the assigned register positions by LOAD CONTROL or by initial program reset. Values corresponding to unassigned register positions are unpredictable.

When the registers are loaded, the information is not checked for exceptions.

The definition of each assigned control-register position appears elsewhere with the description of the function with which the register position is associated. The figure "Control-Register Assignments" here is a summary that shows the function with which each assigned field is associated and the initial value of the field upon execution of initial program reset.

### Programming Note

To ensure that existing programs run if and when new facilities using additional control-register positions are installed, the programs should load only zeros in unassigned control-register positions. Although STORE CONTROL may provide zeros in bit positions corresponding to unassigned register positions, the program should not depend on such zeros.

CR	Bits	Name of Field	Associated With	Initial Value
0	0	Block-multiplexing control	Block-multiplexing channels	0
0	1	SSM-suppression control	SET SYSTEM MASK	0
0	20	Clock-comparator mask	Clock comparator	0
0	21	CPU-timer mask	CPU timer	0
0	24	Interval-timer mask	Interval timer	1
0	25	Interrupt-key mask	Interrupt key	1
0	26	External-signal mask	External signal	1
2	0-31	Channel masks	Channels	1
8	16-31	Monitor masks	MONITOR CALL	0
9	0	Successful-branching-event mask	Program-event recording	0
9	1	Instruction-fetching-event mask	Program-event recording	0
9	2	Storage-alteration-event mask	Program-event recording	0
9	3	GR-alteration-event mask	Program-event recording	0
9	16-31	PER <sup>1</sup> general-register masks	Program-event recording	0
10	8-31	PER starting address	Program-event recording	0
11	8-31	PER ending address	Program-event recording	0
14	0	(Not used)	Machine-check handling	1
14	4	Recovery-report mask	Machine-check handling	0
14	5	Degradation-report mask	Machine-check handling	0
14	6	External-damage-report mask	Machine-check handling	1
14	7	Warning mask	Machine-check handling	0

**Explanation:**

The fields not listed are unassigned. The initial value of unassigned register positions is unpredictable.

<sup>1</sup>PER means program-event recording.

**Control-Register Assignments**

**Monitoring**

Monitoring provides the capability for passing control to a monitoring program when selected indicators are reached in the monitored program. The indicators are MONITOR CALL instructions implanted in the monitored program. When executed, these instructions cause a program interruption for monitoring to take place, provided an interruption is allowed for the monitor class specified by the instruction. Along with the interruption, the monitor-class number and a monitor code are stored for subsequent use by the monitoring program.

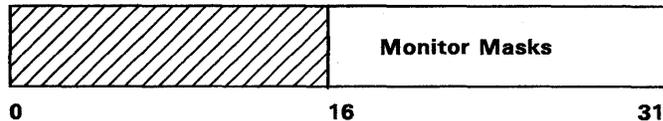
Monitoring includes the instruction MONITOR CALL, which designates one of 16 monitoring classes, together with a set of 16 monitor masks in a control register. One mask bit is associated with each class. The execution of the instruction causes a

program interruption when the monitor-mask bit for the class specified in the instruction is one.

Monitoring is available in both the EC and BC modes.

The monitor-mask bits are in bit positions 16-31 of control register 8.

**Control Register 8**

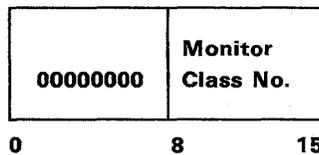


The mask bits, bits 16-31, correspond to monitor classes 0-15, respectively. Any number of monitor-mask bits may be on at any one time; together they

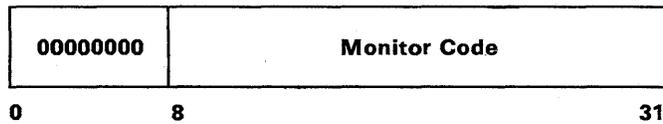
specify the classes of monitor events that are monitored at that time. The mask bits are initialized to zero.

When a MONITOR CALL instruction is interpreted for execution and the corresponding monitor-mask bit is one, a program interruption for monitoring occurs. The cause of the interruption is identified by setting bit 9 of the interruption code to one, and by the information stored at locations 148-149 and 156-159. The format of the information stored at these locations is the same in the EC and BC modes and is as follows:

Locations 148-149



Locations 156-159



The contents of bit positions 8-15 of MONITOR CALL are stored at location 149 and constitute the monitor-class number. The address specified by the B<sub>1</sub> and D<sub>1</sub> fields of the instruction forms the monitor code, which is stored at locations 157-159. Zeros are stored at locations 148 and 156.

### Program-Event Recording

The purpose of the program-event recording (PER) facility is to assist in debugging programs. It permits the program to be alerted to the following PER events:

- Successful execution of a branch instruction
- Fetching of an instruction from designated storage locations
- Alteration of the contents of designated storage locations
- Alteration of the contents of designated general registers

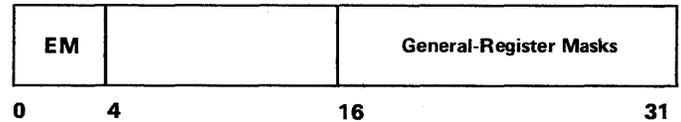
The program can selectively specify one or more of the above PER events to be monitored. The information concerning a PER event is provided to the program by means of a program interruption, with

the cause of the interruption being identified in the interruption code. PER is only available in the EC mode.

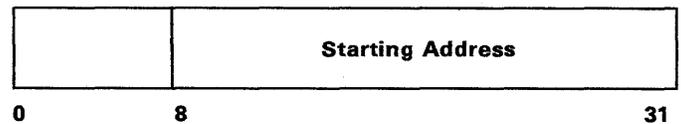
### Control-Register Allocation

The information for controlling program-event recording resides in control registers 9, 10, and 11 and consists of the following fields:

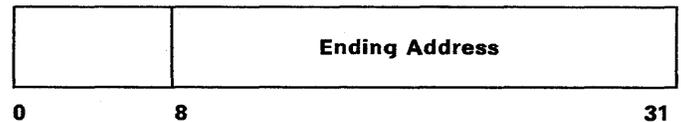
Control Register 9



Control Register 10



Control Register 11



**PER-Event Masks (EM):** Bits 0-3 of control register 9 specify which events are monitored. The bits are assigned as follows:

- Bit 0: Successful-branching event
- Bit 1: Instruction-fetching event
- Bit 2: Storage-alteration event
- Bit 3: General-register-alteration event

Bits 0-3, when ones, specify that the corresponding events are monitored. When the bit is zero, the event is not monitored.

**PER General-Register Masks:** Bits 16-31 of control register 9 specify which general registers are monitored for alteration of their contents. The 16 bits, in the order of ascending bit numbers, correspond one for one with the 16 registers, in the order of ascending addresses. When a bit is one, the associated register is monitored for alteration; if zero, the register is not monitored.

**PER Starting Address:** Bits 8-31 of control register 10 form an address that designates the beginning of the monitored storage area.

**PER Ending Address:** Bits 8-31 of control register 11 form an address that designates the end of the monitored storage area.

**Programming Note**

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 program-event-recording facility, programs that do not use program-event recording should disable program-event recording by setting the PER mask in the EC-mode PSW to zero. No degradation due to program-event recording occurs in the BC mode or when the PER mask in the EC-mode PSW is zero. Disabling of program-event recording in the EC mode by means of the masks in control register 9 does not necessarily prevent performance degradation due to the facility.

**Operation**

Program-event recording (PER) is under control of bit 1 of the EC-mode PSW, the PER mask. When the mask is zero, no PER event can cause an interruption. When the mask is one, a monitored event, as specified by the contents of control registers 9, 10, and 11, causes a program interruption. In BC mode, program-event recording is disabled.

An interruption due to a PER event is taken 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 either completed, terminated, suppressed, or nullified.

When the CPU is disabled for a particular PER event at the time it occurs, either by the 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 program-event recording starting with the execution of the immediately following instruction. If the CPU is enabled for some PER event but an instruction causes the CPU to be disabled for that particular event, the event causes a PER condition to be recognized if it occurs during the execution of the instruction.

When LOAD PSW or SUPERVISOR CALL causes a PER condition and at the same time changes CPU

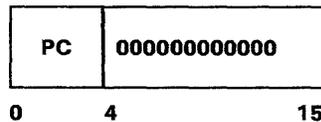
operation from the EC mode to the BC mode, the PER interruption is taken with the old PSW specifying the BC mode and with the interruption code stored in the old PSW. The additional information identifying the PER condition is stored in its regular format at locations 150-155.

Program-event recording applies to emulation instructions in the following way. Emulation instructions indicate all events that have occurred and may additionally indicate events that did not occur and were not called for in the instruction, provided monitoring was enabled for the type of event by the PER mask in the PSW and the PER-event masks, bits 0-3 in control register 9. In such cases, the contents of the remaining positions in control registers 9, 10, and 11 may be ignored. Thus, for example, an emulation instruction may cause general-register alteration to be indicated even though no general registers are altered and even though bits 16-31 of control register 9 are all zeros.

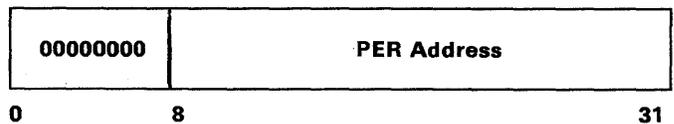
**Identification of Cause**

A program interruption for PER is identified by setting bit 8 of the interruption code to one and by the information placed in storage locations 150-155. The format of the information stored at locations 150-155 is as follows:

Locations 150-151



Locations 152-155



The event causing a PER interruption is identified by a one in bit positions 0-3 of location 150, the PER code (PC), with the rest of the bits in the code set to zeros. The bit position in the PER code for a particular event is the same as the bit position for that event in the PER event-mask field.

The PER address at locations 153-155 is the address of the instruction causing the event. When the instruction is executed by means of EXECUTE, the address of the location containing the EXECUTE instruction is placed in the PER-address field. In

either case, the address of the instruction to be executed next is placed in the PSW. Zeros are stored in bit positions 4-7 of location 150 and at locations 151 and 152.

### Priority of Indication

When a PER interruption occurs and more than one designated PER 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 a program interruption indicates both the PER condition and the other condition.

Except as listed below, a PER event does not cause premature interruption of the interruptible instruction, and the PER condition is held pending until the completion of the instruction.

- When the execution of an interruptible instruction is due to be interrupted by an I/O, external, or repressible machine-check condition, an interruption for a pending PER condition occurs first, and the I/O, external, or machine-check interruption is subsequently subject to the control of mask bits in the new PSW.
- Similarly, when the CPU is placed in the stopped state during the execution of an interruptible instruction, an interruption for a pending PER condition occurs before the stopped state is entered.
- When any program exception is encountered, the pending PER condition is indicated concurrently.
- Depending on the model, in certain situations, a PER condition may cause the execution of an interruptible instruction to be interrupted without an associated asynchronous condition or program exception.

In the case of an instruction-fetching event for SUPERVISOR CALL, the PER interruption occurs immediately after the supervisor-call interruption.

### Programming Notes

1. In the following cases, an instruction can both cause a PER interruption and change the value of bits controlling the occurrence of a PER interruption for that particular event. In these cases the original values of the control bits determine whether a PER interruption occurs.
  - a. The instructions LOAD PSW, 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 storage alteration to be indicated. The old program PSW associated with the PER interruption may indicate that the CPU was disabled for the interruption.

- b. The instruction LOAD CONTROL may cause an instruction-fetching event and change the value of the PER-event masks in control register 9 or the addresses in control registers 10 and 11 controlling indication of the instruction-fetching event.
2. No instructions can both change the values of general-register-alteration masks and cause a general-register-alteration event to be recognized.
  3. When a PER interruption occurs during the execution of an interruptible instruction, the ILC indicates the length of that instruction or EXECUTE, as appropriate. When a PER interruption occurs as a result of LOAD PSW or SUPERVISOR CALL, the ILC indicates the length of these instructions or EXECUTE, as appropriate, unless a concurrent specification exception on LOAD PSW calls for an ILC of 0.
  4. When a PER interruption is caused by branching, the PER address identifies the branch instruction (or EXECUTE, 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 of the PER events—instruction fetching and storage alteration—involve the designation of an area in storage. The storage area monitored for the references 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.

The set of locations designated for monitoring purposes wraps around at location 16777215; that is, location 0 is considered to follow location 16777215. When the starting address is smaller than the ending address, the area is contiguous. When the starting address is larger than the ending address, the set of locations designated for monitoring purposes includes the area from the starting address to location 16777215 and the area from location 0 to, and including, the ending address. When the starting

address is equal to the ending address, only the location designated by that address is monitored.

The monitoring of storage alteration and instruction fetching is performed by comparing all 24 bits of the monitored address with the starting and ending addresses.

## **PER Events**

### **Successful Branching**

Execution of a successful branch operation causes a program-event interruption if bit 0 of the PER-event-mask field is one and the PER mask in the PSW is one.

A successful branch occurs whenever one of the following instructions causes control to be passed to the instruction designated by the branch address:

BRANCH ON CONDITION  
BRANCH AND LINK  
BRANCH ON COUNT  
BRANCH ON INDEX HIGH  
BRANCH ON INDEX LOW OR EQUAL

The branch event is also indicated by an emulation instruction when the emulation instruction itself causes a branch. That is, the branch event is indicated when the location of the next instruction executed by the CPU after leaving emulation mode does not immediately follow the location of the emulation instruction.

The event is indicated by setting bit 0 of the PER code to one.

### **Instruction Fetching**

Fetching the first byte of an instruction from the storage area designated by the contents of control registers 10 and 11 causes a program-event interruption if bit 1 of the PER-event-mask field is one and the PER mask in the PSW is one.

A PER event for instruction fetching is recognized whenever the CPU executes an instruction whose initial byte is located within the monitored area. When the instruction is executed by means of EXECUTE, a PER event is recognized when the first byte of the EXECUTE instruction or the target instruction or both is located in the monitored area.

The event is indicated by setting bit 1 of the PER code to one.

### **Storage Alteration**

Storing of data by the CPU in the storage area designated by the contents of control registers 10 and 11 causes a program-event interruption if bit 2 of the PER-event-mask field is one and the PER mask in the PSW is one.

The contents of storage are considered to have been altered whenever the CPU executes an instruction that causes a whole operand or part of it to be stored within the monitored area of storage. Alteration is considered to take place whenever storing is considered to take place for purposes of indicating protection exceptions. (See the section "Recognition of Access Exceptions" in Chapter 6, "Interruptions.") Storing constitutes alteration for program-event-recording purposes even if the value stored is the same as the original value

Implied locations that are referred to by the CPU in the process of interval-timer updating, interruptions, and execution of I/O instructions, including the interval-timer, PSW, and CSW locations, are not monitored. These locations, however, are monitored when information is stored there explicitly by an instruction. Similarly, monitoring does not apply to storing of data by a channel.

The instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP are considered to alter the second-operand location only when storing actually occurs.

The instruction STORE CHARACTERS UNDER MASK is not considered to alter the storage location when the mask is zero.

The event is indicated by setting bit 2 of the PER code to one.

### **General-Register Alteration**

Alteration of the contents of a general register causes a program-event interruption if bit 3 of the PER-event-mask field is one, the alteration mask corresponding to that general register is one, and the PER mask in the PSW is one.

The contents of a general register are considered to have been altered whenever a new value is placed in the register. Recognition of the event is not contingent on the new value being different from the previous one. The execution of an RR-format arithmetic or movement instruction is considered to fetch the contents of the register, perform the indicated operation, if any, and then replace the value in the register. The register can be designated implicitly, such as in TRANSLATE AND TEST and EDIT AND MARK, or explicitly by an RR, RX, or

RS instruction, including BRANCH AND LINK, BRANCH ON COUNT, BRANCH ON INDEX HIGH, and BRANCH ON INDEX LOW OR EQUAL.

The instructions EDIT AND MARK and TRANSLATE AND TEST are considered to have altered the contents of general register 1 only when these instructions have caused information to be placed in the register.

The instructions MOVE LONG and COMPARE LOGICAL LONG are always considered to alter the contents of the four registers specifying the two operands, including the cases where the padding byte is used, when both operands have zero length, or when condition code 3 is set for MOVE LONG.

The instruction INSERT CHARACTERS UNDER MASK is not considered to alter the general register when the mask is zero.

The instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP are considered to alter the general register, or general-register pair, designated by  $R_1$ , only when the contents are actually replaced, that is, when the first and second operands are not equal.

The event is indicated by setting bit 3 of the PER code to one.

#### **Programming Note**

The following are some examples of general-register alteration:

1. Register-to-register load instructions are considered to alter the register contents even when both operand addresses designate the same register.
2. Addition or subtraction of zero and multiplication or division by one are considered to constitute alteration.
3. Logical and fixed-point shift operations are considered to alter the register contents even for shift amounts of zero.
4. The branching instructions BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL are considered to alter the first operand even when zero is added to its value.

#### ***Indication of Events Concurrently with Other Interruption Conditions***

The following rules govern the indication of PER events caused by an instruction that has also caused a program exception or the monitor event to be indicated, or that causes a supervisor-call interruption.

1. The indication of an instruction-fetching event does not depend on whether the execution of the instruction was completed, terminated, suppressed, or nullified. The event, however, is not indicated when an access exception prohibits access to the first byte of the instruction. When the first halfword of the instruction is accessible but an access exception applies to the second or third halfword of the instruction, it is unpredictable whether the instruction-fetching event is indicated.
2. When the operation is completed, the event is indicated regardless of whether any program exception or the monitoring event is recognized.
3. Successful branching, storage alteration, and general-register alteration 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, general-register or storage alteration is indicated whenever the event has occurred, and 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.
5. When LOAD PSW 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 immediately after the PSW becomes active, the interruption code identifies both the PER condition and the specification exception. When these instructions introduce a PSW-format error of the type that is recognized as part of the execution of the following instruction, the PSW is stored as the old PSW without the specification exception being recognized.

The indication of PER events concurrently with other program interruption conditions is summarized in the figure "Indication of PER Events."

Exception	Type of Ending	PER Event			
		Branch	Instruction Fetch	Storage Alteration	GR Alteration
Operation	S	—	X <sup>1</sup>	—	—
Privileged operation	S	—	X <sup>1</sup>	—	—
Execute	S	—	X <sup>1</sup>	—	—
Protection					
Instruction	S	—	— <sup>1</sup>	—	—
Operand	S or T	—	X	X+	X+
Addressing					
Instruction	S	—	— <sup>1</sup>	—	—
Operand	S or T	—	X	X+	X+
Specification					
Odd instruction address	S	—	—	—	—
Invalid PSW format	C	—	X	—	—
Other	S	—	X	—	—
Data					
Invalid sign	S	—	X	—	—
Other	T	—	X	X+	X+
Fixed-point overflow	C	—	X	—	X
Fixed-point divide					
Division	S	—	X	—	—
Conversion	C	—	X	—	X
Decimal overflow	C	—	X	X	—
Decimal divide	S	—	X	—	—
Exponent overflow	C	—	X	—	—
Exponent underflow	C	—	X	—	—
Significance	C	—	X	—	—
Floating-point divide	S	—	X	—	—
Special operation	S	—	X	—	—
Page access					
Instruction address	N	—	— <sup>1</sup>	—	—
Operand address	N	—	X	X <sup>2</sup>	X <sup>2</sup>
Page state	S	—	X	—	—
Page transition	S	—	X	—	—
Monitor event	C	—	X	—	—

**Explanation:**

- 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. The instruction address in the old PSW has not been updated.
- 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.
- X The event is indicated with the exception if the event has occurred; that is, the contents of the monitored storage location or general register *were* altered, or an attempt was made to execute an instruction whose first byte is located in the monitored area.
- + A model is permitted, but not required, to indicate the event if the event would have occurred had the operation been completed but did not take place because the execution of the instruction was terminated.
- The event is not indicated.

<sup>1</sup>When an access exception applies to the second or third halfword of the instruction but the first halfword is accessible, it is unpredictable whether the instruction-fetching event is indicated.

<sup>2</sup>This condition may occur in the case of the interruptible instructions when the event is recognized in the unit of operation that is completed and the exception causes the next unit of operation to be suppressed or nullified.

**Indication of PER Events**

## Programming Notes

1. The execution of the interruptible instructions MOVE LONG (MVCL) and COMPARE LOGICAL LONG (CLCL) can cause events for general-register alteration and instruction fetching. Additionally, MVCL can cause the storage-alteration event.

Since the execution of MVCL and CLCL can be interrupted, a program event may 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 two instructions:

  - a. The instruction-fetching event is indicated whenever the instruction is fetched for execution, regardless of whether it is the initial execution or a resumption.
  - b. The general-register-alteration event is indicated on the initial execution and on each resumption and does not depend on whether or not the register actually is changed.
  - c. The storage-alteration event is indicated only when data has been stored in the monitored 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 storage area designates a single byte location, a storage-alteration event can be recognized only once in the execution of MOVE LONG.
2. The following is an outline of the general action a program must take to delete the redundant entries

in the PER data for MOVE LONG and COMPARE LOGICAL LONG 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 MVCL or CLCL.
- b. If both conditions are met, delete instruction-fetching and register-alteration events.
- c. If both conditions are met, and the event is storage alteration, delete the event if the current destination-operand address is within the monitored area and the count for the destination operand is not zero.

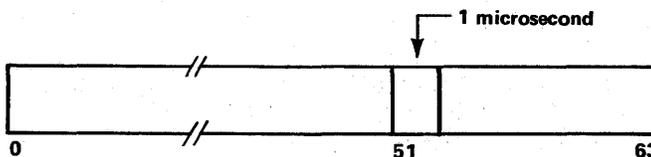
## Timing

### Time-of-Day Clock

The time-of-day (TOD) clock provides a high-resolution measure of real time that is also suitable for the indication of both calendar date and time of day. The cycle of the clock is approximately 143 years.

### Format

The time-of-day clock is a binary counter with the format shown in the following illustration. The bit positions of the clock are numbered 0 to 63, corresponding to the bit positions of a 64-bit unsigned binary integer. Time is measured by incrementing the value of the counter.



In the basic form, the time-of-day clock is incremented at a rate corresponding to adding a one in bit position 51 every microsecond. In models having a higher or lower resolution of the clock, 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 clock is such that the actual incrementing rate is comparable to the instruction-execution rate of the model.

The bit positions of the clock that are always provided are the leftmost positions up to and including the actually incremented bit position. Additional bit positions may be provided to the immediate right of the incremented position; the value they contain is unpredictable. Any remaining bit positions at the extreme right of the doubleword which are not provided are considered to contain zeros.

When incrementing of the clock causes a carry to be propagated out of bit position 0, the carry is ignored, and counting continues. The program is not alerted because no interruption condition is generated as a result of the overflow.

The operation of the clock is not affected or inhibited by any normal activity or event in the system. The clock runs independently of the CPU states, that is, regardless of whether the CPU is in the wait or stopped state, or in the instruction-step or other test mode (test indicator on). Its operation is not affected by resets, initial program loading, or an IML operation. The clock does not run when the CPU power is off.

### States

The following states are distinguished for the time-of-day clock: set, not set, error, and not-operational. The state determines the condition code set by the instruction STORE CLOCK. The clock is incremented, and is said to be running, when it is in either the set or not-set state.

The clock enters the not-set state, and the value of the clock is set to zero, during a power-on reset. The clock is incremented from zero at the completion of power-on reset. When the clock is in the not-set state, STORE CLOCK causes the current value of the running clock to be stored and condition code 1 to be set.

SET CLOCK causes the clock to enter the set state from the not-set, set, or error state. The clock is incremented from the newly set clock value beginning with the first stepping pulse after the clock enters the

set state. When the clock is in the set state, STORE CLOCK causes the current value of the running clock to be stored and condition code 0 to be set.

The clock enters the error state when a malfunction is detected that is likely to have affected the validity of the clock value. A timing-facility-damage machine-check interruption condition is generated whenever the clock enters the error state. When STORE CLOCK is executed with the clock in the error state, condition code 2 is set, and a value of all zeros is stored.

The clock is in the not-operational state when disabled. It depends on the model whether the clock can be placed in this state. A timing-facility-damage machine-check-interruption condition is generated when the clock enters the not-operational state. When the clock is in the not-operational state, STORE CLOCK causes condition code 3 to be set and a value of all zeros to be stored.

### Setting and Inspecting the Value

The clock can be inspected by means of the instruction STORE CLOCK, which causes the current 64-bit clock value to be placed in storage. The execution of STORE CLOCK is interlocked so that successive executions do not provide the same clock value if the clock is running. This unique value may be obtained from additional bits to the right of the bit which is actually incremented; the additional bits are not stored when the clock is in the not-operational state. Zeros are stored for the low-order bits not provided by the clock.

The bit positions which are updated by the clock can be set to a specific value by means of the SET CLOCK instruction. SET CLOCK causes the value of the clock to be changed only when the operator activates the TOD-clock key to permit changing the value of the clock.

### Programming Notes

1. Bit position 31 of the clock is incremented every 1.048576 seconds; hence, for timing applications involving human responses, reference to the high-order clock word may provide sufficient resolution.
2. Communication between systems is facilitated by establishing a standard time origin, or standard epoch, which is the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 AM Greenwich Mean Time (GMT) is recommended as the standard epoch for the clock.

3. 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 run. If the programming support uses the standard epoch, bit 0 of the clock remains one through the years 1972-2041. Ordinarily, testing the high-order bit for a one is sufficient to determine if the clock value is in the standard epoch.

In converting to or from the current date or time, the programming support assumes each day to be 86,400 seconds. It does not take into account "leap seconds" inserted or deleted because of time-correction standards.

4. Because of the limited accuracy of manually setting the clock value, the low-order 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.
5. The following chart shows the time interval between instants at which various bit positions of the time-of-day clock are stepped. This time value may also be considered as the weighted time value that the bit, when one, represents.

TOD-Clock Bit Position	Stepping Interval			
	Days	Hours	Minutes	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
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

6. The following chart shows the clock setting at the start of various years, when the recommended standard epoch is used. The clock settings, expressed in hexadecimal notation, correspond to 0 AM Greenwich Mean Time on January 1 of each year.

Year	Clock Setting (Hex)			
1900	0000	0000	0000	0000
1976	8853	BAF0	B400	0000
1980	8F80	9FD3	2200	0000
1984	96AD	84B5	9000	0000
1988	9DDA	6997	FE00	0000
1992	A507	4E7A	6C00	0000
1996	AC34	335C	DA00	0000
2000	B361	183F	4800	0000

7. The stepping value of time-of-day-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 <sup>1</sup>	72C	E4E2	6E00	0000

<sup>1</sup>Number of days in four years, including a leap year.

## ***Clock Comparator***

The clock comparator provides a means of causing an interruption when the time-of-day clock has passed a value specified by the program.

The clock comparator has the same format as the time-of-day clock. The clock comparator consists of at least bits 0-47. Higher resolution is obtained when more than 48 bits are provided. The bits in positions provided in the clock comparator are compared with the corresponding bits of the time-of-day clock.

When the resolution of the time-of-day 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 STORE CLOCK.

The values of the clock comparator and the time-of-day clock are equal when all compared bit positions are equal. When the values are unequal, the condition is determined by the first unequal pair of bits in a left-to-right comparison of correspondingly numbered bit positions: the zero bit indicates the lesser value and the one bit the greater value.

The clock comparator causes an external interruption with the interruption code 1004 (hex). A request for a clock-comparator interruption exists whenever either of the following conditions occurs:

1. The time-of-day clock is running and the value of the clock comparator is less than the value in the compared portion of the time-of-day clock.
2. The time-of-day clock is in the error or not-operational state.

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 time-of-day clock or when the value of the time-of-day clock is made less than the clock-comparator value. The latter may occur as a result of the time-of-day clock either being set or wrapping to zero.

The clock comparator can be inspected by means of the instruction STORE CLOCK COMPARATOR and can be set to a specific value by means of the SET CLOCK COMPARATOR instruction.

The contents of the clock comparator are initialized to zero by initial program 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 time-of-day clock or as long as the time-of-day clock is in the error or 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 has to be replaced, the time-of-day clock has to be set, or the clock-comparator submask has to be set to zero. Otherwise, loops of external interruptions are formed.
2. The instruction STORE CLOCK 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 time-of-day clock may be incremented one or more times between the instants when instruction execution is begun and when the clock value is accessed. In this situation, the interruption occurs when the execution of STORE CLOCK is completed.

## ***CPU Timer***

The CPU timer provides a means for measuring elapsed CPU time and for causing an interruption when a prespecified amount of time has elapsed.

The CPU timer is a binary counter with a format which is the same as that of the time-of-day clock, except that bit 0 is considered a sign. In the basic form, the CPU timer is decremented at a rate corresponding to 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 reducing the CPU timer is the same as if a one were subtracted in bit position 51 every microsecond.

The CPU timer requests an external interruption with the interruption code 1005 (hex) whenever the value in the CPU timer is negative (bit 0 of the CPU timer is one). The request does not remain pending when the CPU-timer value is made positive.

When both the CPU timer and the time-of-day clock are running, the stepping rates are synchronized such that both are stepped at the same rate.

Normally, decrementing the CPU timer is not affected by concurrent I/O activity. However, the CPU timer may stop during extreme I/O activity and other similar interference situations. In such cases, the time recorded by the CPU timer provides a more

accurate measure of the CPU time used by the program than that which would have been recorded had the CPU timer continued to step.

The CPU timer is decremented when the CPU is executing instructions, during the wait state, and during initial program loading, but it is not decremented when the CPU is in the stopped state. When the manual rate control is set to instruction step, the CPU timer is decremented only during the time when the CPU is actually performing a unit of operation. Depending on the model, the CPU timer may or may not be decremented when the time-of-day clock is in the error or not-operational state or when the CPU is in the check-stop state.

The CPU timer can be inspected by means of the instruction STORE CPU TIMER and can be set to a specific value by means of the SET CPU TIMER instruction.

The contents of the CPU timer are initialized to zero by initial program 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, page faults, and instruction retry. Hence, 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, 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

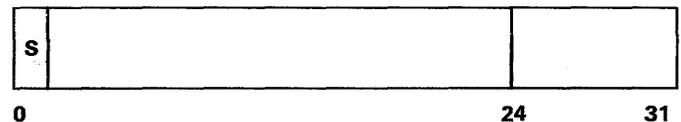
went from positive to negative since the interruption. Subsequently, when the program being timed is to continue, the CPU timer may be set to the saved value by SET CPU TIMER. A CPU-timer interruption will occur immediately after external interruptions are again enabled if the saved value was negative.

The persistence of the CPU-timer-interruption request means, however, that after an external interruption for the CPU timer has occurred, either the value of the CPU timer has to be replaced or the CPU-timer submask has to 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 the instants 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.

#### Interval Timer

The interval timer is a binary counter that occupies a word at storage location 80 and has the following format:



The interval timer is treated as a 32-bit signed binary integer. In the basic form, the contents of the interval timer are reduced by one in bit position 23 every 1/300 of a second. Higher resolution of timing may be obtained in some models by counting with higher frequency in one of the positions 24 through 31. In each case, the frequency is adjusted to cause decrementing in bit position 23 at the rate of 300 times per second. The cycle of the interval timer is approximately 15.5 hours.

The interval timer causes an external interruption, with bit 8 of the interruption code set to one and bits 0-7 set to zeros. Bits 9-15 of the interruption code are zeros unless set to ones for another condition that is concurrently indicated.

A request for an interval-timer interruption is generated whenever the interval-timer value is

decremented from a positive or zero number to a negative number. The request is preserved and remains pending in the CPU until it is cleared by an interval-timer interruption or a program reset. The overflow occurring as the interval-timer value is decremented from a large negative number to a large positive number is ignored.

The interval timer is not necessarily synchronized with the time-of-day clock.

The interval-timer contents are updated at the appropriate frequency whenever other machine activity permits. The updating occurs only between instruction executions, except that the interval timer may be updated during the execution of an interruptible instruction, such as MOVE LONG. An updated interval-timer value is normally available at the end of each instruction execution. When the execution of an instruction or other machine activity causes updating to be delayed by more than one period, the contents of the interval timer may be reduced by more than one unit in a single updating cycle. Interval-timer updating may be omitted when I/O data transmission approaches the limit of storage capability, or when a channel sharing CPU equipment and operating in burst mode causes CPU activity to be locked out. The program is not alerted when omission of updating causes the real-time count to be lost.

The value of the interval timer is accessible by fetching the word at location 80 as an operand, provided the location is not protected against fetching. It may be changed at any time by storing a word at location 80. When location 80 is protected, any attempt by the program to change the value of the interval timer causes a program interruption for protection exception.

When the contents of the interval timer are fetched by a channel or are used as the source of an instruction, the result is unpredictable. Similarly, storing by the channel at location 80 causes the contents of the interval timer to be unpredictable.

The interval timer is not decremented when the interval-timer control is set to disable. The interval timer is also not decremented when the CPU is not in the operating state or when the rate control is set to instruction step.

## Programming Notes

1. The value of the interval timer may be changed without losing the real-time count by storing the new value at locations 84-87 and then shifting bytes 80-87 to locations 76-83 by means of the instruction MOVE (MVC). Thus, in a single operation, the new interval-timer value is placed at location 80, and the old value is made available at location 76.  
If any means other than the instruction MOVE (MVC) are used to interrogate and then replace the value of the interval timer, including MOVE LONG or two separate instructions, the program may lose a time increment when an updating cycle occurs between fetching and storing.
2. When the value of the interval timer is to be recorded on an I/O device, the program should first store the interval-timer value in a temporary storage location to which the I/O operation subsequently refers. When the channel fetches the interval-timer value directly from location 80, the value obtained is unpredictable.

## Externally Initiated Functions

### Resets

Four reset functions are provided:

- Program reset
- Initial program reset
- Clear reset
- Power-on reset

Program reset provides a means of clearing equipment-check indications and any resultant unpredictability in the CPU and I/O state with the least amount of information destroyed. In particular, it is used to clear check conditions when the machine state is to be preserved for analysis or resumption of operation.

Initial program reset provides the functions of program reset together with initialization of the current PSW, CPU timer, clock comparator, and control registers.

Clear reset causes initial program reset to be performed and, additionally, clears or initializes all storage locations and the remaining registers with the exception of the time-of-day clock. Such clearing is useful in debugging programs and in ensuring user privacy. Clearing does not affect external storage, such as direct-access storage devices used by the control program to hold the contents of unaddressable pages.

Power-on reset combines the functions of clear reset with initializing the time-of-day clock and selecting storage size.

Program reset and clear reset are initiated manually using the operator facilities (see Chapter 13, "Operator Facilities"). Initial program reset is part of the initial-program-loading function. Power-on reset is performed as part of turning power on. The reset actions are tabulated in the figure "Summary of Reset Actions."

#### **Program Reset**

Program 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 are cleared.
3. Any pending machine-check-interruption conditions, error indications, and check-stop state are cleared.
4. Any buffers containing prefetched instructions or operands, or results due to be stored, are cleared.
5. The CPU is placed in the stopped state after actions 1-4 have been completed.
6. I/O system reset is performed in each channel.
7. Any ongoing machine-save function is canceled, and partially saved information is invalidated.

Register and storage contents remain unchanged by program reset. However, if a register or storage location is being accessed at the time the program-reset operation is performed, the subsequent contents of the register or location are unpredictable.

Area Affected	Reset Function			
	Program Reset	Initial Program Reset	Clear Reset	Power-On Reset
CPU state	S	S	S	S
Channels	R	R	R	R
PSW	U	C	C	C
CPU timer	U	C	C	C
Clock comparator	U	C	C	C
Time-of-day clock	U <sup>1</sup>	U <sup>1</sup>	U <sup>1</sup>	T
Control registers	U	N	N	N
General registers	U	U	C	C
Floating-point registers	U	U	C	C
Capacity counts	U	U	N	N
Page descriptions	U	U	C	C
Storage	U	U	P	P
Machine save	H	H	I	I

**Explanation:**

C The contents are cleared to zero and validated.

H Any ongoing machine-save function is halted, and any partially altered save information is invalidated.

I The contents are invalidated.

N The contents are initialized.

P The first n storage pages are cleared and made addressable, where n is the lesser of the available-frame-capacity and page-capacity counts. Any remaining pages are left disconnected.

R I/O system reset is performed in the channels, and I/O-interruption conditions are cleared. As part of this reset, system reset is signaled to the I/O control units and devices configured to each channel.

S The CPU is reset, terminating current operations, after which the CPU is in the stopped state.

T The TOD clock is initialized to zero and validated, and it enters the not-set state.

U The contents remain unchanged by the reset.

<sup>1</sup> Access to the TOD clock by means of STORE CLOCK at the time a reset function is performed does not affect the value of the TOD clock.

**Summary of Reset Actions**

As part of the I/O system reset performed in each channel (see the section "I/O System Reset" in Chapter 12, "Input/Output Operations"), pending I/O-interruption conditions are cleared, and system reset is signaled to all control units and devices configured to the channel. 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 publication on 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 CPU signaling the reset.

Program reset is performed when the system-reset-normal key is activated. It is also part of the initial-program-reset function.

### Initial Program Reset

Initial program reset combines the program-reset functions with the following actions:

1. The contents of the current PSW, CPU timer, and clock comparator are set to zero.
2. All assigned control-register positions are set to their initial values.

These clearing and initializing functions include validation.

Clearing the current PSW to zero causes the PSW to assume the BC-mode format. The instruction-length code and interruption code in the PSW are unpredictable, because these values are not retained when a new PSW is introduced.

Initial program reset is part of the clear-reset function. It is also part of the initial-program-loading function when the load-normal or load-clear key is activated.

### Clear Reset

Clear reset combines the initial-program-reset function with an initializing function which causes the following actions:

1. The general and floating-point registers are set to zero.
2. The storage key of every storage page is set to zero.
3. The page bits of every storage page are set to zeros.
4. All page frames that had been made temporarily unavailable by DECONFIGURE PAGE instructions are made available. (This excludes frames made permanently unavailable by maintenance intervention.)
5. The page-capacity, existing-frame-capacity, available-frame-capacity, and free-frame-capacity counts are initialized.
6. Let  $n$  be the lesser of AFCC, the current available-frame-capacity count, and PCC, the page-capacity count. Then each of  $n$  page frames is assigned to one of the first  $n$  storage pages, namely those with page addresses 0 to  $n$  minus one. These pages are cleared to zero bytes and have their page states set to addressable. Any remaining pages have their page states set to disconnected.
7. Any previously saved machine-save information is invalidated.

Validation is included in setting registers and capacity counts and in clearing storage and page descriptions.

Clear reset is performed when the system-reset-clear key is activated. Clear reset is also part of the power-on-reset function, and part of the initial-program-loading function when performed upon activating the load-clear key.

### Programming Notes

1. For the program-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 time-of-day clock, CPU timer, and interval timer, and for the possibility of taking a machine-check interruption, all CPU activity can be quiesced 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 program reset at the time the timing facilities are being updated or a machine-check interruption occurs, the CPU must be in the stopped state.
2. Program reset, initial program reset, and clear reset do not affect the value and state of the time-of-day clock.
3. Clear reset causes all bit positions of the interval timer to be cleared to zeros.
4. Program reset and initial program reset leave machine-save information unchanged if no machine save is being performed at the time of the reset.
5. 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 program reset or initial program reset is executed while the CPU is in the check-stop state, the contents of registers, the CPU state, and the storage location accessed at the time of the error may still be in error after the check-stop state is cleared by these resets. In such a case, a clear reset is required in order to clear the error.

### Power-On Reset

Power-on reset causes the following actions:

1. The clear-reset function is performed.
2. The value of the time-of-day clock is set to zero, and the clock enters the not-set state.

Power-on reset is part of the power-on sequence of the machine. The power-on sequence is not complete until the clear-reset function has been performed successfully and the time-of-day clock has entered the not-set state.

## ***Initial Program Loading***

Initial program loading (IPL) is provided to initiate processing when the contents of storage or of the PSW are not suitable for processing.

Initial program loading is initiated manually by designating an input device with the load-unit-address controls and subsequently activating the load-normal or load-clear key. The load-normal key causes an initial-program-reset operation to be performed, and the load-clear key causes a clear-reset operation to be performed. The CPU enters the load state. Subsequently, a read operation is initiated from the selected input device. The CPU does not necessarily enter the stopped state during the execution of the reset operation. The load indicator is on while the CPU is in the load state.

The read operation is performed as if a START I/O instruction were executed that specified the channel, subchannel, and I/O device designated by the load-unit-address controls. The operation uses an implied channel-address word (CAW) containing a subchannel key of zero, and a channel-command-word (CCW) address of 0, but the CAW location in storage, location 72, is not accessed. The load-unit-address controls provide the 12 rightmost bits of the I/O address; zeros are implied for the leftmost bits.

Although the location of the first CCW to be executed is specified by the CCW address as 0, the first CCW actually executed is an implied CCW, containing, in effect, a read command with the modifier bits set to zeros, a data address of 0, a byte count of 24, the chain-command flag set to one, the SLI flag set to one, the chain-data flag set to zero, the skip flag set to zero, and the PCI flag set to zero. The CCW fetched, as a result of command chaining, from storage 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 CCWs used for the IPL sequence are ignored.

When the I/O device provides channel-end status for the last operation of the IPL chain and no exceptional conditions are detected in the operation, a new PSW is obtained from storage locations 0-7. When this PSW specifies the EC mode, the I/O address that was used for the IPL operation is stored at locations 186-187, and zeros are stored at location 185; when the BC mode is specified, the I/O address is stored at locations 2-3. The CPU leaves the load state and enters the operating state, with CPU operation proceeding under the control of the new

PSW, provided the rate control is set to process; if the rate control is set to instruction step, the CPU enters the stopped state after the new PSW has been obtained.

When channel-end status for the IPL operation is presented, either separate from or along with device-end status, no I/O-interruption condition is generated. Similarly, any PCI flags specified by the program in the CCWs used for the IPL sequence are ignored. If the device-end status for the IPL operation is provided separately after channel-end status, it causes an I/O interruption condition to be generated.

If the IPL I/O operation or the PSW loading is not completed satisfactorily, the CPU idles in the load state, and the load indicator remains on. This occurs when the device designated by the load-unit-address controls is not operational, when the device or channel signals any condition other than channel end, device end, or status modifier during or at the completion of the IPL I/O operation, or when the PSW loaded from location 0 has a PSW-format error that is recognized during the loading procedure. The address of the I/O device used in the IPL operation is not stored. The contents of storage locations 0-7 are unpredictable, but the contents of other storage locations remain unchanged.

When fewer than eight bytes are read into locations 0-7, the PSW fetched from location 0 at the conclusion of the IPL operation is unpredictable.

### **Programming Notes**

1. The information read and placed at locations 8-15 and 16-23 may be used as CCWs for reading additional information during the IPL sequence: 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.

The status-modifier bit has its normal effect during the IPL operation, causing the channel 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.

The PSW that is loaded at the completion of the IPL procedure may be provided by the first eight bytes of the IPL I/O operation or may be placed at locations 0-7 by a subsequent CCW.

2. When the PSW in location 0 has bit 14 set to one, the CPU is placed in the wait state after the IPL procedure is completed; at that point, the load and manual indicators are off, and the wait indicator is on.
3. Activating the load-normal key permits an IPL program to be loaded with a minimum disturbance of storage contents. This function may be useful in debugging. When the power is turned on or 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.

### **Machine Save**

The machine-save operation saves the current CPU status and the status and contents of storage page 0 for subsequent retrieval by programming. The operation is initiated manually by the machine-save key (see Chapter 13, "Operator Facilities").

Machine save causes the following actions:

1. The current contents of all CPU registers and the status of page 0 are saved in internal storage. The format of the saved information is not defined. The figure "Machine Status, Retrieval Format" describes the machine-status information in the 256-byte format in which it is moved to addressable storage by a subsequent RETRIEVE STATUS AND PAGE instruction.
2. The current contents of page 0, that is, the 2,048 bytes at addresses 0-2047, are saved in internal storage.

The register contents and the status and contents of page 0 remain unchanged.

When a machine-save operation has been successfully completed, the save indicator is turned on.

<u>Byte Offset</u>	<u>Bits</u>	<u>Contents</u>
0-7		CPU timer <sup>1</sup>
8-15		Clock comparator <sup>1</sup>
16-23		Program-status word
24-31		Time-of-day clock <sup>1</sup>
32-63		Floating-point registers 0, 2, 4, 6
64-127		General registers 0-15
128-191		Control registers 0-15
192-199		CPU ID <sup>1</sup>
200-203		Page-capacity count <sup>2</sup>
204-207		Existing-frame-capacity count <sup>2</sup>
208-211		Available-frame-capacity count <sup>2</sup>
212-215		Free-frame-capacity count <sup>2</sup>
216	0	Zero
	1-3	Page bits of page 0
	4	Zero
	5	Reference bit of page 0
	6	Change bit of page 0
	7	Zero
217	0-3	Access-control bits of page 0
	4	Fetch-protection bit of page 0
	5-7	Zeros
218-219		Frame index <sup>2</sup> of page 0
220-255		Zeros

#### **Explanation:**

<sup>1</sup>The formats of these fields are the same as those produced by STORE CPU TIMER, STORE CLOCK COMPARATOR, STORE CLOCK, and STORE CPU ID, respectively.

<sup>2</sup>The capacity counts and the frame index are right-aligned with leftmost bits of zeros.

#### **Machine Status, Retrieval Format**

A machine save replaces the information saved by the previous machine save.

When a clear-reset operation is performed, any previously saved information becomes invalid. Subsequent execution of the RETRIEVE STATUS AND PAGE instruction returns condition code 3 until another machine-save operation is successfully performed.

A reset operation occurring while a machine save is in progress halts the machine-save operation. If an incomplete machine save partially alters previously saved information, the saved information is indicated to be invalid, and subsequent execution of RETRIEVE STATUS AND PAGE returns condition code 3 until the next successful machine-save operation. Invalid machine saves cannot be retrieved.

The CPU must be in the stopped state before a machine-save operation can be initiated. If an error is encountered during the operation, the saved information becomes invalid, the CPU enters the check-stop state, and the save indicator is not turned on.

#### **Programming Notes**

1. Machine save may be used as part of a machine-dump procedure when the normal supervisor program is not functioning properly, such as after a hard wait (wait state with interruptions disabled). By preserving the complete machine status and page 0, machine save permits loading a dump program, which can preserve additional pages if necessary. The dump program can then merge the saved information with the undisturbed pages to create a complete image of the machine at the time of the machine save. The machine should not be cleared before loading the dump program.
2. When the supervisor program is still functioning, it is less disruptive to use the supervisor to invoke a dump program without a machine save. An intermediate option is the restart function.
3. The format of the byte at offset 216 corresponds to the byte inserted by the instruction INSERT PAGE BITS.
4. Unassigned bits in the retrieval format of the machine status are stored as zeros. The program should not depend on such zeros, however, to ensure that existing programs run if new facilities using these bits are defined.

## Chapter 5. Program Execution

### Contents

Instructions	5-1	Page-Description Accesses	5-6
Operands	5-1	Storage-Operand References	5-7
Instruction Format	5-2	Storage-Operand Fetch References	5-7
Register Operands	5-3	Storage-Operand Store References	5-7
Immediate Operands	5-3	Storage-Operand Update References	5-7
Storage Operands	5-3	Storage-Operand Consistency	5-8
Operand-Address Generation	5-4	Relation Between Operand Accesses	5-8
Instruction Execution	5-4	Other Storage References	5-9
Sequential Instruction Execution	5-4	Serialization	5-9
Branching	5-4	CPU Serialization	5-9
Interruptions	5-5	Channel Serialization	5-10
Sequence of Storage References	5-5		
Instruction Fetch	5-6		

Normally, operation of the CPU is controlled by instructions taken in sequence. A change in the sequential operation may be caused by branching, LOAD PSW, interruptions, or manual intervention.

### Instructions

Each instruction consists of two major parts:

1. An operation code (op code), which specifies the operation to be performed, and
2. 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, 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 eight-bit field containing the immediate operand is called the I field.

Operands in storage may either have an implied length, be specified by a bit mask, or, in other cases, be specified by a four-bit or eight-bit length specification, called the L field, in the instruction.

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 independently 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, designated by the B, X, and D fields, respectively, in the instruction.

For purposes of describing 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. An exception is instructions with "store" in the instruction name, other than STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK, where the result replaces the second operand. Except for storing the final result, the contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

## Instruction Format

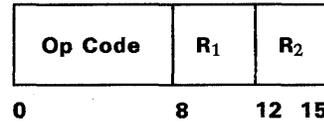
An instruction is one, two, or three halfwords in length and must be located in storage on a halfword integral boundary. Each instruction is in one of six basic formats: RR, RX, RS, SI, S, and SS, with two variations of SS. (See the figure "Basic Instruction Formats.")

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.

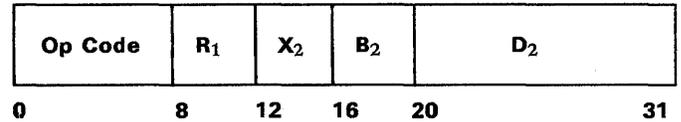
The format names indicate, in general terms, the classes of operands which participate in the operation:

- RR denotes a register-and-register operation.
- RX denotes a register-and-indexed-storage operation.
- RS denotes a register-and-storage operation.
- SI denotes a storage-and-immediate operation.
- S denotes an operation using an implied operand and storage.
- SS denotes a storage-and-storage operation.

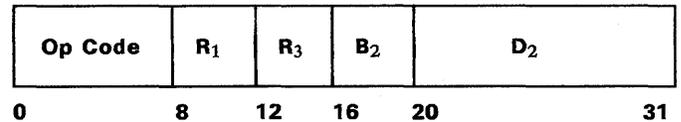
### RR Format



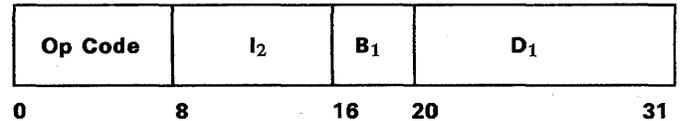
### RX Format



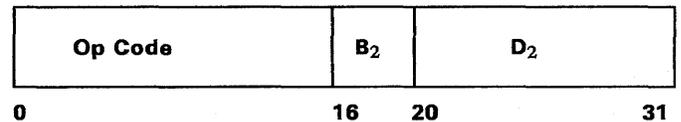
### RS Format



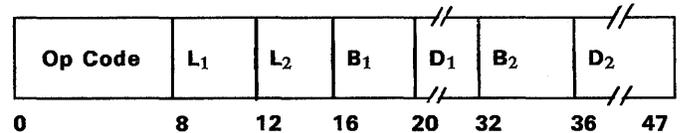
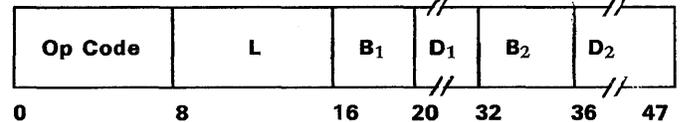
### SI Format



### S Format



### SS Format



Basic Instruction Formats

The first byte or, in the S format, the first two bytes of an instruction contain the op code. For some instructions in the S format, all or a portion of the second byte is ignored.

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	Instruction Format
00	One halfword	RR
01	Two halfwords	RX
10	Two halfwords	RS/SI/S
11	Three halfwords	SS

In the format illustration for each individual instruction description, the op-code field shows the op code in hexadecimal representation. The hexadecimal representation uses one graphic for a four-bit code, and therefore two graphics for an eight-bit byte. The graphics 0-9 are used for the codes 0000-1001; the graphics A-F are used for codes 1010-1111.

The remaining fields in the format illustration for each instruction are designated by code names, consisting of a letter and possibly a subscript number. The subscript number denotes the operand to which the field applies.

### Register Operands

In the RR, RX, and RS formats, the contents of the register designated by the  $R_1$  field are called the first operand. In the RR format, the  $R_2$  field designates the register containing the second operand, and the same register may be designated for the first and second operand. In the RS format, the use of the  $R_3$  field depends on the instruction.

The R field designates a general register in the general instructions and a floating-point register in the floating-point instructions. In the instructions LOAD CONTROL and STORE CONTROL the R field designates a control register.

Unless otherwise indicated in the individual instruction description, the register operand is one register in length (32 bits for a general register or a control register and 64 bits for a floating-point register), and the second operand is the same length as the first.

### Immediate Operands

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 designate the first operand, which is one byte in length.

### Storage Operands

In the SI and 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 S, RS, and SS formats, the contents of the general register designated by the  $B_2$  field are added to the contents of the  $D_2$  field to form the second-operand address. In the RX format, 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 to form the second-operand address.

In the SS format, with two length fields given,  $L_1$  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 a single, eight-bit length field, 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, except for the following instructions: EDIT, EDIT AND MARK, TRANSLATE, and TRANSLATE AND TEST. RETRIEVE STATUS AND PAGE does not use the L field, the operand lengths being fixed.

## ***Operand-Address Generation***

An operand address that refers to storage 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 is a 24-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 specify 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 purposes.

The index is a 24-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 instruction format. The RX format instructions permit double indexing; that is, the index can be used to provide the address of an element within an array.

The displacement is a 12-bit number contained in a field, called the D field, in the instruction. The displacement provides for relative addressing of up to 4,095 bytes beyond the location designated by the base address. 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.

In forming the address, the base address and index are treated as 24-bit unsigned binary integers. The displacement is similarly treated as a 12-bit unsigned binary integer, and 12 high-order zeros are appended. The three are added as 24-bit binary numbers, ignoring overflow. The sum is always 24 bits long. The bits of the generated address are numbered 8-31, corresponding to the numbering of the base-address and index bits in the general register.

A zero in any of the B<sub>1</sub>, B<sub>2</sub>, or X<sub>2</sub> fields indicates the absence of the corresponding address component. For the absent component, a zero is used in forming the address, regardless of the contents of general register 0. A displacement of zero has no special significance.

An instruction can designate the same general register both for address computation and as the location of an operand. Address computation is completed prior to the execution of the operation.

Unless otherwise indicated in the individual instruction definition, the computed operand address designates an operand in storage. When a storage operand is designated, the address designates the leftmost byte of the operand. For branching instructions, the second-operand address is used as the branch address. For shifting instructions, the second-operand address is not used as an address but specifies the shift amount.

## **Instruction Execution**

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 machine in relation to the program currently being executed. The active or controlling PSW is called the current PSW.

### ***Sequential Instruction Execution***

In program execution, instructions are normally executed sequentially, one at a time, left to right in storage. An instruction is fetched from the location designated by the instruction address in the current PSW. The instruction address is then increased by the number of bytes in the instruction in order to address the next instruction in sequence. The instruction is then executed, and the same steps are repeated using the new value of the instruction address.

### ***Branching***

The normal sequential execution of instructions may be changed by the use of the branching instructions in order to perform subroutine linkage, decision-making, and loop control. A branch instruction affects instruction sequencing by introducing a new instruction address into the current PSW.

Subroutine linkage is provided by the BRANCH AND LINK instructions, which permit not only the introduction of a new instruction address but also the preservation of the return address and associated information.

Facilities for decision making are provided by the BRANCH ON CONDITION instruction. This instruction inspects a two-bit condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. Each of these operations can set the code in any one of four states, and the instruction BRANCH ON CONDITION can specify any selection of these four states as the criterion for

branching. For example, the condition code reflects such conditions as nonzero, first operand high, equal, overflow, channel busy, and zero. Once set, the condition code remains unchanged until modified by an instruction that causes a different condition code to be set.

The two bits of the condition code provide 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.

Loop control can be performed by the use of **BRANCH ON CONDITION** to test the outcome of address arithmetic and counting operations. For some particularly frequent combinations of arithmetic and tests, the instructions **BRANCH ON COUNT**, **BRANCH ON INDEX HIGH**, and **BRANCH ON INDEX LOW** are provided. These branches, being specialized, provide increased performance for these tasks.

### ***Interruptions***

Interruptions permit the CPU to change state as a result of conditions external to the system, in input/output (I/O) devices, or in the CPU itself. Six classes of interruption conditions are possible: machine check, supervisor call, program, external, I/O, and restart.

Each class has two related PSWs, called old and new, in permanently assigned storage locations. In all classes, an interruption involves storing information identifying the cause of the interruption, storing the current PSW in its old position, and making the PSW at the new position the current PSW.

The old PSW holds all necessary CPU status information existing at the time of the interruption. If, at the conclusion of the program invoked by the interruption, there is an instruction to make the old PSW the current PSW, the CPU is restored to the state prior to the interruption, and the interrupted program continues.

### **Sequence of Storage References**

Conceptually, the CPU processes instructions one at a time, with the execution of one instruction preceding the execution of the following instruction. The execution of the instruction specified 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 appears to the program to be performed sequentially, with one instruction being fetched after the preceding operation is completed and before the execution of the current operation is begun, even though, as observed by the CPU itself, 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 to modify an instruction in storage by the immediately preceding instruction.

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 are those that would have been obtained if the operations had been performed in the conceptual sequence. However, channels may, unless otherwise constrained, observe a sequence that differs from the conceptual sequence.

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 of 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, a channel may be able to observe intermediate, or partially completed, results.

When the program on the CPU interacts with a program on a channel, the programs may have to take into consideration that a single operation may consist of a series of storage references, that a storage reference may in turn consist of a series of accesses, and that the conceptual and actual sequences of these accesses may differ. Storage references associated with instruction execution are of the following types: instruction fetches and storage-operand references. For the purposes of the

following discussion, page-description accesses are also considered to be storage references.

### ***Instruction Fetch***

Instruction fetching consists in fetching the one, two, or three halfwords specified 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 specifies 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 EXECUTE 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 for 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.

There is no limit established as to the number of instructions which may be prefetched, and multiple copies may be fetched of the contents of a single storage location. As a result, the instruction executed is not necessarily the most recently fetched copy. Storing caused by channels does not necessarily change the copy of prefetched instructions. However, if a store that is conceptually earlier occurs on the CPU and modifies the location from which the instruction is subsequently fetched, the updated information is obtained.

All copies of prefetched instructions are discarded by a serializing operation and as the CPU enters the operating state.

### **Programming Note**

When a channel modifies an instruction, it is possible for the CPU to recognize the changes to some but not all modified bit positions of the instruction.

### ***Page-Description Accesses***

References to the page description are handled as follows:

1. Whenever a reference to storage is made and protection checking applies to the reference, the four access-control bits and the fetch-protection bit of the storage key associated with the storage location must appear to be inspected concurrently with the reference to the storage location.
2. When storing is performed, the change bit is set in the associated storage key concurrently with the store operation.
3. The instruction SET STORAGE KEY causes the four access-control bits, the fetch-protection bit, and the change bit to be set concurrently. The instruction also modifies the reference bit. The access to the access-control bits, the fetch-protection bit, and the change bit for SET STORAGE KEY follows the sequence rules for storage-operand store references, and is a single-access reference.
4. The instruction INSERT STORAGE KEY inspects but does not modify the storage key.
5. The instruction SET PAGE BITS provides a consistent image of the change bit. The instruction modifies both the reference and change bits, and the three programmable page bits. The page bits are only accessible by the CPU. The access to the change bit follows the sequence rules for storage-operand update references, with the following exception: if the change bit is being set to zero, no storing in the associated storage page by a channel is permitted between the fetching of the change bit and the setting of the change bit to zero.
6. The instruction INSERT PAGE BITS inspects but does not modify the reference, change, and page bits. The page bits are only accessible by the CPU.
7. The instruction RESET REFERENCE BIT modifies only the reference bit. The access to the storage key for RESET REFERENCE BIT follows the sequence rules for storage-operand update references. All bits of the storage key other than the reference bit remain unchanged.
8. Whenever a reference to storage is made and page-state checking applies to the reference, the page state and frame index associated with the storage location must appear to be inspected concurrently with the reference to the storage location.

9. The instruction **CONNECT PAGE** causes the page state and frame index to be set concurrently in the page description, with the access to the page state and frame index following the sequence rules for storage-operand store references.
10. During the execution of the instructions **DECONFIGURE PAGE** and **DISCONNECT PAGE**, the accesses to set the reference bit and the change bit to zeros occur concurrently with or after the access to set the page state to disconnected.
11. The instructions **MAKE ADDRESSABLE** and **MAKE UNADDRESSABLE** modify only the page state.
12. The instruction **LOAD FRAME INDEX** inspects but does not modify the page state and frame index. The page state and frame index may only be modified explicitly by other instructions.

The record of references provided by the reference bit is not necessarily accurate, and the handling of the reference bit is not subject to the concurrency rules. However, in the majority of situations, reference recording approximately coincides with the storage reference.

In certain situations, the change bit may be set when no storing has actually taken place.

### ***Storage-Operand References***

A storage-operand reference is the fetching or storing of the explicit operand or operands in the storage locations specified by the instruction.

During the execution of an instruction, all, or a portion, 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 channels 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 reference to the location is called a storage-operand fetch reference. A fetch reference is identified in individual instruction definitions by indicating that the access exception is for fetch.

All bits within a single byte of a fetch reference are accessed concurrently. When an operand consists of

more than one byte, the bytes may be fetched piecemeal a byte at a time from storage. Unless otherwise specified, the bytes are not necessarily fetched in any particular order.

### ***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 reference to the location is called a storage-operand store reference. A store reference is identified in individual instruction definitions by indicating that the access exception is for store.

All bits within a single byte of a store reference are accessed concurrently. When an operand consists of more than one byte, the bytes may be stored piecemeal a byte at a time into storage. Unless otherwise specified, the bytes are not necessarily stored in any particular order.

The CPU may delay storing results into 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 order 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 channels. The results of any one instruction are stored in the order specified for that instruction.

The CPU does not fetch operands from a storage location until all information destined for that location by the CPU has been stored. Prefetched instructions may seem 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 of a fetch and subsequently of a store. The combination of the two accesses is referred to as an update reference. Instructions such as **MOVE ZONES**, **TRANSLATE**, **OR (OI)**, and **ADD DECIMAL** cause an update to the first-operand location. No special interlock is provided between the fetch and store, and accesses by channels are permitted. An update reference is identified in the individual instruction definition by indicating that the

access exception is for both fetch and store. The fetch and store accesses associated with an update reference are not necessarily contiguous, and it is possible for a channel to make one or more interleaved accesses to the same location. The interleaved accesses can be either fetches or stores.

### ***Storage-Operand Consistency***

A fetch reference is said to be a single-access reference if the a 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 following cases, storage-operand references are single-access references.

1. Instructions which operate on decimal data. The storage references associated with the following instructions are not necessarily single-access references: the decimal instructions and the instructions CONVERT TO BINARY, CONVERT TO DECIMAL, MOVE WITH OFFSET, PACK, and UNPACK.
2. Page-access exceptions.
  - a. When a storage operand in which there is storing crosses a page boundary and a page-access exception is recognized, the storage-operand store references to the part of the operand which does not cause the page-access exception are not necessarily single-access references.
  - b. In an instruction involving two storage operands (for example, an SS-format instruction or MOVE LONG) for which there is storing in a storage operand, the storage-operand store references for one operand are not necessarily single-access references when a page-access exception is recognized for the other operand.

When a storage-operand store reference to a location is not a single-access reference, the contents placed at a byte location are not necessarily the same for each store access; thus, intermediate results in a single-byte location may be observed by channels.

### **Programming Notes**

1. When multiple fetch accesses are made to a single byte that is being changed by a channel, the result is not necessarily limited to that which could be obtained by fetching the bits individually. For example, the process used in MULTIPLY DECIMAL may consist of repetitive additions and subtractions each of which causes the second operand to be fetched from storage.
2. When CPU instructions are used to modify storage locations being accessed by a channel simultaneously, multiple store accesses to a single byte by the CPU may result in intermediate values being observed by a channel. To avoid these intermediate values (especially when modifying a CCW chain), only instructions making single-access references should be used. Either one storage page should be operated on at a time or preliminary testing should be performed to ensure that all required pages are addressable. The instructions which operate on decimal data should not be used.

### ***Relation Between Operand Accesses***

Storage-operand fetches associated with one instruction execution must appear to precede all storage-operand references for conceptually subsequent instructions. A storage-operand store specified by one instruction must appear 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 must 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.

### **Programming Note**

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 a channel 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 to itself may yield a result other than zero.

### **Other Storage References**

Updating of the interval timer occurs after storage-operand references for the conceptually previous instruction and before storage-operand references for the conceptually subsequent instruction. Interval-timer updates can also occur within an interruptible instruction between units of operation.

The interruption codes not stored within the old PSW are not necessarily single-access stores. The external and SVC interruption-code stores occur between the conceptually previous and conceptually subsequent operations. The program interruption-code store accesses may precede the storage-operand references associated with the instruction which results in the program interruption.

The CSW and I/O-communications-area stores occur within the conceptual limits of the interruption or I/O instruction with which they are associated.

### **Serialization**

The order of functions performed by a CPU is normally independent of the functions performed by channels. Similarly, the order of functions performed by a channel is normally independent of the functions performed by other channels and by the CPU. However, at certain points in its execution, serialization of the CPU occurs. Serialization also occurs at certain points for channels.

### **CPU Serialization**

All interruptions and the execution of certain instructions cause serialization of CPU operation. A serialization operation consists in completing all conceptually prior storage accesses by the CPU, as observed by channels, before the conceptually following storage accesses occur. Serialization affects the order of all CPU accesses to storage and to the page descriptions.

Serialization is performed by all interruptions and by the execution of the following instructions:

1. The general instructions BRANCH ON CONDITION (BCR) with the R<sub>1</sub> and R<sub>2</sub> field containing 15 and zero, respectively, and COMPARE AND SWAP, COMPARE DOUBLE AND SWAP, STORE CLOCK, SUPERVISOR CALL, and TEST AND SET.
2. LOAD PSW.
3. All I/O instructions.

The sequence of events associated with a serializing operation is as follows:

- All conceptually previous storage accesses by the CPU are completed, as observed by channels. This includes all conceptually previous stores and changes to page descriptions.
- 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, 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 the 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.
- Finally, instruction fetch and operand accesses for conceptually subsequent operations may begin.

A serializing function affects the order of storage accesses that are under the control of the CPU. It does not affect the order of storage accesses under the control of a channel.

## **Programming Notes**

1. When a serializing operation takes place, channels observe instruction and operand fetching and result storing to take place in the order established by the serializing operation.  
Storing by a channel into a location from which a serializing instruction is fetched does not necessarily affect the execution of the serializing instruction unless a serializing operation has been performed after the storing and before the execution of the serializing instruction.
2. For programs that are intended to run also on multiprocessing configurations of System/370, it should be noted that the serializing operations affect the order of CPU accesses to storage and to the storage key, as observed by other CPUs as well as by channels. Therefore, serializing instructions should be inserted wherever it is necessary to control the interaction of programs that may run concurrently on different CPUs.

## ***Channel Serialization***

Serialization of a channel occurs as follows:

1. For a single channel program, all storage accesses and page-description accesses by the channel follow the execution of START I/O or START I/O FAST RELEASE, as observed by the CPU and other channels. This includes all accesses for the CAW, CCWs, and data.
2. For the last CCW of a chain, all storage accesses and page-description accesses are completed, as observed by the CPU and other channels, before the interruption condition indicating channel end is presented to the CPU.
3. If a CCW in the chain contains a PCI bit which is one, all storage accesses and page-description accesses due to CCWs preceding it in the chain are completed, as observed by the CPU and other channels, before the PCI condition is presented to the CPU.

The serialization of a channel does not affect the order of storage accesses or page-description accesses caused by a program in the CPU or another channel. It also does not affect the order of storage accesses or page-description accesses caused by other channel programs on the same channel.

## Chapter 6. Interruptions

### Contents

Interruption Action	6-1	Floating-Point-Divide Exception	6-11
Source Identification	6-2	Monitor Event	6-11
Enabling and Disabling	6-4	Operation Exception	6-11
Instruction-Length Code	6-4	Page-Access Exception	6-11
Zero ILC	6-5	Page-State Exception	6-12
ILC on Instruction-Fetch Exceptions	6-5	Page-Transition Exception	6-12
Exceptions Associated with the PSW	6-6	PER Event	6-12
Early Exception Recognition	6-6	Privileged-Operation Exception	6-12
Late Exception Recognition	6-6	Protection Exception	6-12
Types of Instruction Ending	6-6	Significance Exception	6-13
Interruptible Instructions	6-7	Special-Operation Exception	6-13
Point of Interruption	6-7	Specification Exception	6-13
Ending of Interruptible Instructions	6-7	Recognition of Access Exceptions	6-13
Machine-Check Interruption	6-8	Nontransparent Nullification	6-16
Supervisor-Call Interruption	6-8	Multiple Program-Interruption Conditions	6-17
Program Interruption	6-8	External Interruption	6-19
Program-Interruption Conditions	6-9	Clock Comparator	6-19
Addressing Exception	6-9	CPU Timer	6-19
Data Exception	6-9	External Signal	6-20
Decimal-Divide Exception	6-10	Interrupt Key	6-20
Decimal-Overflow Exception	6-10	Interval Timer	6-20
Execute Exception	6-10	Input/Output Interruption	6-20
Exponent-Overflow Exception	6-10	Restart Interruption	6-21
Exponent-Underflow Exception	6-10	Priority of Interruptions	6-21
Fixed-Point-Divide Exception	6-10		
Fixed-Point-Overflow Exception	6-11		

The interruption facility permits the CPU to change its state as a result of conditions external to the machine, within the machine, 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: machine check, supervisor call, program, external, input/output, and restart.

### Interruption Action

An interruption consists in storing the current PSW as an old PSW, storing further detail 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 take place 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 the figure "Interruption Action."

## Source Identification

The six classes of interruptions (machine check, supervisor call, program, external, I/O, and restart) 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 storage locations during the interruption. (See also the section "Assigned Storage Locations" in Chapter 3, "Storage.") For supervisor-call, program, external, and I/O interruptions, the interruption code comprises 16 bits.

For machine-check interruptions, the interruption code comprises 64 bits and is stored at 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 save areas. (See Chapter 11, "Machine-Check Handling.")

For supervisor-call interruptions, in the EC mode, the interruption code is stored at locations 138-139, and the instruction-length code is stored in bit positions 5 and 6 of location 137. In the BC mode,

the interruption code and instruction-length code are placed in the old PSW.

For program interruptions, in the EC mode, the interruption code is stored at locations 142-143, and the instruction-length code is stored in bit positions 5 and 6 of location 141. In the BC mode, the interruption code and instruction-length code are placed in the old PSW. Further information may be provided in the form of the access-exception address, monitor-class number, monitor code, PER code, and PER address, which are stored at locations 144-159.

For external interruptions, in the EC mode, the interruption code is stored at locations 134-135. In the BC mode, the interruption code is placed in the old PSW.

For I/O interruptions, in the EC mode, the interruption code, which contains the I/O address, is stored at locations 186-187. In the BC mode, the interruption code is placed in the old PSW. Additional information is provided by the contents of the channel-status word (CSW) stored at location 64. Further information may be provided by the limited channel logout stored at location 176.

Source Identification	Interruption Code	PSW Mask Bits		Mask Bits in Control Registers			Execution of Instruction Identified by Old PSW
		BC	EC	Register	Bit	ILC Set	
Machine check (old PSW 48, new PSW 112)	Locations 232-239 <sup>1</sup>						
Exigent condition		13	13			x	terminated
Repressible condition		13	13	14	4-7	x	unaffected <sup>7</sup>
Supervisor call (old PSW 32, new PSW 96)							
Instruction bits	00000000 ssssssss					1, 2	completed
Program (old PSW 40, new PSW 104)							
Operation	00000000 p0000001					1, 2, 3	suppressed
Privileged operation	00000000 p0000010					1, 2	suppressed
Execute	00000000 p0000011					2	suppressed
Protection	00000000 p0000100					1, 2, 3	suppressed or terminated
Addressing	00000000 p0000101					1, 2, 3	suppressed or terminated
Specification	00000000 p0000110					0, 1, 2, 3	suppressed or completed
Data	00000000 p0000111					2, 3	suppressed or terminated
Fixed-point overflow	00000000 p0001000	36	20			1, 2	completed
Fixed-point divide	00000000 p0001001					1, 2	suppressed or completed
Decimal overflow	00000000 p0001010	37	21			2, 3	completed
Decimal divide	00000000 p0001011					2, 3	suppressed
Exponent overflow	00000000 p0001100					1, 2	completed
Exponent underflow	00000000 p0001101	38	22			1, 2	completed
Significance	00000000 p0001110	39	23			1, 2	completed
Floating-point divide	00000000 p0001111					1, 2	suppressed

Interruption Action (Part 1 of 2)

Source Identification	Interruption Code	PSW Mask Bits		Mask Bits in Control Registers		ILC Set	Execution of Instruction Identified by Old PSW
		BC	EC	Register	Bit		
Special operation	00000000 p0010011			0	1	2	suppressed
Page access	00000000 p0011000					1, 2, 3	nullified
Page state	00000000 p0011010					2	suppressed
Page transition	00000000 p0011011					2	suppressed
Monitor event	00000000 p1000000			8	16+	2	completed
PER event	00000000 1n0nnnn <sup>2</sup>	*	1	9	0-3	0, 1, 2, 3	completed <sup>3</sup>
External (old PSW 24, new PSW 88)							
Interval timer	00000000 1eeeeeee	7	7	0	24	x	unaffected
Interrupt key	00000000 e1eeeeee	7	7	0	25	x	unaffected
External signal 2	00000000 ee1eeeee	7	7	0	26	x	unaffected
External signal 3	00000000 eee1eeee	7	7	0	26	x	unaffected
External signal 4	00000000 eeee1eee	7	7	0	26	x	unaffected
External signal 5	00000000 eeeee1ee	7	7	0	26	x	unaffected
External signal 6	00000000 eeeee1e	7	7	0	26	x	unaffected
External signal 7	00000000 eeeeeee1	7	7	0	26	x	unaffected
Clock comparator	00010000 00000100	7	7	0	20	x	unaffected
CPU timer	00010000 00000101	7	7	0	21	x	unaffected
Input/output (old PSW 56, new PSW 120)							
Channel 0	00000000 dddddddd <sup>4</sup>	0	6	2	0 <sup>5</sup>	x	unaffected
Channel 1	00000001 dddddddd <sup>4</sup>	1	6	2	1 <sup>5</sup>	x	unaffected
Channel 2	00000010 dddddddd <sup>4</sup>	2	6	2	2 <sup>5</sup>	x	unaffected
Channel 3	00000011 dddddddd <sup>4</sup>	3	6	2	3 <sup>5</sup>	x	unaffected
Channel 4	00000100 dddddddd <sup>4</sup>	4	6	2	4 <sup>5</sup>	x	unaffected
Channel 5	00000101 dddddddd <sup>4</sup>	5	6	2	5 <sup>5</sup>	x	unaffected
Channel 6 and up	cccccccc dddddddd <sup>4</sup>	6	6	2	6+	x	unaffected
Restart (old PSW 8, new PSW 0)							
Restart key	00000000 00000000 <sup>6</sup>					x	unaffected

**Explanation:**

- <sup>1</sup> A model-dependent machine-check-interruption code of 64 bits is stored at locations 232-239.
  - <sup>2</sup> When the interruption code indicates a PER event, an ILC of 0 may be stored only when bits 8-15 of the interruption code are 10000110 (PER, specification).
  - <sup>3</sup> The unit of operation is completed, unless a program exception concurrently indicated has caused the unit of operation to be nullified, suppressed, or terminated.
  - <sup>4</sup> In the EC mode, the I/O address is stored at locations 186-187.
  - <sup>5</sup> For channels 0-5, channel masks in control register 2 have no effect in the BC mode.
  - <sup>6</sup> Bits 16-31 in the old PSW in the BC mode are set to zeros. No interruption code is provided in the EC mode.
  - <sup>7</sup> For a repressible machine-check-interruption condition, the effect of the condition is identified by the validity bits in the machine-check-interruption code. The instruction has been unaffected only if all the associated validity bits are ones.
- + Plus the following bits in the control register.
  - \* In the BC mode, program-event recording is disabled.
  - c Channel-address bits.
  - d Device-address bits.
  - e If one, the bit indicates another concurrent external-interruption condition.
  - n A possible nonzero code, indicating another concurrent program-interruption condition.
  - p If one, the bit indicates a concurrent program-event interruption condition.
  - s Bits of the I field of SUPERVISOR CALL.
  - x Unpredictable in the BC mode; not stored in the EC mode.

**Interruption Action (Part 2 of 2)**

For restart interruptions, in the EC mode, no interruption code is stored. In the BC mode, an interruption code of zero is placed in the old PSW.

### ***Enabling and Disabling***

By means of mask bits in the current PSW and in control registers, the CPU may be enabled or disabled for all I/O, external, 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 these interruptions can take place.

When a mask bit is zero, the CPU is disabled for the corresponding interruptions. The conditions that cause I/O or external interruptions remain pending. Machine-check-interruption conditions 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.

Program interruptions for which mask bits are not provided, as well as the supervisor-call and restart interruptions, are always taken.

The mask bits may allow or disallow all interruptions within the class, or they may selectively allow or disallow interruptions for particular causes. This control may be provided by mask bits in the PSW that are assigned to particular causes, such as the bits assigned to the four maskable program-interruption conditions. Alternatively, there may be a hierarchy of masks, where a mask bit in the PSW controls all interruptions within a type, and mask bits in a control register provide more detailed control over the sources.

When the mask bit is one, the CPU is enabled for the corresponding interruptions. When the mask bit is zero, these interruptions are disallowed. Interruptions that are controlled by a hierarchy of masks are allowed only when all controlling mask bits are ones.

### **Programming Notes**

1. Mask bits in the PSW provide a means of disallowing all 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 housekeeping 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. Since 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.

### ***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.

When the old PSW specifies the EC mode, the ILC for supervisor-call and program interruptions is stored in bit positions 5 and 6 of the bytes at locations 137 and 141, respectively. For machine-check, external, I/O, and restart interruptions, the ILC is not stored since it cannot be related to the length of the last-executed instruction.

When the old PSW specifies the BC mode, the ILC is stored in bit positions 32 and 33 of that PSW. The ILC is meaningful, however, only after a program or supervisor-call interruption. For I/O, external, machine-check, and restart interruptions, the ILC does not indicate the length of the last-executed instruction and is unpredictable. Similarly, the ILC is unpredictable in the PSW stored during execution of the machine-save function and when the PSW is displayed.

For supervisor-call and program interruptions, a nonzero instruction-length code identifies in halfwords the length of the instruction that was last executed. 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.

The value of a nonzero instruction-length code is related to the leftmost two bits of the instruction. The value is not contingent 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, indicates that the location of the instruction causing the interruption is not made available to the program.

An ILC of 0 occurs only when a specification exception is recognized that is due to a PSW-format error, other than one due to an odd instruction address, and the invalid PSW has been introduced by LOAD PSW or an interruption. (See the section "Exceptions Associated with the PSW" later in this chapter.) In the case of LOAD PSW, the address of the instruction has been replaced by the instruction address of 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 and the supervisor-call interruption, a PER event may be indicated concurrently with a specification exception having an ILC of 0.

### ILC on Instruction-Fetch Exceptions

When a program interruption occurs because of an exception that prohibits access to the instruction, 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 or protection exception is encountered on fetching an instruction, the ILC is arbitrarily set to 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. By reducing the instruction address in the old PSW by the number of halfword locations indicated in the instruction-length code, the address originally appearing in the PSW may be obtained.
2. When a page-access exception is recognized while fetching an instruction, including the target

instruction of EXECUTE, 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 other than page access are encountered on fetching the target instruction of EXECUTE, the ILC is 2.

### 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 old PSW must be reduced to obtain the address of the last instruction executed, unless one of the following situations exists:
  - a. The interruption is caused by a page-access exception.
  - b. An interruption for a PER event occurs before the execution of an interruptible instruction is ended.
  - c. The interruption is caused by a PER event due to LOAD PSW or a branch or linkage instruction, including SUPERVISOR CALL.
  - d. The interruption is caused by an access exception encountered in fetching an instruction, and the instruction address has been introduced into the PSW by a means other than sequential operation (by a branch instruction, LOAD PSW, or an interruption).
  - e. The interruption is caused by a specification exception because of an odd instruction address.

For situations a and b above, a unit of operation is nullified, and the instruction designated by the instruction address is the same as the last one executed. These two are the only cases where the instruction address in the old PSW identifies the instruction causing the exception.

For situations c, d, and e, the address of the last instruction executed has been replaced in the old PSW and cannot be calculated using the one appearing in the old PSW.

2. When a PER event is indicated, bit 8 in the interruption code is one, the PER address in the

word at location 152 identifies the location of the instruction causing the interruption, and the instruction-length code (ILC) is redundant. 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.

### ***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 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**

A program interruption for a specification exception occurs immediately after the PSW becomes active if a one is introduced in an unassigned bit position of an EC-mode PSW (that is, bit positions 0, 2-5, 16, 17, 24-39).

The interruption takes place 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 is taken instead, and the pending interruption is subject to the mask bits of the new PSW introduced by the program interruption.

When the execution of LOAD PSW or an interruption introduces a PSW with one of the above error conditions, the instruction-length code is set to 0, and the newly introduced PSW, except for the interruption code and the instruction-length code in the BC mode, is stored unmodified as the old PSW. When one of the above 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 updated by two halfword locations. The PSW containing the invalid value introduced into the system-mask field is stored as the old PSW.

When a PSW with one of the above error conditions is introduced during initial program loading, the loading sequence is not completed, and the load indicator remains on.

### **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 63 is one).
- An access (protection, addressing, or page-access) exception 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 address.

The instruction-length code and instruction address stored in the program old PSW under these conditions are discussed in the section "ILC on Instruction-Fetch Exceptions" in this chapter.

If the invalid PSW causes the CPU to be enabled for a pending I/O, external, or machine-check interruption, the corresponding interruption occurs, and the PSW invalidity is not recognized. Similarly, the specification or access exception is not recognized in a PSW specifying the wait state.

### **Programming Notes**

1. The execution of LOAD PSW, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK is suppressed on a protection or addressing 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 instruction-length code 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 occurs. (See the section "Priority of Interruptions" in this chapter.)

### ***Types of Instruction Ending***

Instruction execution is said to end in one of four ways: completion, nullification, suppression, and termination.

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 instruction to be executed.

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 interruption after suppression designates the next sequential instruction.

Nullification 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 instead of the next sequential instruction.

Termination of instruction execution causes the contents of any fields due to be changed by the instruction to be unpredictable. The operation may have replaced all, part, or none of the contents of the designated result fields and may have changed the condition code if such change was 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 instruction-length code are not affected; and the state or the operation of the machine has not been affected in any other way. The instruction address in the old PSW on an interruption after termination designates the next sequential instruction.

## ***Interruptible Instructions***

### **Point of Interruption**

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. The entire execution of an instruction is one operation.

For the following instructions, referred to as interruptible instructions, an interruption is permitted after a partial execution of the instruction:

COMPARE LOGICAL LONG  
MOVE LONG

The execution of an interruptible instruction is considered to consist 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 particular condition that causes the execution of the instruction to be interrupted.

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, of one unit of operation.

### **Ending 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 prior units of operation have been completed.

On completion of a unit of operation other than the last one and on nullification of any unit of operation, the instruction address in the old PSW designates the interrupted instruction, and 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 on the interruption is made the current PSW. It depends on the instruction how the operand parameters are adjusted.

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 from the point of interruption. As in the case of completion and nullification, it depends on the instruction how the operand parameters are adjusted.

When a unit of operation of an interruptible instruction is terminated, the contents, in general, of any fields due to be changed by the instruction are unpredictable. On an interruption, the instruction address in the old PSW designates the next sequential instruction.

### **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 machine-check, external, and I/O conditions and 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 single unit of operation is performed, but the unit of operation for the various cases of stopping may be different.

## 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 location of the fault and extent of the damage.

A machine-check interruption causes the old PSW to be stored at location 48 and a new PSW to be fetched from location 112. When the old PSW specifies the BC mode, the contents of the interruption-code and instruction-length-code fields in the old PSW are unpredictable. In the EC mode, the instruction-length code is not stored.

The cause and severity of the malfunction are identified by a 64-bit machine-check-interruption code stored at locations 232-239. Further information identifying the cause of the interruption and the location of the fault may be stored at locations 216-511.

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.

## 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 location 32 and a new PSW to be fetched from location 96.

The contents of bit positions 8-15 of SUPERVISOR CALL are placed in the low-order byte of the interruption code. The high-order byte of the interruption code is set to zero. The instruction-length code is 1, unless the instruction

was executed by means of EXECUTE, in which case the code is 2.

When the old PSW specifies the EC mode, the interruption code is placed at locations 138-139, the instruction-length code is placed in bit positions 5 and 6 of the byte at location 137, with the other bits set to zeros, and zeros are stored at location 136. When the old PSW specifies the BC mode, the interruption code and instruction-length code appear in the old PSW.

## Program Interruption

Exceptions resulting from execution of the program, including the improper specification or use of instructions and data, or from the detection of a PER event or monitor event, generate a program interruption.

A program interruption causes the old PSW to be stored at location 40 and a new PSW to be fetched from location 104.

The cause of the interruption is identified by the interruption code. When the old PSW specifies the EC mode, the interruption code is placed at locations 142-143, the instruction-length code is placed in bit positions 5 and 6 of the byte at location 141 with the rest of the bits set to zeros, and zeros are stored at location 140. When the old PSW specifies the BC mode, the interruption code and the instruction-length code are placed in the old PSW. For some causes, additional information identifying the reason for the interruption is stored at locations 144-159 in both the EC and BC modes.

Except for the PER-event condition, the condition causing the interruption is identified 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.

The PER-event condition is indicated by setting bit 8 of the interruption code to one, with bits 0-7 set to zeros. When this is the only condition, bits 9-15 are also set to zeros. When a PER-event condition is indicated concurrently with another condition, bit 8 is one, and the coded value for the other condition appears in bit positions 9-15.

A program interruption can occur only when the corresponding mask bit, if any, is one. The program mask in the PSW permits masking four of the exceptions, bit 1 in control register 0 controls whether SET SYSTEM MASK causes a special-operation exception, bits 16-31 in control register 8 permit masking interruption conditions due to

monitor events, and, in the EC mode, masks are provided for controlling interruptions due to PER events. When the 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 format error or causes an exception to be recognized in the process of instruction fetching, a string of program interruptions takes place. See the section "Priority of Interruptions" in this chapter for a description of how such strings are terminated.
2. Some of the conditions indicated as program exceptions may be recognized also by an I/O operation, in which case the exception is indicated in the channel-status word.

#### ***Program-Interruption Conditions***

The following is a detailed description of each program-interruption condition.

#### **Addressing Exception**

An addressing exception is recognized when the CPU causes a reference to a virtual-storage location that is not provided. A storage location is not provided when the page address, bits 8-20 of the storage address, equals or exceeds the page-capacity count. An address designating a storage location that is not provided is referred to as invalid.

The execution of the instruction is suppressed when the location of the instruction, including the location of the target instruction of EXECUTE, is not provided. Except for some specific instructions whose execution is suppressed, the operation is terminated when an operand location is not provided. For termination, changes may occur only to result fields, which include 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 provided, the operation is suppressed.

The instructions whose execution is always suppressed are LOAD PSW, SET CLOCK COMPARATOR, SET CPU TIMER, SET SYSTEM MASK, STORE CLOCK COMPARATOR, STORE CPU ID, STORE CPU TIMER, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK.

When part of an operand location is provided and part is not, storing may be performed in the part that is provided.

When the address of any halfword of an instruction is invalid, the instruction-length code (ILC) is 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the ILC is 1, 2, or 3.

In all cases of addressing exceptions not associated with instruction fetching, the ILC is 1, 2, or 3, designating the length of the instruction that caused the reference. When an addressing exception is associated with fetching the target of EXECUTE, the ILC is 2.

#### **Data Exception**

A data exception is recognized when:

1. The sign or digit codes of operands in the decimal instructions (described in Chapter 8, "Decimal Instructions") or in CONVERT TO BINARY are invalid.
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.
3. The multiplicand in MULTIPLY DECIMAL has an insufficient number of high-order zeros.

For all instruction other than EDIT and EDIT AND MARK, the action taken for a data exception depends on whether a sign code is invalid. The operation is suppressed when a sign code is invalid, regardless of whether any other condition causing the exception exists; when no sign code is invalid, the operation is terminated. When the operation is terminated, the contents of the sign position in the rightmost byte of the result field either remain unchanged or are set to the preferred sign code; the contents of the remainder of the result field are unpredictable.

In the case of EDIT and EDIT AND MARK, an invalid sign code is not recognized; the operation is terminated on a data exception for an invalid digit code.

The instruction-length code is 2 or 3.

## Programming Notes

1. The definition for data exception permits termination when no sign code is invalid. On some models, valid digit codes may be placed in the result location even if the original contents were invalid. Thus it is possible, after getting a data exception, for all fields to appear valid.
2. When, on a program interruption for data exception, the program finds that a sign code is invalid, the operation has been suppressed if the following two conditions are met:
  - a. The invalid sign of the source field is not located in the numerical portion of the result field.
  - b. The sign code appears in a position specified by the instruction to be checked for valid sign. (This condition excludes the first operand of ZERO AND ADD and both operands of EDIT and EDIT AND MARK.)

An invalid sign code for the rightmost byte of the result field is not generated when the operation is terminated. However, an invalid second-operand sign code is not necessarily preserved when it appears in the numerical portion of the result field.

## 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 can be indicated 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.

## Decimal-Overflow Exception

A decimal-overflow exception is recognized when one or more significant high-order digits are lost because the destination field in a decimal operation is too small to contain the result.

The interruption may be disallowed by PSW bit 21 in the EC mode and by PSW bit 37 in the BC mode.

The operation is completed. The result is obtained by ignoring the overflow information, and condition code 3 is set.

The instruction-length code is 2 or 3.

## Execute Exception

The execute exception is recognized when the target instruction of EXECUTE is another EXECUTE.

The operation is suppressed.

The instruction-length code is 2.

## Exponent-Overflow Exception

An exponent-overflow exception is recognized when the result characteristic in floating-point addition, subtraction, multiplication, or division 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 or 2.

## Exponent-Underflow Exception

An exponent-underflow exception is recognized when the result characteristic in floating-point addition, subtraction, multiplication, halving, or division is less than zero and the result fraction is not zero.

The interruption may be disallowed in the EC mode by PSW bit 22, and in the BC mode by PSW bit 38.

The operation is completed. The 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 or 2.

## Fixed-Point-Divide Exception

A fixed-point-divide exception is recognized when in fixed-point division the divisor is zero or the quotient exceeds the register size, or when the result of CONVERT TO BINARY exceeds 31 bits.

In the case of division, the operation is suppressed. Execution of CONVERT TO BINARY is completed by ignoring the high-order bits that cannot be placed in the register.

The instruction-length code is 1 or 2.

### **Fixed-Point-Overflow Exception**

A fixed-point-overflow exception is recognized when an overflow occurs during signed binary arithmetic or left-shift operations.

The interruption may be disallowed in the EC mode by PSW bit 20, and in the BC mode by PSW bit 36.

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 or 2.

### **Floating-Point-Divide Exception**

A floating-point-divide exception is recognized when a floating-point division by a number with a zero fraction is attempted.

The operation is suppressed.

The instruction-length code is 1 or 2.

### **Monitor Event**

A monitor event is recognized when MONITOR CALL is executed and the mask bit in control register 8 corresponding to the class specified by instruction bits 12-15 is one.

The operation is completed.

As part of the interruption, information identifying the event is stored at locations 148-149 and 156-159. See the section "Monitoring" in Chapter 4, "Control," for a detailed description of the interruption condition.

The instruction-length code is 2.

### **Operation Exception**

An operation exception is recognized when the CPU encounters an instruction which has an invalid operation code.

For the purpose of checking the operation code of an instruction, the operation code is defined as follows:

1. When the first eight bits of an instruction have the hexadecimal value B2, the first 16 bits form the operation code.
2. In all other cases, the first eight bits alone form the operation code.

The operation is suppressed.

The instruction-length code is 1, 2, or 3.

### **Programming Notes**

1. In the case of I/O instructions with the values 9C, 9D, and 9E in bit positions 0-7, the value of bit 15 is used to distinguish between two instructions. Bits 8-14, however, are not checked for zeros, and these operation codes never cause an operation exception to be recognized.

To ensure that presently written programs run if and when the operation codes 9C, 9D, and 9E are extended further to provide for new functions, only zeros should be placed in bit positions 8-14. Similarly, zeros should be placed in bit positions 8-15 in the instruction with the operation code 9F. In accordance with these recommendations, the operation codes for seven of the I/O instructions are shown as 9C00, 9C01, 9D00, 9D01, 9E00, 9E01, and 9F00.

2. The operation code 00, with a two-byte instruction format, and the set of sixteen 16-bit operation codes B2E0 to B2EF, with a four-byte instruction format, are allocated for use by the program when an indication of invalid operation is required. It is improbable that these operation codes will ever be assigned to an instruction implemented in the CPU.
3. Some models may offer instructions not described in this publication, such as those provided for emulation. 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 otherwise alter the machine so as to affect the execution of subsequent instructions. In order to avoid the possibility of accidentally causing such operation, an instruction with an operation code not described in this publication should be issued only when the specific function associated with the operation code is desired.

### **Page-Access Exception**

A page-access exception is recognized when storage is addressed either explicitly or implicitly by the CPU and the addressed storage location is in a page that is in the connected or disconnected state.

The exception is recognized as part of the execution of the instruction when an attempt is made to access either the instruction or operand location. However, page-access exceptions are not recognized for the page operands of the instructions CLEAR PAGE, CONNECT PAGE, DECONFIGURE PAGE,

**DISCONNECT PAGE, MAKE ADDRESSABLE, and MAKE UNADDRESSABLE.**

The unit of operation is nullified, except for the possible effects on storage described in the section "Nontransparent Nullification" in this chapter.

The address of the storage location causing the exception is stored at locations 145-147, and zeros are stored at location 144. The low-order 11 bits of the address stored are unpredictable.

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. When the exception occurs during fetching of an instruction, the ILC is 1, 2, or 3, the value being unpredictable.

#### **Page-State Exception**

A page-state exception is recognized when the target page of the CLEAR PAGE instruction is in the disconnected state.

The operation is suppressed.

The instruction-length code is 2 or 3.

#### **Page-Transition Exception**

A page-transition exception can only be recognized for instructions that cause a page-state transition. These instructions are CONNECT PAGE, DECONFIGURE PAGE, DISCONNECT PAGE, MAKE ADDRESSABLE, and MAKE UNADDRESSABLE.

The exception is recognized as part of the execution of the instruction when attempting to perform an invalid page-state transition. For the definition of an invalid page-state transition, see the section "Page States" in Chapter 3, "Storage."

The operation is suppressed.

The instruction-length code is 2.

#### **PER Event**

A PER event is recognized when program-event recording (PER) is specified by the contents of control registers 9-11 and one or more of these events occur.

In the EC mode, the interruption may be disallowed by PSW bit 1. In the BC mode, program-event recording is disabled.

The unit of operation is completed, unless another concurrently indicated condition has caused the unit of operation to be nullified, suppressed, or terminated.

As part of the interruption, information identifying the event is stored at locations 150-155. See the

section "Program-Event Recording" in Chapter 4, "Control," for a detailed description of the interruption condition.

The instruction-length code is 0, 1, 2, or 3. Code 0 can be set only if a specification exception is indicated concurrently.

#### **Privileged-Operation Exception**

A privileged-operation exception is recognized when the CPU encounters a privileged instruction in the problem state.

The operation is suppressed.

The instruction-length code is 1 or 2.

#### **Protection Exception**

A protection exception is recognized when the CPU attempts to access a storage location that is protected against the type of reference by the storage key.

The execution of the instruction is suppressed when the location of the instruction, including the location of the target instruction of EXECUTE, is protected against fetching. Except for some specific instructions whose execution is suppressed, the operation is terminated when a protection exception is encountered during a reference to an operand location. 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 designated 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.

The instructions whose execution is always suppressed are: LOAD PSW, SET CLOCK COMPARATOR, SET CPU TIMER, SET SYSTEM MASK, STORE CLOCK COMPARATOR, STORE CPU ID, STORE CPU TIMER, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK.

On fetching, the protected information is not loaded into a register or moved to another storage location. When a part of an operand location is protected against storing and part is not, storing may be performed in the unprotected part. The contents of a protected location remain unchanged.

For a protected operand location, the instruction-length code is 1, 2, or 3, designating the length of the instruction that caused the reference.

When the location of any part of the instruction is protected against fetching, the ILC is 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the ILC is 1, 2, or 3.

#### **Significance Exception**

A significance exception is recognized when the result fraction in floating-point addition or subtraction is zero.

The interruption may be disallowed in the EC mode by PSW bit 23, and in the BC mode by PSW bit 39.

The operation is completed. The 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 and sign of the result.

The instruction-length code is 1 or 2.

#### **Special-Operation Exception**

A special-operation exception is recognized when a SET SYSTEM MASK instruction is encountered in the supervisor state and the SSM-control bit, bit 1 of control register 0, is one.

The execution of SET SYSTEM MASK is suppressed.

The instruction-length code is 2.

#### **Specification Exception**

A specification exception is recognized for the following causes:

1. An odd instruction address is introduced into the PSW.
2. An operand address does not designate an integral boundary in an instruction requiring such integral-boundary designation.
3. The storage address in SET STORAGE KEY or INSERT STORAGE KEY does not have zeros in the four low-order bit positions.
4. An odd-numbered general register is designated by an R field of an instruction that requires an even-numbered register designation.
5. A floating-point register other than 0, 2, 4, or 6 is specified for a short or long operand, or a floating-point register other than 0 or 4 is specified for an extended operand.
6. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.
7. 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.

8. Bit positions 8-11 of MONITOR CALL do not contain zeros.
9. A one is introduced into an unassigned bit position of the EC-mode PSW (bit positions 0, 2-5, 16-17, 24-39).
10. Page 0 is designated to become connected or disconnected.

The execution of the instruction identified by the old PSW is suppressed. However, for cause 9, the operation that produces the invalid PSW is completed, and an interruption occurs immediately thereafter.

When the instruction address is odd (cause 1), the instruction-length code (ILC) is 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the ILC is 1, 2, or 3.

For causes 2-8 and 10, the ILC is 1, 2, or 3, designating the length of the instruction causing the reference.

When the exception is recognized because of cause 9, and the invalid bit value has been introduced by LOAD PSW or an interruption, the ILC is 0. When the exception due to cause 9 is introduced by SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the ILC is 2.

See the section "Exceptions Associated with the PSW" in this chapter for more details of when specification exceptions associated with the PSW are recognized.

#### **Recognition of Access Exceptions**

The addressing, page-access, and protection exceptions are collectively referred to as access exceptions.

An access exception due to fetching an instruction is indicated when an instruction halfword cannot be fetched without encountering the exception. The exception is indicated as part of the execution of the instruction.

Except for the specific cases described below, an access exception due to a reference to an operand location is indicated whenever a reference to a part of the designated storage operand causes the exception. The exception for a partially inaccessible operand is recognized even if the operation could be completed without the use of the inaccessible part of the operand. The access exception is indicated as part of the execution of the instruction making the reference.

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. Additionally, each instruction can cause an access exception to be recognized due to instruction fetch.

The following are exceptions or special cases where the instruction does not explicitly specify the extent of the storage operand or where the instruction provides for completion of execution without the use of the entire operand (the handling of these cases is summarized in the figure "Recognition of Access Exceptions"):

1. When the instructions COMPARE LOGICAL (CLC or CL), COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) with a nonzero mask, and COMPARE LOGICAL LONG (CLCL) designate part of an operand in an inaccessible location but the operation can be completed by using the accessible operand parts, it is unpredictable whether the access exception for the inaccessible part is indicated.
2. Access exceptions are not indicated for that part of the first operand (argument) of TRANSLATE AND TEST (TRT) which is not used for the completion of the operation.
3. Access exceptions are not indicated for that part of the second operand (list) of TRANSLATE (TR) and TRANSLATE AND TEST which is not used for the completion of the operation.
4. Access exceptions are not indicated for that part of the second operand (source) of EDIT (ED) and EDIT AND MARK (EDMK) which is not used for the completion of the operation.
5. When the instructions MOVE WITH OFFSET (MVO), PACK, and UNPACK (UNPK) designate part of the second operand in an inaccessible location but the operation can be completed by using the accessible operand parts, it is unpredictable whether the access exception for the inaccessible part is indicated.
6. Access exceptions are not indicated for that part of the second operand (source) of MOVE LONG (MVCL) which is not used for the completion of the operation.
7. When the mask in INSERT CHARACTERS UNDER MASK (ICM) and COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) is zero, access exceptions are indicated for the one byte designated by the second-operand address.
8. When the mask in STORE CHARACTERS UNDER MASK (STCM) is zero, access exceptions are not indicated.
9. When the saved information is invalid, access exceptions are not indicated for either operand of RETRIEVE STATUS AND PAGE (RSP).

Instruction

Is an access exception indicated for that part of the designated operand which is not used for the completion of the operation ?

Instructions that can be completed without the use of the entire designated or implied operand:

CLC, CL	Unpredictable
CLM (nonzero mask)	Unpredictable
CLCL	Unpredictable*
TRT (first operand)	No
TR, TRT (second operand)	No
ED, EDMK (second operand)	No
RSP (invalid save)	No

Instructions in which the second operand may specify more data than can be processed with the designated first operand:

PACK, UNPK, MVO	Unpredictable
MVCL	No

Special cases:

ICM, CLM (zero mask)	Yes for one byte
STCM (zero mask)	No

**Explanation:**

- Unpredictable It is unpredictable whether the exception is indicated.
- No The exception is not indicated.
- Yes The exception is indicated.
- \* For CLCL, any addressing or page-access exception indicated is for the current page or the following page of each operand.

Access exceptions include the following:

- Addressing
- Page access
- Protection

**Recognition of Access Exceptions**

The execution of the interruptible instructions CLCL and MVCL is initiated only when no addressing or page-access exceptions for the initial page of each operand exist, and the initiation may additionally be contingent on the absence of addressing or page-access exceptions for the following page of each operand. After the execution of the instruction has been initiated, an addressing or page-access exception may be indicated as early as when execution has progressed to the point where the last accessible page of the operand preceding the page causing the exception is being processed.

The extents of the operands that are actually used in the operation may be established in a pretest for operand accessibility at the beginning of the execution of the instruction.

In the case of TR, ED, and EDMK, the initiation of the execution is contingent only on the absence of addressing or page-access exceptions for that part of the second operand that is actually used for the completion of the operation.

If the first operand of TR or either operand of ED or EDMK is changed by an I/O operation, after the initial pretest but before completion of execution, the results are unpredictable. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable.

This case is an exception to the general rule that the operation is nullified on page-access exceptions. When, in this case, an interruption for page access occurs, the instruction address in the old PSW points to the instruction causing the exception even though partial results have been stored.

#### **Programming Note**

An access exception is indicated as part of the execution of the instruction with which the exception is associated. In particular, the exception is not recognized when the CPU has made an attempt to fetch from the inaccessible location or otherwise has detected the access exception, but a branch instruction or an interruption changes the instruction sequence such that the instruction is not executed.

The following are some specific storage references where access exceptions, including storage protection when applicable, are recognized even if the operation could be completed without the use of the inaccessible part of the operand:

1. Fetching the operand of TEST UNDER MASK with a zero mask.

2. Fetching parts of operands of algebraic-compare instructions (C and CH).
3. Fetching parts of operands of floating-point instructions.
4. References to the first-operand location of decimal instructions when the second operand in addition and subtraction is zero or in multiplication and division is one.
5. Storing the pattern character in an edit operation when the pattern character remains unchanged.
6. Storing during SHIFT AND ROUND DECIMAL when no shifting or rounding takes place.
7. Storing during move operations when the first- and second-operand locations coincide.
8. Storing the first operand of OR (OI and OC) when the corresponding second-operand byte is zero, as well as the analogous cases for AND and EXCLUSIVE OR.
9. Storing the first operand of TRANSLATE when the argument and function bytes are the same.

With a nonzero mask in INSERT CHARACTERS UNDER MASK, COMPARE LOGICAL CHARACTERS UNDER MASK, and STORE CHARACTERS UNDER MASK, access exceptions are indicated only for the extent of the storage operand designated by the mask. In MOVE LONG or COMPARE LOGICAL LONG, no exceptions are recognized for any operand having a length of zero.

#### ***Nontransparent Nullification***

For page-access exceptions, the unit of operation is nullified, except that, on some models, a channel may observe the effects on storage described in the following cases:

- When a storage operand for which there is a store-type access crosses a page boundary and a page-access exception is recognized, that part of the operand which does not cause a page-access exception may be changed to an intermediate value and then changed back to the original value.
- In an instruction involving two storage operands (for example, an SS-format instruction or MOVE LONG) for which there is a store-type access to an operand, that operand may change to an intermediate value and then back to the original value when a page-access exception is recognized for the other operand.

Except for the instructions which operate on decimal data, the intermediate value, if any, is always equal to what would have been the final value if the page-access exception had not occurred.

### **Programming Note**

On some models, when CPU instructions are used to modify storage locations simultaneously being accessed by a channel, page-access exceptions may result in intermediate values being observed by the channel. To avoid getting these intermediate values (especially when modifying a CCW chain), either one storage page should be operated on at a time or preliminary testing should be performed to ensure that all required pages are addressable.

### ***Multiple Program-Interruption Conditions***

Except for PER 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 two conditions exist of the same priority, it is unpredictable which is indicated. In particular, the priority of access exceptions associated with the two parts of an operand that crosses a page boundary is unpredictable and is not necessarily related to the sequence specified for the access of bytes within the 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.

The figure "Priority of Program-Interruption Conditions" lists the priorities of all program-interruption conditions other than PER events. 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." The priorities of access exceptions for a single access are, in descending order of priorities:

1. Addressing exception
2. Page-access exception
3. Protection exception due to an attempt to access a protected instruction or operand location

The relative priorities of any two conditions can be found by comparing the priority numbers within a table from left to right until a mismatch is found. If the first inequality is between numeric characters, the two conditions are either mutually exclusive, or, if both can occur, the condition with the smaller number is indicated. If the first inequality is between alphabetic characters, the two conditions are not exclusive, and it is unpredictable which is indicated when both occur.

The second instruction halfword is accessed only if bits 0-1 of the instruction are not 00. The third instruction halfword is accessed only if bits 0-1 of the instruction are 11.

1. Specification exception due to a one in an unassigned bit position of an EC-mode PSW.<sup>1</sup>
2. Specification exception due to an odd instruction address in the PSW.
3. Access exceptions for first halfword of EXECUTE.<sup>2</sup>
4. Access exceptions for second halfword of EXECUTE.<sup>2</sup>
5. Specification exception due to target instruction of EXECUTE not being specified on halfword boundary.<sup>2</sup>
6. Access exceptions for first instruction halfword.
- 7.A Access exception for second instruction halfword.<sup>3</sup>
- 7.B Access exception for third instruction halfword.<sup>3</sup>
- 7.C.1 Operation exception.
- 7.C.2 Privileged-operation exception.
- 7.C.3 Execute exception.
- 7.C.4 Special-operation exception.
- 8.A Specification exception due to conditions other than those included in 1, 2 and 5 above.
- 8.B Access exceptions for any particular access to an operand in storage.<sup>4</sup>
- 8.C Data exception.<sup>5</sup>
- 8.D Decimal-divide exception.<sup>6</sup>
- 8.E Page-state exception.
9. Page-transition exception.
10. Fixed-point divide, floating-point divide, and conditions, other than PER events, which result in completion. These conditions are mutually exclusive, or their priority is specified in the corresponding definitions.

**Explanation:**

Numbers indicate priority, with priority decreasing in ascending order of numbers; letters indicate no priority.

<sup>1</sup>A one may be introduced in an unassigned bit position of an EC-mode PSW by a new PSW loaded as a result of an interruption or by the instructions LOAD PSW, SET SYSTEM MASK, and STORE THEN OR SYSTEM MASK. The priority shown in the chart is that 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. If the recognition of this exception is considered to be part of the execution of the instruction introducing the error, then it is of lower priority than all other exceptions for that instruction.

<sup>2</sup>Priorities 3, 4, and 5 apply only to an EXECUTE instruction. Priorities 6-10 apply to instructions other than EXECUTE, including the target instruction of EXECUTE.

<sup>3</sup>Separate accesses may occur for each halfword of an instruction. The second instruction halfword is accessed if bits 0-1 of the instruction are not 00. The third instruction halfword is accessed only if bits 0-1 of the instruction are 11.

<sup>4</sup>As in instruction fetching, separate accesses may occur for each portion of an operand. Each of the accesses is of equal priority. Addressing exceptions for INSERT STORAGE KEY, RESET REFERENCE BIT, and SET STORAGE KEY are also included in 8.B. For MOVE LONG and COMPARE LOGICAL LONG, an access exception for a particular operand can be indicated only if the R field for that operand designates an even-numbered register. For instructions requiring that storage operands be specified on integral boundaries, an access exception may be indicated for the extent of the operand that would be implied if alignment were not required.

<sup>5</sup>The exception can be indicated only if the sign, digit, or digits responsible for the exception were fetched without encountering an access exception.

<sup>6</sup>The exception can be indicated only if the sign, digit, or digits responsible for the exception were fetched without encountering an access exception, and only if the digit or digits used in establishing the exception are valid.

**Priority of Program-Interruption Conditions**

## External Interruption

The external interruption provides a means by which the CPU responds to various signals originating either from within or from outside of the machine.

An external interruption causes the old PSW to be stored at location 24 and a new PSW to be fetched from location 88.

The source of the interruption is identified in the interruption code. When the old PSW specifies the EC mode, the interruption code is stored at locations 134-135, and zeros are stored at locations 132-133. When the old PSW specifies the BC mode, the interruption code is placed in bit positions 16-31 of the old PSW, and the instruction-length code is unpredictable.

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 is taken, 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. Whether the CPU is enabled for external interruption is controlled by the external mask, PSW bit 7, and external submask bits in control register 0. Each source for an external interruption has a submask bit assigned to it, and the source can cause an interruption only when the external-mask bit is one and the corresponding submask bit is one. The use of the submask bits does not depend on whether the CPU is in the EC or BC mode.

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:

Interval timer, interrupt key, external signals 2-7  
(indicated concurrently)  
Clock comparator  
CPU timer

### *Clock Comparator*

An interruption request for the clock comparator exists whenever either of the following conditions is met:

1. The time-of-day 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 time-of-day clock, both comparands being considered unsigned binary integers
2. The time-of-day clock is in the error or not-operational state.

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.

The condition is indicated by an external-interruption code of 1004 (hex).

The submask bit is in bit position 20 of control register 0. This bit is initialized to zero.

### *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.

The condition is indicated by an external-interruption code of 1005 (hex).

The submask bit is in bit position 21 of control register 0. This bit is initialized to zero.

## **External Signal**

An interruption request for an external signal is generated when a signal is received on one or more of the signal-in lines. Up to six signal-in lines may be connected, providing for external signal 2 through external signal 7. 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 program reset.

Facilities are provided for holding a separate external-signal request pending for each of the six lines.

External signals 2-7 are indicated by setting to one interruption-code bits 10-15, respectively. Bits 0-7 are set to zeros, and any other bits in the rightmost byte are made zeros unless set to ones for other conditions that are concurrently indicated.

All external signals are subject to control by the submask bit in bit position 26 of control register 0. This bit is initialized to one.

External signaling is independent of I/O operations and interruptions.

For a detailed description, see the *IBM System/360 and System/370 Direct Control and External Interruption Features—Original Equipment Manufacturers' Information*, GA22-6845.

## **Programming Note**

The pattern presented in bit positions 10-15 of the interruption code depends on the pattern received before the interruption is taken. Because of circuit skew, all simultaneously generated external signals do not necessarily arrive at the same time, and some may not be included in the external interruption resulting from the earliest signals. These late signals may cause another interruption to be taken.

## **Interrupt Key**

An interruption request for the interrupt key is generated when the operator activates that key. The request is preserved and remains pending until it is cleared. The pending request is cleared when it causes an interruption and by program reset. When several requests are made before the interruption is taken, only one interruption occurs.

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 condition is indicated by setting bit 9 in the interruption code to one and by setting bits 0-7 to

zeros. Bits 8 and 10-15 are zeros unless set to ones for another condition that is concurrently indicated.

The submask bit is in bit position 25 of control register 0. This bit is initialized to one.

## **Interval Timer**

An interruption request for the interval timer is generated when the value of the interval timer is decremented from a positive number or zero to a negative number. 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 program reset.

The condition is indicated by setting bit 8 in the interruption code to one and by setting bits 0-7 to zeros. Bits 9-15 are zeros unless set to ones for another condition that is concurrently indicated.

The submask bit is in bit position 24 of control register 0. This bit is initialized to one.

## **Input/Output Interruption**

The input/output (I/O) interruption provides a means by which a CPU responds to conditions in I/O devices and channels.

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 in channels or devices until accepted by the CPU. The I/O interruption occurs at the completion of a unit of operation. Priority is established among requests so that only one interruption request is processed at a time. For more details, see the section "Input/Output Interruptions" in Chapter 12, "Input/Output Operations."

When the CPU becomes enabled for I/O interruptions, and a channel has established priority for an 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 location 56, a channel-status word to be stored at location 64, and a new PSW to be fetched from location 120. Upon detection of equipment errors, additional information may be stored in the form of a limited channel logout at location 176.

When the old PSW specifies the EC mode, the I/O address identifying the channel and device causing the interruption is stored at locations 186-187, and zeros are stored at location 185. When the old PSW specifies the BC mode, the interruption code in PSW bit positions 16-31 contains the I/O address, and the instruction-length code in the PSW is unpredictable.

An I/O interruption can occur only while the CPU is enabled for interruption by the channel presenting the request. Mask bits in the PSW and channel masks in control register 2 determine whether the CPU is enabled for interruption by a channel; the method of control depends on whether the current PSW specifies the EC or BC mode.

The channel-mask bits in control register 2 start at bit position 0 and extend for as many contiguous bit positions as the number of channels provided. The assignment is such that a bit is assigned to the channel whose address is equal to the position of the bit in control register 2. Channel-mask bits for installed channels are initialized to one by initial program reset. The state of the channel-mask bits for unavailable channels is unpredictable.

When the current PSW specifies the EC mode, each channel is controlled by the I/O-mask bit, PSW bit 6, and the corresponding channel-mask bit in control register 2; the channel can cause an interruption only when the I/O-mask bit is one and the corresponding channel-mask bit is one.

When the current PSW specifies the BC mode, interruptions from channels 6 and up are controlled by the I/O-mask bit, PSW bit 6, in conjunction with the corresponding channel-mask bit: the channel can cause an interruption only when the I/O-mask bit is one and the corresponding channel-mask bit is one. Interruptions from channels 0-5 are controlled by channel-mask bits 0-5 in the PSW; an interruption can occur only when the mask bit corresponding to the channel is one. In the BC mode, bits 0-5 in control register 2 do not participate in controlling I/O interruptions; they are, however, preserved in the control register if the corresponding channels are installed.

## Restart Interruption

The restart interruption provides a means for the operator to invoke the execution of a program. The CPU cannot be disabled for this interruption.

A restart interruption causes the old PSW to be stored at location 8 and a new PSW, specifying the start of the program to be executed, to be fetched from location 0. The instruction-length and interruption codes are not stored in the EC mode. In the BC mode, the instruction-length code in the PSW is unpredictable, and zeros are stored in the interruption-code field.

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 pending interruption

conditions for which the CPU is enabled have been taken. If the CPU is in the stopped state, the CPU enters the operating state and exchanges the PSWs without first taking any pending interruptions.

The restart interruption is initiated by activating the restart key.

When the rate control is set to instruction step, it is unpredictable whether restart causes a unit of operation or additional interruptions to be performed after the PSWs have been exchanged.

### Programming Note

In order to perform restart when the CPU is in the check-stop state, the CPU has to be reset. This can be accomplished by means of program reset, which does not clear the contents of registers, including the control registers, but causes the channels to be reset.

## 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 the program-event-recording facility is enabled. 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, normally, requests for I/O and external interruptions remain unaffected.

In the absence of an exigent machine-check condition, requests for interruption 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 taken. 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 continues until no more interruptions are to be serviced. The priority is reevaluated after the new PSW is loaded. Each evaluation is performed taking 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, are taken 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 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 I/O, external, or restart interruption or the stop function.

If the new PSW for a program interruption contains a one in an unassigned bit position of an EC-mode PSW, 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 program 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 is enabled for the interruption just taken. 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 in the EC mode, 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; program reset is required.

Interruptions for all requests for which the CPU is enabled are taken before the CPU is placed in the stopped state. When the CPU is in the stopped state, restart has a higher priority than pending I/O, external, or repressible machine-check conditions.

#### **Programming Note**

The order in which concurrent interruption requests are honored can be changed to some extent by masking.

## Chapter 7. General Instructions

### Contents

Data Format	7-1	MONITOR CALL	7-21
Binary-Integer Representation	7-2	MOVE	7-22
Signed and Unsigned Binary Arithmetic	7-3	MOVE INVERSE	7-22
Signed and Logical Comparison	7-3	MOVE LONG	7-23
Instructions	7-4	MOVE NUMERICS	7-25
ADD	7-7	MOVE WITH OFFSET	7-25
ADD HALFWORD	7-7	MOVE ZONES	7-26
ADD LOGICAL	7-7	MULTIPLY	7-26
AND	7-7	MULTIPLY HALFWORD	7-27
BRANCH AND LINK	7-8	OR	7-27
BRANCH ON CONDITION	7-9	PACK	7-28
BRANCH ON COUNT	7-10	SET PROGRAM MASK	7-28
BRANCH ON INDEX HIGH	7-10	SHIFT LEFT DOUBLE	7-29
BRANCH ON INDEX LOW OR EQUAL	7-10	SHIFT LEFT DOUBLE LOGICAL	7-29
COMPARE	7-11	SHIFT LEFT SINGLE	7-30
COMPARE AND SWAP	7-11	SHIFT LEFT SINGLE LOGICAL	7-30
COMPARE DOUBLE AND SWAP	7-11	SHIFT RIGHT DOUBLE	7-30
COMPARE HALFWORD	7-13	SHIFT RIGHT DOUBLE LOGICAL	7-31
COMPARE LOGICAL	7-13	SHIFT RIGHT SINGLE	7-31
COMPARE LOGICAL CHARACTERS		SHIFT RIGHT SINGLE LOGICAL	7-32
UNDER MASK	7-13	STORE	7-32
COMPARE LOGICAL LONG	7-14	STORE CHARACTER	7-32
CONVERT TO BINARY	7-15	STORE CHARACTERS UNDER MASK	7-32
CONVERT TO DECIMAL	7-16	STORE CLOCK	7-33
DIVIDE	7-16	STORE HALFWORD	7-33
EXCLUSIVE OR	7-17	STORE MULTIPLE	7-33
EXECUTE	7-17	SUBTRACT	7-34
INSERT CHARACTER	7-18	SUBTRACT HALFWORD	7-34
INSERT CHARACTERS UNDER MASK	7-18	SUBTRACT LOGICAL	7-34
LOAD	7-19	SUPERVISOR CALL	7-35
LOAD ADDRESS	7-19	TEST AND SET	7-35
LOAD AND TEST	7-19	TEST UNDER MASK	7-36
LOAD COMPLEMENT	7-20	TRANSLATE	7-36
LOAD HALFWORD	7-20	TRANSLATE AND TEST	7-37
LOAD MULTIPLE	7-20	UNPACK	7-38
LOAD NEGATIVE	7-21		
LOAD POSITIVE	7-21		

This chapter includes all the unprivileged instructions described in this publication, other than the decimal and floating-point instructions.

### 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."

Data resides in general registers or in storage or is

introduced from the instruction stream.

In a storage-to-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. All overlapping fields are considered valid.

## 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 with high-order zeros.

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. 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 inverting each bit of the number, including the sign, and adding a one in the low-order bit position.

This type of number representation can be considered the low-order 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, all these bits are ones. Therefore, when a signed operand must be extended with high-order bits, 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 the set of negative numbers is one larger than the set of positive 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. The number zero consists of all-zero bits.

A signed binary integer of either sign, except for zero and for the maximum negative number, is changed to the number with opposite sign by forming its two's complement. This operation of complementing a number is equivalent to subtracting the number from zero. The complement of zero is zero.

The complement of the maximum negative number cannot be represented in the same number of bits. When an operation, such as a subtraction of the maximum negative number from zero, attempts to produce the 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 minus one. The product of two maximum negative numbers is representable as a double-length positive number.

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 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.

In an arithmetic operation, a carry out of the numeric field of a signed binary integer changes the sign. However, in algebraic left-shifting the sign bit does not change even if significant high-order 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.

## Signed and Unsigned Binary Arithmetic

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 extended on the left to the length of the longer operand by propagating the sign-bit value. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position disagree, an overflow occurs. The sign bit is not changed after the overflow.

Subtraction is performed by adding the one's complement of the second operand and a low-order one to the first operand.

Signed addition and subtraction produce an overflow when the result is outside the range of representation for signed binary integers. Specifically, for ADD and SUBTRACT, 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  $2^{32}$ . An overflow causes a program interruption for fixed-point overflow if it is allowed.

Addition of unsigned binary integers is performed by adding all bits of each operand. When one of the operands is shorter, the shorter operand is extended on the left with zeros. Unsigned binary arithmetic is used in address arithmetic for adding the X, B, and D fields. It is also used to obtain the addresses of the function bytes in the instructions TRANSLATE and TRANSLATE AND TEST. Furthermore, unsigned binary arithmetic is used on 32-bit unsigned binary integers by the instructions ADD LOGICAL and SUBTRACT LOGICAL. Given the same two operands, ADD and ADD LOGICAL produce the same 32-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 the high-order bit position. Such a carry is not necessarily considered an overflow, and no program interruption can occur for ADD LOGICAL.

SUBTRACT LOGICAL differs from ADD LOGICAL in that the one's complement of the second operand and a low-order one are added to the first operand.

## Programming Notes

1. Logical addition and subtraction may be used to program multiple-precision arithmetic. Thus, for multiple-precision binary-integer addition, ADD LOGICAL is used to add the corresponding lower-order parts of the operands. If the condition code indicates a carry, a one is added to the first operand of the next higher pair of integers before adding the second operand. If the integers are signed, the ADD instruction is used on the highest-order parts after propagating any carry. The condition code then indicates any overflow or the proper sign and magnitude of the entire result; an overflow is also indicated by a fixed-point-overflow interruption if it is allowed. If the integers are unsigned, ADD LOGICAL is used throughout.
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 necessarily 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.

The instructions COMPARE and COMPARE HALFWORD are signed-binary comparison operations. These instructions are equivalent to SUBTRACT 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  $2^{32}$  or more. Thus, unlike SUBTRACT, COMPARE can cause no overflow.

Logical comparison is performed by a left-to-right, bit-by-bit comparison of the two operands. The operands are equal when all their bits are equal. When the operands are unequal, the condition is determined by the first unequal pair of bits in a left-to-right comparison of corresponding bit positions: the zero bit indicates the low operand and the one bit the high operand. Since the remaining bit 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 the following table. The table also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

A detailed definition of instruction formats, operand designation and length, and address generation is contained in the section "Instructions" in Chapter 5, "Program Execution." Exceptions to the general rules stated in that section are explicitly identified in the individual instruction descriptions.

Several instruction descriptions in this chapter contain references to other CPUs, even though the

4300 Processors make no provision for multiprocessing, so as to permit the writing of problem-state programs that are compatible with multiprocessing configurations of System/370 (see the section "Problem-State Compatibility" in Chapter 1, "Introduction")

*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, for example, LTR is the mnemonic and R<sub>1</sub>, R<sub>2</sub> the operand designation.*

Name	Mnemonic	Characteristics	Code
ADD	AR RR C	IF R	1A
ADD	A RX C A	IF R	5A
ADD HALFWORD	AH RX C A	IF R	4A
ADD LOGICAL	ALR RR C	R	1E
ADD LOGICAL	AL RX C A	R	5E
AND	NR RR C	R	14
AND	N RX C A	R	54
AND (character)	NC SS C A	ST	D4
AND (immediate)	NI SI C A	ST	94
BRANCH AND LINK	BALR RR	B R	05
BRANCH AND LINK	BAL RX	B R	45
BRANCH ON CONDITION	BCR RR	\$ <sup>1</sup> B	07
BRANCH ON CONDITION	BC RX	B	47
BRANCH ON COUNT	BCTR RR	B R	06
BRANCH ON COUNT	BCT RX	B R	46
BRANCH ON INDEX HIGH	BXH RS	B R	86
BRANCH ON INDEX LOW OR EQUAL	BXLE RS	B R	87
COMPARE	CR RR C		19
COMPARE	C RX C A		59
COMPARE AND SWAP	CS RS C A	SP \$ R	ST BA
COMPARE DOUBLE AND SWAP	CDS RS C A	SP R	ST BB
COMPARE HALFWORD	CH RX C A		49
COMPARE LOGICAL	CLR RR C		15
COMPARE LOGICAL	CL RX C A		55
COMPARE LOGICAL (character)	CLC SS C A		D5
COMPARE LOGICAL (immediate)	CLI SI C A		95
COMPARE LOGICAL CHARACTERS UNDER MASK	CLM RS C A		BD
COMPARE LOGICAL LONG	CLCL RR C A	SP II R	0F
CONVERT TO BINARY	CVB RX A	D IK R	4F
CONVERT TO DECIMAL	CVD RX A		ST 4E
DIVIDE	DR RR	SP IK R	1D
DIVIDE	D RX A	SP IK R	5D
EXCLUSIVE OR	XR RR C		17
EXCLUSIVE OR	X RX C A		57
EXCLUSIVE OR (character)	XC SS C A		ST D7
EXCLUSIVE OR (immediate)	XI SI C A		ST 97
EXECUTE	EX RX A	SP EX	44
INSERT CHARACTER	IC RX A		43
INSERT CHARACTERS UNDER MASK	ICM RS C A		R BF
LOAD	LR RR		R 18
LOAD	L RX A		R 58
LOAD ADDRESS	LA RX		R 41
LOAD AND TEST	LTR RR C		R 12
LOAD COMPLEMENT	LCR RR C	IF	R 13
LOAD HALFWORD	LH RX A		R 48
LOAD MULTIPLE	LM RS A		R 98
LOAD NEGATIVE	LNR RR C		R 11
LOAD POSITIVE	LPR RR C	IF	R 10
MONITOR CALL	MC SI	SP MO	AF
MOVE (character)	MVC SS A		ST D2

Summary of General Instructions (Part 1 of 2)

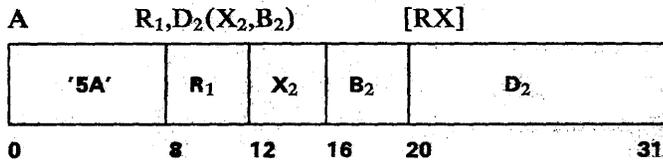
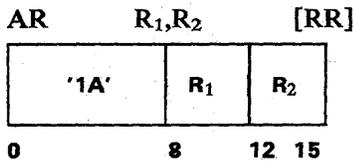
Name	Mnemonic	Characteristics						Code	
MOVE (immediate)	MVI	SI		A			ST	92	
MOVE INVERSE	MVCIN	SS		A			ST	E8	
MOVE LONG	MVCL	RR	C	A	SP	II	R	ST	0E
MOVE NUMERICS	MVN	SS		A				ST	D1
MOVE WITH OFFSET	MVO	SS		A				ST	F1
MOVE ZONES	MVZ	SS		A				ST	D3
MULTIPLY	MR	RR			SP		R		1C
MULTIPLY	M	RX		A	SP		R		5C
MULTIPLY HALFWORD	MH	RX		A			R		4C
OR	OR	RR	C				R		16
OR	O	RX	C	A			R		56
OR (character)	OC	SS	C	A				ST	D6
OR (immediate)	OI	SI	C	A				ST	96
PACK	PACK	SS		A				ST	F2
SET PROGRAM MASK	SPM	RR	L						04
SHIFT LEFT DOUBLE	SLDA	RS	C		SP	IF	R		8F
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS			SP		R		8D
SHIFT LEFT SINGLE	SLA	RS	C			IF	R		8B
SHIFT LEFT SINGLE LOGICAL	SLL	RS					R		89
SHIFT RIGHT DOUBLE	SRDA	RS	C		SP		R		8E
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS			SP		R		8C
SHIFT RIGHT SINGLE	SRA	RS	C				R		8A
SHIFT RIGHT SINGLE LOGICAL	SRL	RS					R		88
STORE	ST	RX		A				ST	50
STORE CHARACTER	STC	RX		A				ST	42
STORE CHARACTERS UNDER MASK	STCM	RS		A				ST	BE
STORE CLOCK	STCK	S	C	A		\$		ST	B205
STORE HALFWORD	STH	RX		A				ST	40
STORE MULTIPLE	STM	RS		A				ST	90
SUBTRACT	SR	RR	C			IF	R		1B
SUBTRACT	S	RX	C	A		IF	R		5B
SUBTRACT HALFWORD	SH	RX	C	A		IF	R		4B
SUBTRACT LOGICAL	SLR	RR	C				R		1F
SUBTRACT LOGICAL	SL	RX	C	A			R		5F
SUPERVISOR CALL	SVC	RR				\$			0A
TEST AND SET	TS	S	C	A		\$		ST	93
TEST UNDER MASK	TM	SI	C	A					91
TRANSLATE	TR	SS		A				ST	DC
TRANSLATE AND TEST	TRT	SS	C	A			R		DD
UNPACK	UNPK	SS		A				ST	F3

**Explanation:**

A	Access exceptions	RS	RS instruction format
B	PER branch event	RX	RX instruction format
C	Condition code is set	S	S instruction format
D	Data exception	SI	SI instruction format
EX	Execute exception	SP	Specification exception
IF	Fixed-point-overflow exception	SS	SS instruction format
II	Interruptible instruction	ST	PER storage-alteration event
IK	Fixed-point-divide exception	\$	Causes serialization
L	New condition code loaded	\$ <sup>1</sup>	Causes serialization when the M <sub>1</sub> and R <sub>2</sub> fields contain all ones and all zeros, respectively.
MO	Monitor event		
R	PER general-register-alteration event		
RR	RR instruction format		

**Summary of General Instructions (Part 2 of 2)**

## ADD



The second operand is added to the first operand, and the sum is placed in the first-operand location. The operands and the sum are treated as 32-bit signed binary integers.

An overflow causes a program interruption when the fixed-point-overflow mask bit is one.

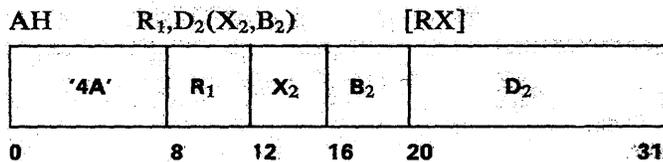
### Resulting Condition Code:

- 0 Sum is zero
- 1 Sum is less than zero
- 2 Sum is greater than zero
- 3 Overflow

### Program Exceptions:

Access (fetch, operand 2 of A only)  
Fixed-Point Overflow

## ADD HALFWORD



The second operand is added to the first operand, and the sum is placed in the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand and the sum are treated as 32-bit signed binary integers.

An overflow causes a program interruption when the fixed-point-overflow mask bit is one.

### Resulting Condition Code:

- 0 Sum is zero
- 1 Sum is less than zero
- 2 Sum is greater than zero
- 3 Overflow

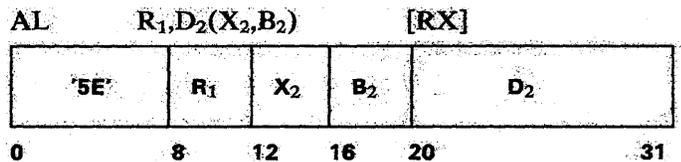
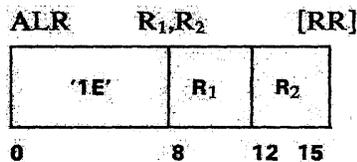
### Program Exceptions:

Access (fetch, operand 2)  
Fixed-Point Overflow

### Programming Note

An example of the use of ADD HALFWORD is given in Appendix A.

## ADD LOGICAL



The second operand is added to the first operand, and the sum is placed in the first-operand location. The operands and the sum are treated as 32-bit unsigned binary integers.

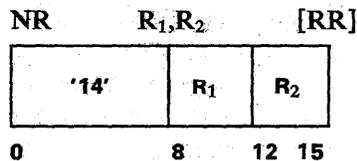
### Resulting Condition Code:

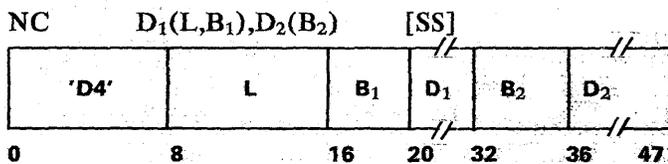
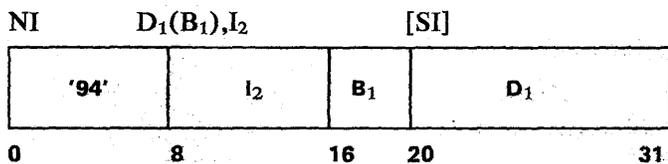
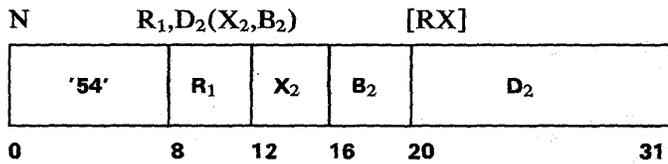
- 0 Sum is zero, with no carry
- 1 Sum is not zero, with no carry
- 2 Sum is zero, with carry
- 3 Sum is not zero, with carry

### Program Exceptions:

Access (fetch, operand 2 of AL only)

## AND





The AND of the first and second operands is placed in the first-operand location.

The connective AND is applied to the operands bit by bit. A bit position in the result is set to one if the corresponding bit positions in both operands contain ones; otherwise, the result bit is set to zero.

For 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 the necessary operand byte is fetched.

For NI, the first operand is one byte in length, and only one byte is stored.

**Resulting Condition Code:**

- 0 Result is zero
- 1 Result is not zero
- 2 -
- 3 -

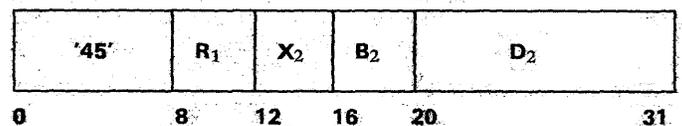
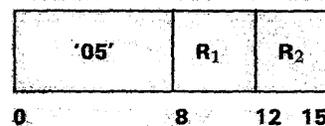
**Program Exceptions:**

Access (fetch, operand 2, N and NC; fetch and store, operand 1, NI and NC)

**Programming Notes**

1. An example of the use of the AND instruction is given in Appendix A.
2. The instruction AND may be used to set a bit to zero.
3. Accesses to the first operand of NI 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 may also be updating the location. An example of this effect is shown for the instruction OR (OI) in the section "Multiprocessing Examples" in Appendix A.

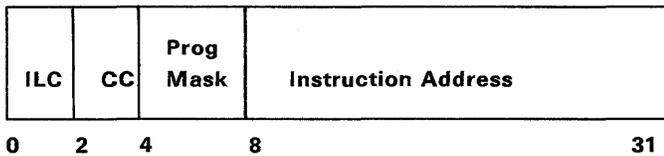
**BRANCH AND LINK**



Information from the current PSW, including the updated instruction address, is loaded as link information in the general register designated by R<sub>1</sub>. Subsequently, the instruction address is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, bits 8-31 of the general register designated by R<sub>2</sub> are used as the branch address; however, when the R<sub>2</sub> field contains zeros, the operation is performed without branching. The branch address is computed before the link information is loaded.

The link information consists of the instruction-length code (ILC), the condition code (CC), the program mask bits, and the updated instruction address, arranged in the following format:



The instruction-length code is 1 or 2.

**Condition Code:** The code remains unchanged.

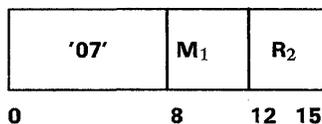
**Program Exceptions:** None.

### Programming Notes

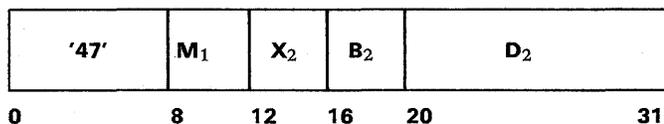
1. An example of the use of BRANCH AND LINK is given in Appendix A.
2. When the R<sub>2</sub> field in the RR format contains all zeros, the link information is loaded without branching.
3. When BRANCH AND LINK is the subject instruction of EXECUTE, the instruction-length code is 2.
4. The format and the contents of the link information do not depend on whether the PSW specifies the EC or BC mode. In both modes, the link information is in the format of the rightmost 32 bit positions of the BC-mode PSW.

### BRANCH ON CONDITION

BCR M<sub>1</sub>,R<sub>2</sub> [RR]



BC M<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>) [RX]



The instruction address in the current PSW is replaced by the branch address if the condition code has one of the values specified by M<sub>1</sub>; 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, bits 8-31 of the general register specified by R<sub>2</sub> are used as the branch address; however, when the R<sub>2</sub> field

contains zeros, the operation is performed without branching.

The M<sub>1</sub> 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	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<sub>1</sub> and R<sub>2</sub> fields of BCR are all ones and all zeros, respectively, a serialization function is performed. CPU operation is delayed until all previous accesses by this CPU to storage have been completed, as observed by channels and other CPUs. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed.

**Condition Code:** The code remains unchanged.

**Program Exceptions:** None.

### Programming Notes

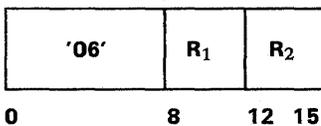
1. An example of the use of BRANCH ON CONDITION is given in Appendix A.
2. When a branch is to depend on more than one condition, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made when the condition code is 0 or 1.
3. When all four mask bits are zero or when the R<sub>2</sub> 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<sub>2</sub> 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 function is actually required.

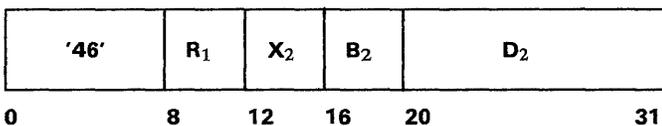
- Note that the relation between the RR and RX formats in branch-address specification is not the same as in operand-address specification. For branch instructions in the RX format, the branch address is the address specified by X<sub>2</sub>, B<sub>2</sub>, and D<sub>2</sub>; in the RR format, the branch address is contained in the register specified by R<sub>2</sub>. For operands, the address specified by X<sub>2</sub>, B<sub>2</sub>, and D<sub>2</sub> is the operand address, but the register specified by R<sub>2</sub> contains the operand itself.

### BRANCH ON COUNT

BCTR R<sub>1</sub>,R<sub>2</sub> [RR]



BCT R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>) [RX]



A one is subtracted from the first operand, and the result is placed in the first-operand location. The first operand and result are treated as 32-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 format, the second-operand address is used as the branch address. In the RR format, the contents of bit positions 8-31 of the general register specified by R<sub>2</sub> are used as the branch address; however, when the R<sub>2</sub> field contains zeros, the operation is performed without branching.

The branch address is computed before the counting operation.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

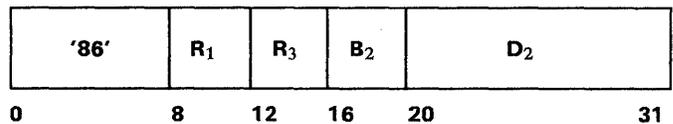
None.

### Programming Notes

- An example of the use of BRANCH ON COUNT is given in Appendix A.
- 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 minus one and causes branching to be executed; an initial count of minus one results in minus 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, because of the number range, an initial count of minus 2<sup>31</sup> results in a positive value of 2<sup>31</sup>-1.
- Counting is performed without branching when the R<sub>2</sub> field in the RR format contains zero.

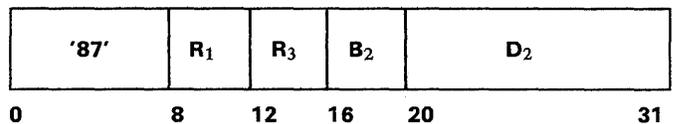
### BRANCH ON INDEX HIGH

BXH R<sub>1</sub>,R<sub>3</sub>,D<sub>2</sub>(B<sub>2</sub>) [RS]



### BRANCH ON INDEX LOW OR EQUAL

BXLE R<sub>1</sub>,R<sub>3</sub>,D<sub>2</sub>(B<sub>2</sub>) [RS]



An increment is added to the first operand, and the sum is compared with a comparand. Subsequently, the sum is placed in the first-operand location. The second-operand address is used as a branch address.

For BXH, 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 BXLE, 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, the even and odd registers of the pair specified by the  $R_3$  field are used as the increment and the comparand, respectively. When the  $R_3$  field is odd, the register specified by the  $R_3$  field is used as both the increment and the comparand. The branch address is computed before the addition and comparison.

For purposes of the addition and comparison, all operands and results are treated as 32-bit signed binary integers. Overflow caused by the addition is ignored. When the first-operand and comparand locations coincide, the original register contents are used as the comparand.

The sum is placed in the first-operand location regardless of whether the branch is taken.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

None.

#### Programming Notes

1. An example of the use of BRANCH ON INDEX HIGH is given in Appendix A.
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 register  $R_1$  by an arbitrary amount.

### COMPARE

CR  $R_1, R_2$  [RR]

'19'	$R_1$	$R_2$
------	-------	-------

0 8 12 15

C  $R_1, D_2(X_2, B_2)$  [RX]

'59'	$R_1$	$X_2$	$B_2$	$D_2$
------	-------	-------	-------	-------

0 8 12 16 20 31

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.

#### Resulting Condition Code:

- 0 Operands are equal
- 1 First operand is low
- 2 First operand is high
- 3 -

#### Program Exceptions:

Access (fetch, operand 2 of C only)

### COMPARE AND SWAP

CS  $R_1, R_3, D_2(B_2)$  [RS]

'BA'	$R_1$	$R_3$	$B_2$	$D_2$
------	-------	-------	-------	-------

0 8 12 16 20 31

### COMPARE DOUBLE AND SWAP

CDS  $R_1, R_3, D_2(B_2)$  [RS]

'BB'	$R_1$	$R_3$	$B_2$	$D_2$
------	-------	-------	-------	-------

0 8 12 16 20 31

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 CS, the first and third operands are 32 bits in length, with each operand occupying a general register. The second operand is a word in storage.

For CDS, the first and third operands are 64 bits in length, with each operand occupying an even-odd pair of general registers. The second operand is a doubleword in storage.

When the result of the comparison is unequal, the second-operand location remains unchanged. No attempt to store occurs, and no change-bit and store-protection actions are taken.

When an equal comparison occurs, no access by another CPU to the second-operand location is permitted between the moment that the second operand is fetched for comparison and the moment that the third operand is stored at the second-operand location.

Serialization is performed before the operand is fetched, and again after the operation is completed. CPU operation is delayed until all previous accesses

by this CPU to storage have been completed, as observed by channels and other CPUs, and then the second operand is fetched. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed, including placing the result value, if any, in storage, as observed by channels and other CPUs.

The second operand of CS must be designated on a word boundary. The R<sub>1</sub> and R<sub>3</sub> fields for CDS must each designate an even register, and the second operand for CDS must be designated on a doubleword boundary. 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)  
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.
- 2. The instruction CS can be used by programs sharing common storage areas in either a multiprogramming or multiprocessing environment. Two examples are:
  - a. By performing the following procedure, a program can modify the contents of a storage location even though the possibility exists that the program may be interrupted by another program that will update the location or even though the possibility exists that another CPU 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 the instruction CS is executed with the R<sub>1</sub> field designating the register that contains the original value and the R<sub>3</sub> field designating the register that contains the updated value. If condition code 0 is set, the update has been successful. If condition code 1 is set, the

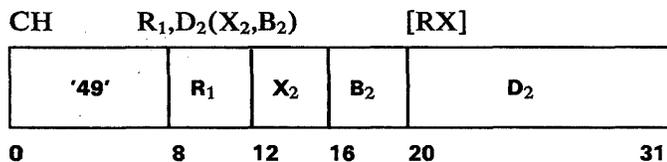
storage location no longer contains the original value, the update has not been successful, and the general register designated by the R<sub>1</sub> field of the CS instruction contains the new current value of the storage location. When condition code 1 is set, the program can repeat the procedure using the new current value.

- b. The instruction CS can be used for controlled sharing of a common storage area in a manner similar to that described in the programming note under TEST AND SET, but it provides the added capability of leaving a message 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, a negative value indicating that the area is in use, 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 programs desiring to seize the area can use CS to update the control word to indicate that the area is in use or to add messages to the list. The single program which has seized the area can also safely use CS to remove messages from the list.
- 3. The instruction CDS can be used in a manner similar to that described for CS. 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 2b above. If multiple programs are permitted to add and delete messages by using CS, there is a possibility the list will be incorrectly updated. This would occur if, after one program has fetched the address of the most recent message in order to remove the message, another program removes the first two messages and then adds the first message back into the chain. The first 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 CDS 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 program is delayed while changes exactly equal in number to a multiple of 2<sup>32</sup> take

place *and* only if the last change places the original message address in the control word.

- The instructions CS and CDS do not interlock against storage accesses by channels. Therefore, the instructions should not be used to update a location which is in an I/O input area, since the input data may be lost.

### COMPARE HALFWORD



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. The first operand is treated as a 32-bit signed binary integer.

#### Resulting Condition Code:

- 0 Operands are equal
- 1 First operand is low
- 2 First operand is high
- 3 -

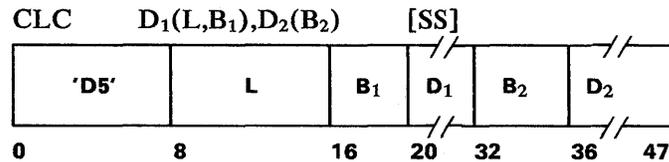
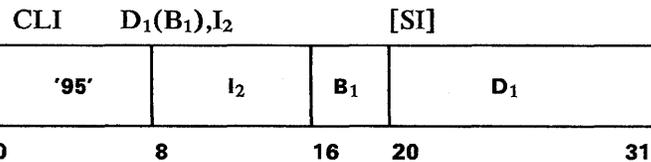
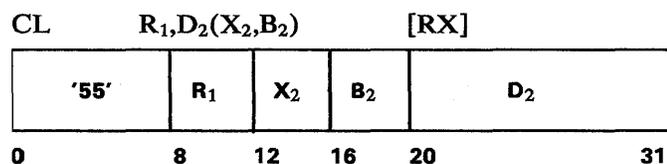
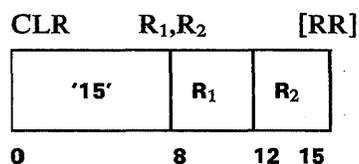
#### Program Exceptions:

Access (fetch, operand 2)

#### Programming Note

An example of the use of COMPARE HALFWORD is given in Appendix A.

### COMPARE LOGICAL



The first operand is compared with the second operand, and the result is indicated in the condition code.

The comparison proceeds left to right, bit for bit, and ends as soon as an inequality is found or an end of the fields is reached. For CL and CLC, access exceptions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte.

#### Resulting Condition Code:

- 0 Operands are equal
- 1 First operand is low
- 2 First operand is high
- 3 -

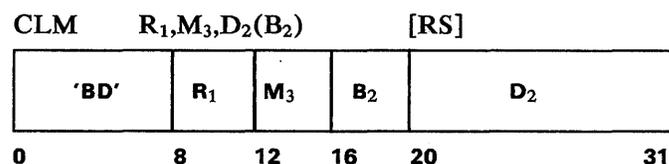
#### Program Exceptions:

Access (fetch, operand 2, CL and CLC; fetch, operand 1, CLI and CLC)

#### Programming Notes

- Examples of the use of the COMPARE LOGICAL instructions are given in Appendix A.
- The COMPARE LOGICAL instructions treat all bits of each operand alike as part of a field of unstructured logical data. For CLC, the comparison may extend to field lengths of 256 bytes.

### COMPARE LOGICAL CHARACTERS UNDER MASK



The first operand is compared with the second operand 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 the four bytes, left to right, of the general register designated by the  $R_1$  field. 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, bit for bit, 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.

**Resulting Condition Code:**

- 0 Selected bytes are equal, or mask is zero
- 1 Selected field of first operand is low
- 2 Selected field of first operand is high
- 3 -

**Program Exceptions:**

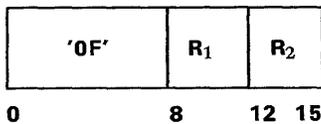
Access (fetch, operand 2)

**Programming Note**

An example of the use of COMPARE LOGICAL CHARACTERS UNDER MASK is given in Appendix A.

**COMPARE LOGICAL LONG**

CLCL  $R_1, R_2$  [RR]

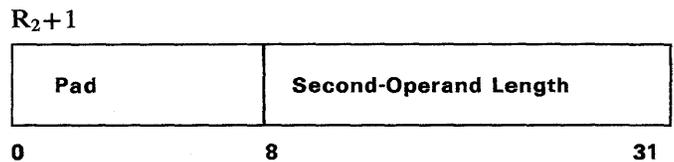
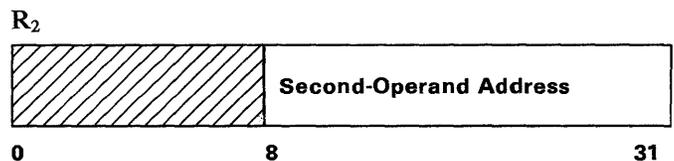
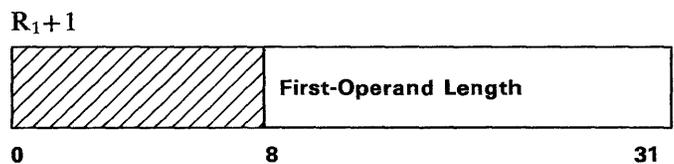
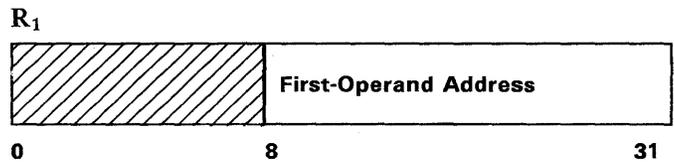


The first operand is compared with the second operand, and the result is indicated in the condition code. The shorter operand is considered to be extended on the right with padding bytes.

The  $R_1$  and  $R_2$  fields each specify 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 bits 8-31 of the general registers specified by the  $R_1$  and  $R_2$  fields, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 8-31 of general registers  $R_1+1$  and  $R_2+1$ , respectively. Bit positions 0-7 of register  $R_2+1$  contain the padding byte. The contents of bit positions 0-7 of registers  $R_1$ ,  $R_2$ , and  $R_1+1$  are ignored.

Graphically, the contents of the registers just described are as follows:



The comparison proceeds left to right, bit for bit, 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 causes termination, the contents of registers  $R_1+1$  and  $R_2+1$  are decremented by the number of bytes compared, and the contents of registers  $R_1$  and  $R_2$  are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. The high-order bits which are not part of the address in registers  $R_1$  and  $R_2$  are set to zeros; the contents of the high-order byte of 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 registers  $R_1$  and  $R_2$  at completion identify the first unequal byte in each operand. The lengths in bit positions 8-31 of 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 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. The bits which are not part of the address in registers  $R_1$  and  $R_2$  are set to zeros, including the case when one or both of the initial length values are zero. The contents of bit positions 0-7 of 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.

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 are equal, or both have zero length
- 1 First operand is low
- 2 First operand is high
- 3 -

**Program Exceptions:**

Access (fetch, operands 1 and 2)  
Specification

**Programming Notes**

1. An example of the use of COMPARE LOGICAL LONG is given in Appendix A.
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, and, in the absence of dynamic modification of the operand area by another CPU or a channel, 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. Another programming note concerning interruptible instructions is included in the section "Point of Interruption" in Chapter 6, "Interruptions."
4. Special precautions should be taken when COMPARE LOGICAL LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.

**CONVERT TO BINARY**

CVB	$R_1, D_2(X_2, B_2)$			[RX]
'4F'	$R_1$	$X_2$	$B_2$	$D_2$
0	8	12	16	20
				24
				28
				31

The radix of the second operand is changed from decimal to binary, and the result is placed in the first-operand location.

The second operand occupies eight bytes in storage and is treated as packed decimal data, as described in Chapter 8, "Decimal Instructions." It is checked for valid sign and digit codes, and a data exception is recognized when an invalid code is detected.

The result of the conversion is a 32-bit signed binary integer, which is placed in the general register specified by  $R_1$ . The maximum positive number that can be converted and still be contained in a 32-bit register 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 low-order bits of the binary result in the register, and a fixed-point-divide exception is recognized.

**Condition Code:** The code remains unchanged.

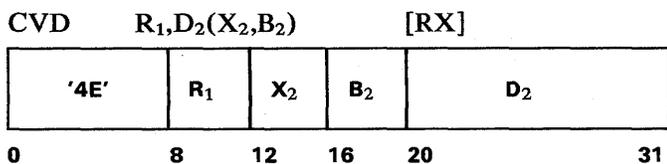
**Program Exceptions:**

Access (fetch, operand 2)  
Data  
Fixed-Point Divide

**Programming Notes**

1. An example of the use of CONVERT TO BINARY is given in Appendix A.
2. When the second operand is negative, the result is in two's-complement notation.

**CONVERT TO DECIMAL**



The radix of the first operand is changed from binary to decimal, and the result is stored at the second-operand location. The first operand is treated as a 32-bit signed binary integer.

The result occupies eight bytes in storage and is in the format for packed decimal data, as described in Chapter 8, "Decimal Instructions." The low-order four bits of the result represent the sign. A positive sign is encoded as 1100; a negative sign is encoded as 1101.

**Condition Code:** The code remains unchanged.

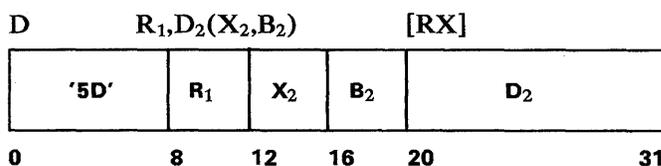
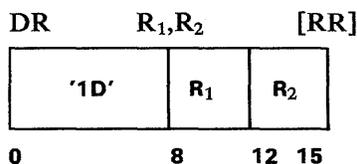
**Program Exceptions:**

Access (store, operand 2)

**Programming Notes**

1. An example of the use of CONVERT TO DECIMAL is given in Appendix A.
2. 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.

**DIVIDE**



The doubleword first operand (the dividend) is divided by the second operand (the divisor), and the remainder and the quotient are placed in the first-operand location.

The R<sub>1</sub> field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R<sub>1</sub> is odd, a specification exception is recognized.

The dividend is treated as a 64-bit signed binary integer. The divisor, the remainder, and the quotient are treated as 32-bit signed binary integers. The remainder and quotient replace the dividend in the pair of registers specified by the R<sub>1</sub> field. The remainder is placed in the even-numbered register, and the quotient is placed in the odd-numbered register.

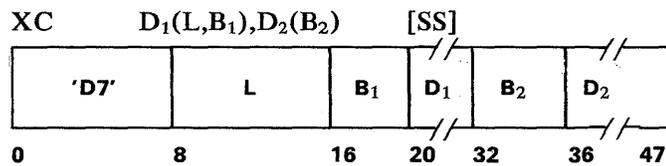
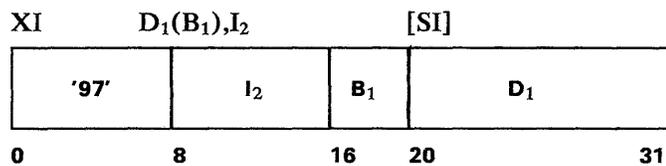
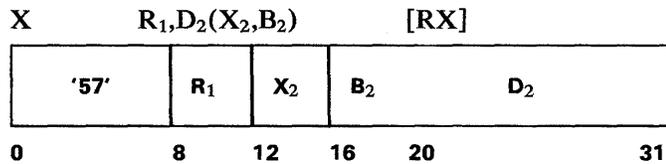
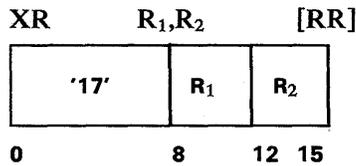
The sign of the quotient is determined by the rules of algebra. The remainder has the same sign as the dividend, except that a zero quotient or a zero remainder is always positive. When the magnitudes of the dividend and divisor are such that the quotient cannot be expressed by a 32-bit signed binary integer, a fixed-point-divide exception is recognized, and the operation is suppressed.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Access (fetch, operand 2 of D only)  
Fixed-Point Divide  
Specification

## EXCLUSIVE OR



The EXCLUSIVE OR of the first and second operands is placed in the first-operand location.

The connective EXCLUSIVE OR is applied to the operands bit by bit. A bit position in the result is set to one if the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

For 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 the necessary operand byte is fetched.

For XI, the first operand is one byte in length, and only one byte is stored.

### Resulting Condition Code:

- 0 Result is zero
- 1 Result is not zero
- 2 -
- 3 -

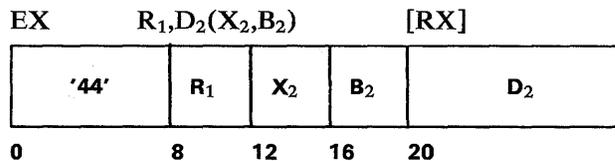
### Program Exceptions:

Access (fetch, operand 2, X and XC; fetch and store, operand 1, XI and XC)

### Programming Notes

1. An example of the use of EXCLUSIVE OR is given in Appendix A.
2. The instruction EXCLUSIVE OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.
3. A field EXCLUSIVE-ORed with itself becomes all zeros.
4. For XR, the sequence A EXCLUSIVE-OR B, B EXCLUSIVE-OR A, A EXCLUSIVE-OR 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 XI and 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, the instruction EXCLUSIVE OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for the instruction OR (OI) in the section "Multiprocessing Examples," in Appendix A.

## EXECUTE



The single instruction at the second-operand address is modified by the contents of the general register specified by R<sub>1</sub>, and the resulting target instruction is executed.

When the R<sub>1</sub> field is not zero, bits 8-15 of the instruction designated by the second-operand address are ORed with bits 24-31 of the register specified by R<sub>1</sub>. The ORing does not change either the contents of the register specified by R<sub>1</sub> or the instruction in storage, and it is effective only for the interpretation of the instruction to be executed. When the R<sub>1</sub> 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 of the current PSW is increased by the length of EXECUTE. This updated address and the instruction-length code of EXECUTE are used 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 of the current PSW is replaced by the branch address specified by the target instruction.

When the target instruction is in turn an EXECUTE, an execute exception is recognized.

The effective address of EXECUTE must be even; otherwise, a specification exception is recognized. Access exceptions are not recognized for the second-operand address when the address is odd.

**Condition Code:** The code may be set by the target instruction.

**Program Exceptions:**

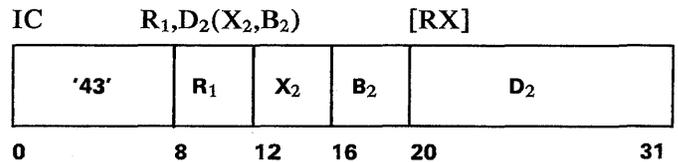
Access (fetch, target instruction)  
Execute  
Specification

**Programming Notes**

1. An example of the use of EXECUTE is given in Appendix A.
2. The ORing of eight bits from the general register with the designated instruction permits indirect length, index, mask, immediate-data, and register specification.
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 or by the target instruction.
5. When an interruptible instruction is made the target of EXECUTE, the program normally should not designate any register updated by the interruptible instruction as the R<sub>1</sub>, X<sub>2</sub>, or B<sub>2</sub> register for EXECUTE, since 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 EXECUTE. Similarly, the

program should normally not let the destination field of an interruptible instruction include the location of the EXECUTE, since the new contents of the location may be interpreted when resuming execution.

**INSERT CHARACTER**



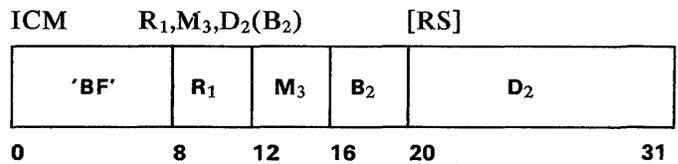
The byte at the second-operand location is inserted into bit positions 24-31 of the general register designated by the R<sub>1</sub> field. The remaining bits in the register remain unchanged.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Access (fetch, operand 2)

**INSERT CHARACTERS UNDER MASK**



Bytes from contiguous locations beginning at the second-operand address are inserted into the first-operand location under control of a mask.

The contents of the M<sub>3</sub> field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of the general register designated by the R<sub>1</sub> field. 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.

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 all inserted bits are not 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.

**Resulting Condition Code:**

- 0 All inserted bits are zeros, or mask is zero
- 1 Leftmost bit of the inserted field is one
- 2 Leftmost bit of the inserted field is zero, and not all inserted bits are zeros
- 3 -

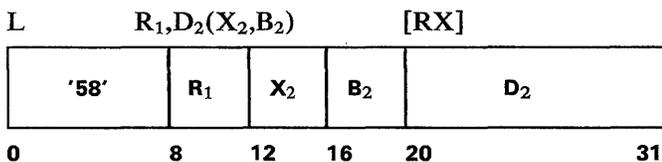
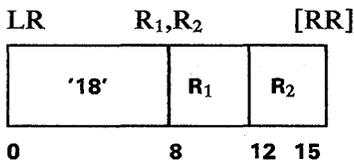
**Program Exceptions:**

Access (fetch, operand 2)

**Programming Notes**

1. Examples of the use of INSERT CHARACTERS UNDER MASK are given in Appendix A.
2. The condition code for INSERT CHARACTERS UNDER MASK (ICM) is defined such that, when the mask is 1111, the instruction causes the same condition code to be set as for LOAD AND TEST. Thus, the instruction may be used as a storage-to-register load-and-test operation.
3. An ICM instruction with a mask of 1111 or 0001 performs a function similar to that of a LOAD (L) or INSERT CHARACTER (IC), respectively, with the exception of the condition-code setting. However, the performance of ICM may be slower.

**LOAD**



The second operand is placed unchanged in the first-operand location.

**Condition Code:** The code remains unchanged.

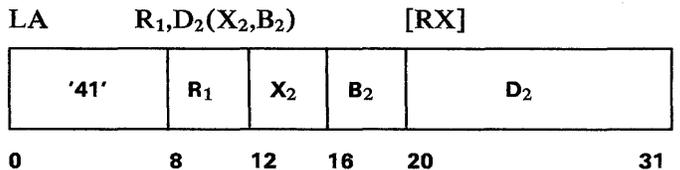
**Program Exceptions:**

Access (fetch, operand 2 of L only)

**Programming Note**

An example of the use of LOAD is given in Appendix A.

**LOAD ADDRESS**



The address specified by the X<sub>2</sub>, B<sub>2</sub>, and D<sub>2</sub> fields is placed in bit positions 8-31 of the general register specified by the R<sub>1</sub> field. Bits 0-7 of the register are set to zeros. The address computation follows the rules for address arithmetic.

No storage references for operands take place, and the address is not inspected for access exceptions.

**Condition Code:** The code remains unchanged.

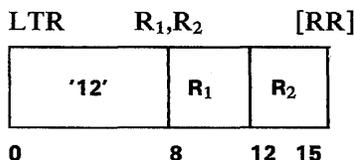
**Program Exceptions:**

None.

**Programming Notes**

1. An example of the use of the LOAD ADDRESS instruction is given in Appendix A.
2. The same general register may be specified by the R<sub>1</sub>, X<sub>2</sub>, and B<sub>2</sub> fields, except that general register 0 can be specified only by the R<sub>1</sub> field. In this manner, it is possible to increment the low-order 24 bits of a general register, other than register 0, by the contents of the D<sub>2</sub> field of the instruction. The register to be incremented should be specified by R<sub>1</sub> and by either X<sub>2</sub> (with B<sub>2</sub> set to zero) or B<sub>2</sub> (with X<sub>2</sub> set to zero).

**LOAD AND TEST**



The second operand is placed unchanged in the first-operand location, and the sign and magnitude of the second operand, treated as a 32-bit signed binary integer, are indicated in the condition code.

**Resulting Condition Code:**

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

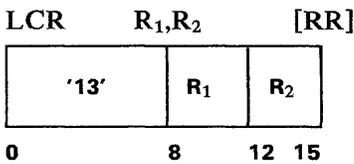
**Program Exceptions:**

None.

**Programming Note**

When the R<sub>1</sub> and R<sub>2</sub> fields designate the same register, the operation is equivalent to a test without data movement.

**LOAD COMPLEMENT**



The two's complement of the second operand is placed in the first-operand location. The second operand and result are treated as 32-bit signed binary integers.

An overflow causes a program interruption when the fixed-point-overflow mask bit is one.

**Resulting Condition Code:**

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Overflow

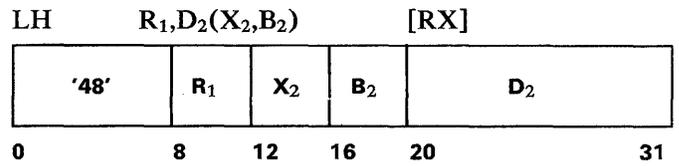
**Program Exceptions:**

Fixed-Point Overflow

**Programming Note**

The operation complements all numbers. Zero and the maximum negative number remain unchanged. An overflow condition occurs when the maximum negative number is complemented.

**LOAD HALFWORD**



The second operand is extended to a 32-bit signed binary integer and placed in the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed binary integer. The second operand is extended by propagating the sign-bit value through the 16 high-order bit positions.

**Condition Code:** The code remains unchanged.

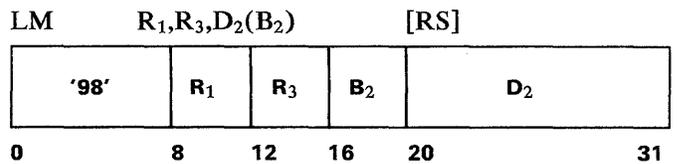
**Program Exceptions:**

Access (fetch, operand 2)

**Programming Note**

An example of the use of LOAD HALFWORD is given in Appendix A.

**LOAD MULTIPLE**



The set of general registers starting with the register specified by R<sub>1</sub> and ending with the register specified by R<sub>3</sub> is loaded from storage beginning at the location designated by the second-operand address and continuing through as many locations as needed.

The general registers are loaded in the ascending order of their register numbers, starting with the register specified by R<sub>1</sub> and continuing up to and including the register specified by R<sub>3</sub>, with register 0 following register 15.

**Condition Code:** The code remains unchanged.

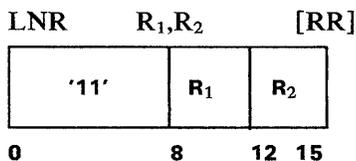
**Program Exceptions:**

Access (fetch, operand 2)

### Programming Note

All combinations of register numbers specified by  $R_1$  and  $R_2$  are valid. When the register numbers are equal, only four bytes are transmitted. When the number specified by  $R_2$  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 in the first-operand location. The second operand and result are treated as 32-bit signed binary integers.

#### Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 -
- 3 -

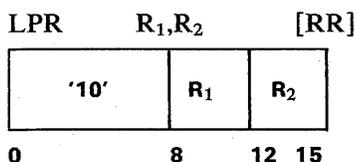
#### Program Exceptions:

None.

### Programming Note

The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged.

### LOAD POSITIVE



The absolute value of the second operand is placed in the first-operand location. The second operand and the result are treated as 32-bit signed binary integers.

An overflow causes a program interruption when the fixed-point-overflow mask bit is one.

### Resulting Condition Code:

- 0 Result is zero
- 1 -
- 2 Result is greater than zero
- 3 Overflow

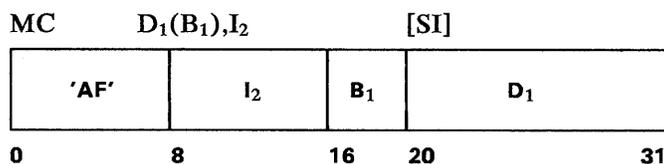
#### Program Exceptions:

Fixed-Point Overflow

### Programming Note

The operation complements negative numbers; positive numbers and zero remain unchanged. An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged.

### MONITOR CALL



A program interruption is caused if the appropriate monitor-mask bit in control register 8 is one.

The monitor-mask bits are in bit positions 16-31 of control register 8. The mask bits, bits 16-31, correspond to monitor classes 0-15, respectively.

Bit positions 12-15 in the  $I_2$  field contain a binary number specifying one of 16 monitoring classes. When the monitor-mask bit corresponding to the class specified by the  $I_2$  field is one, a monitor-event program interruption occurs. The contents of the  $I_2$  field are stored at location 149, with zeros stored at location 148. Bit 9 of the program-interruption code is set to one.

The first-operand address is not used to address data; instead, the address specified by the  $B_1$  and  $D_1$  fields forms the monitor code, which is placed in the word at location 156. Address computation follows the rules of address arithmetic; bits 0-7 are set to zeros.

When the monitor-mask bit corresponding to the class specified by bits 12-15 of the instruction is zero, no interruption occurs, and the instruction is executed as a no-operation.

Bit positions 8-11 of the instruction must contain zeros; otherwise, a specification exception is recognized.

**Condition Code:** The code remains unchanged.

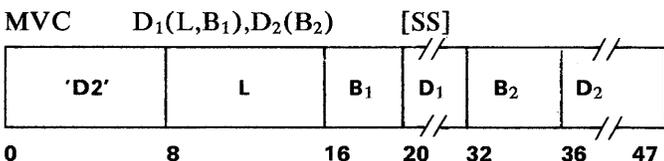
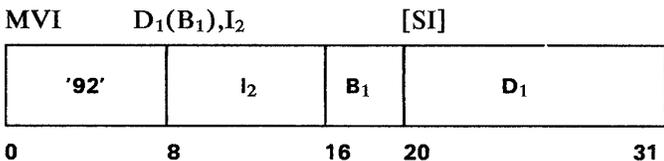
**Program Exceptions:**

Monitor Event Specification

**Programming Notes**

1. The MONITOR CALL instruction 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. This function may be useful in performing various measurement functions; specifically, by implanting MONITOR CALL instructions within the programs, tracing information can be generated indicating which programs were executed, counting information can be generated indicating how often particular programs are used, and timing information can be generated indicating how long a particular program required for execution.
2. The monitor masks provide a means of disallowing all interruptions due to MONITOR CALL or allowing monitoring for all or selected classes.
3. The monitor code provides a means of associating descriptive information, in addition to the class number, with each MONITOR CALL instruction. Without the use of a base register, up to 4,096 distinct monitor codes can be associated with a monitoring interruption. With the base register designated by a nonzero value in the B<sub>1</sub> field, each monitoring interruption can be identified by a 24-bit code.

**MOVE**



The second operand is placed in the first-operand location.

For 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 the necessary operand byte is fetched.

For MVI, the first operand is one byte in length, and only one byte is stored.

**Condition Code:** The code remains unchanged.

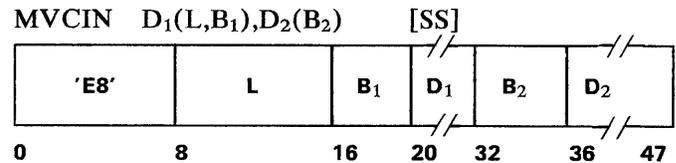
**Program Exceptions:**

Access (fetch, operand 2 of MVC; store, operand 1, MVI and MVC)

**Programming Notes**

1. Examples of the use of the MOVE instructions are given in Appendix A.
2. 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 in 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 0 to location 16,777,215. The first operand may wrap around from location 16,777,215 to location 0.

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)

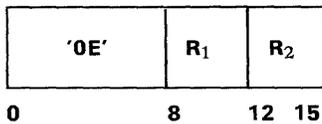
**Programming Notes**

1. The contents of each byte moved remain unchanged.
2. 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.

**Note:** The MOVE INVERSE instruction is also available in the System/370 mode. This will be incorporated in a future update of the IBM System/370 Principles of Operation.

**MOVE LONG**

MVCL R<sub>1</sub>,R<sub>2</sub> [RR]

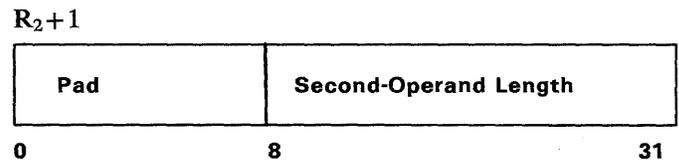
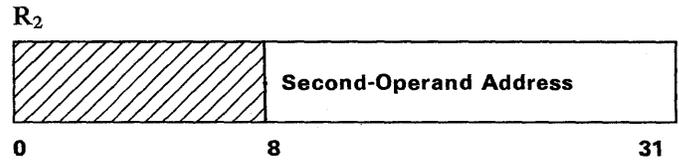
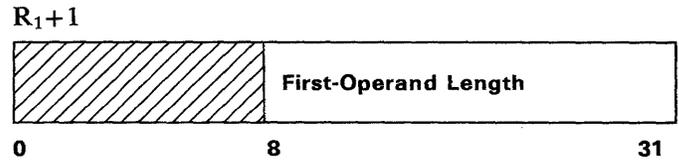
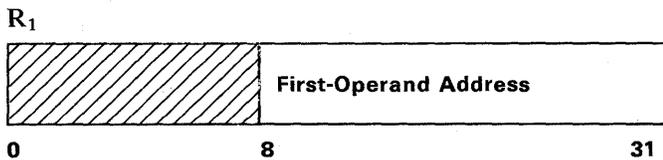


The second operand is placed in the first-operand location, provided overlapping of operand locations does not affect the final contents of the first-operand location. The remaining rightmost byte positions, if any, of the first-operand location are filled with padding bytes.

The R<sub>1</sub> and R<sub>2</sub> fields each specify 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 bits 8-31 of the general registers specified by the R<sub>1</sub> and R<sub>2</sub> fields, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 8-31 of general registers R<sub>1</sub>+1 and R<sub>2</sub>+1, respectively. Bit positions 0-7 of register R<sub>2</sub>+1 contain the padding byte. The contents of bit positions 0-7 of registers R<sub>1</sub>, R<sub>2</sub>, and R<sub>1</sub>+1 are ignored.

Graphically, the contents of the registers just described are as follows:



The movement starts at the left end of both fields and proceeds to the right. The operation is ended when the number of bytes specified by bit positions 8-31 of register R<sub>1</sub>+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 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. 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 16,777,215 to location 0, operand bytes in locations up to and including 16,777,215 are considered to be to the left of bytes in locations from 0 up.

When the length specified by bit positions 8-31 of register R<sub>1</sub>+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 causes termination, the contents of registers R<sub>1</sub>+1 and R<sub>2</sub>+1 are decremented by the number of bytes moved, and the contents of register R<sub>1</sub> and R<sub>2</sub> are

incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. The high-order bits which are not part of the address in registers  $R_1$  and  $R_2$  are set to zeros; the contents of the high-order byte of 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 register  $R_2+1$  is 0, the address in register  $R_2$  is incremented by the original contents of register  $R_2+1$ , and 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, 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.

At the completion of the operation, the length in register  $R_1+1$  is decremented by the number of bytes stored at the first-operand location, and the address in register  $R_1$  is incremented by the same amount. The length in register  $R_2+1$  is decremented by the number of bytes moved out of the second-operand location, and the address in register  $R_2$  is incremented by the same amount. The bits which are not part of the address in registers  $R_1$  and  $R_2$  are set to zeros, including the case when one or both of the original length values are zeros or when condition code 3 is set. The contents of bit positions 0-7 of 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. 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 is not recognized, and no change bits are set.

#### **Resulting Condition Code:**

- 0 First-operand and second-operand lengths are equal
- 1 First-operand length is low
- 2 First-operand length is high
- 3 No movement performed because of destructive overlap

#### **Program Exceptions:**

Access (fetch, operand 2; store, operand 1)  
Specification

#### **Programming Notes**

1. The instruction MOVE LONG may be used for clearing storage by setting the padding byte to zero and the second-operand length to zero.
2. 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 MVCL, 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 actually occur on one or more System/370 models.
  - a. When a protection exception occurs on a 2,048-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, in the case of reconfigurable storage, an addressing exception on a block does not necessarily suppress processing of subsequent blocks which are addressable.
  - b. The model may update the general registers only when an I/O interruption occurs, or a program interruption occurs which is required to nullify or suppress. Thus, if after a move into several blocks of the first operand, an addressing or protection exception occurs, the registers remain unchanged.
3. When the first-operand length is zero, the operation consists in setting the condition code and setting the high-order bytes of registers  $R_1$  and  $R_2$  to zero.
4. When the contents of the  $R_1$  and  $R_2$  fields are the same, the operation proceeds the same way as when two distinct pairs of registers having the same contents are specified. Condition code 0 is set.
5. The following is a detailed description of those cases in which movement takes place, that is, where destructive overlap does not exist. Depending on whether the second operand wraps

around from location 16,777,215 to location 0, movement takes place in the following cases:

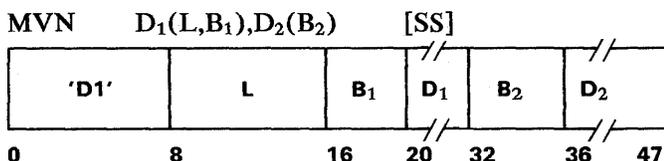
- a. When the second operand does not wrap around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, *or* if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating 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.

6. Special precautions must be taken if MOVE LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.
7. 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 or on the interval timer not being updated during the execution of the instruction. Similarly, the program should normally not let the first operand of MOVE LONG include the location of the instruction since the new contents of the location may be interpreted for a resumption after an interruption, or the instruction may be refetched without an interruption.
8. Further programming notes concerning interruptible instructions are included in the section "Interruptible Instructions" in Chapter 6, "Interruptions."

### MOVE NUMERICS



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 the necessary operand byte is fetched.

**Condition Code:** The code remains unchanged.

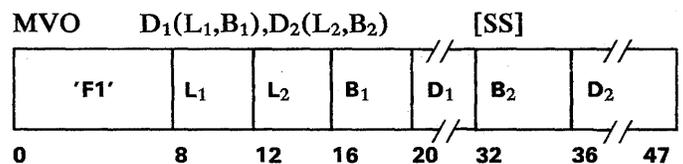
### Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)

### Programming Notes

1. An example of the use of MOVE NUMERICS is given in Appendix A.
2. MVN 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 MVN 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.

### 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 in 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 and each result byte were stored immediately after the necessary operand bytes are fetched. The left digit of each second-operand byte remains available for the next result byte and is not refetched.

**Condition Code:** The code remains unchanged.

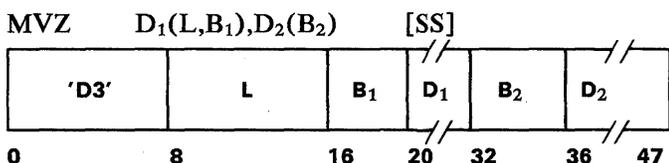
**Program Exceptions:**

Access (fetch, operand 2; fetch and store, operand 1)

**Programming Notes**

1. An example of the use of MOVE WITH OFFSET is given in Appendix A.
2. Access to the rightmost byte of the first operand of MVO 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.
3. MVO 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.

**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.

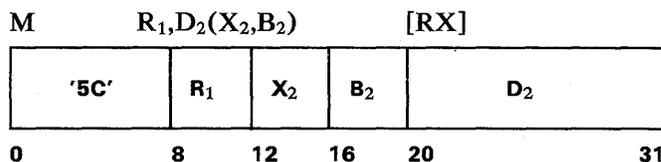
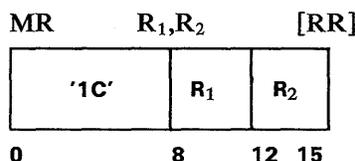
**Program Exceptions:**

Access (fetch, operand 2; fetch and store, operand 1)

**Programming Notes**

1. An example of the use of MOVE ZONES is given in Appendix A.
2. MVZ 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 MVZ 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.

**MULTIPLY**



The second word of the first operand (multiplicand) is multiplied by the second operand (multiplier), and the doubleword product is placed at the first-operand location.

The R<sub>1</sub> field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R<sub>1</sub> is odd, a specification exception is recognized.

Both the multiplicand and multiplier are treated as 32-bit signed binary integers. The multiplicand is taken from the odd-numbered register of the pair specified by the R<sub>1</sub> field. The contents of the even-numbered register are ignored. The product is a 64-bit signed binary integer, which replaces the

contents of the even-odd pair of general registers specified by the  $R_1$  field. 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.

**Condition Code:** The code remains unchanged.

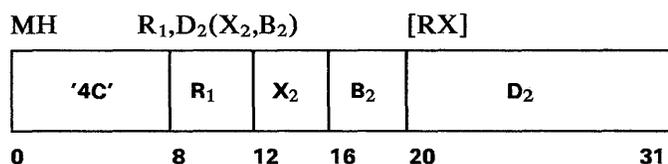
**Program Exceptions:**

Access (fetch, operand 2 of M only)  
Specification

**Programming Notes**

1. An example of the use of MULTIPLY is given in Appendix A.
2. The significant part of the product usually occupies 62 bits or fewer. Only when two maximum negative numbers are multiplied are 63 significant product bits formed.

**MULTIPLY HALFWORD**



The first operand (multiplicand) is multiplied by the second operand (multiplier), and the product is placed at the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed binary integer.

The multiplicand is treated as a 32-bit signed binary integer and is replaced by the low-order 32 bits of the signed-binary-integer product. The bits to the left of the 32 low-order bits 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.

**Condition Code:** The code remains unchanged.

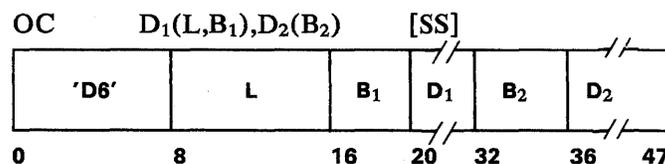
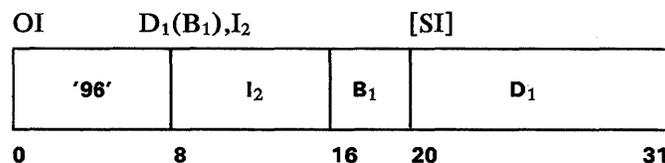
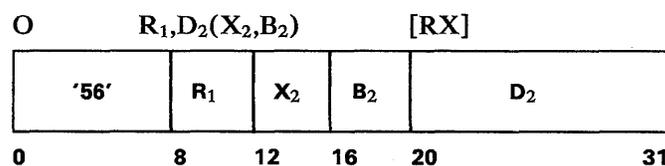
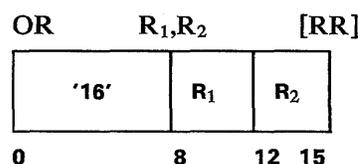
**Program Exceptions:**

Access (fetch, operand 2)

**Programming Notes**

1. An example of the use of MULTIPLY HALFWORD is given in Appendix A.
2. The significant part of the product usually occupies 46 bits or fewer. Only when two maximum negative numbers are multiplied are 47 significant product bits formed. Since the low-order 32 bits of the product are stored unchanged, 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 low-order part of the product in two's-complement notation.

**OR**



The OR of the first and second operands is placed in the first-operand location.

The connective OR is applied to the operands bit by bit. A bit position in the result is 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 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 the necessary operand byte is fetched.

For OI, the first operand is only one byte in length, and only one byte is stored.

**Resulting Condition Code:**

- 0 Result is zero
- 1 Result is not zero
- 2 -
- 3 -

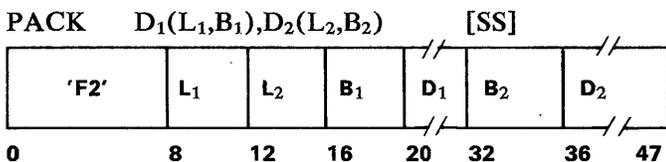
**Program Exceptions:**

Access (fetch, operand 2, O and OC; fetch and store, operand 1, OI and OC)

**Programming Notes**

1. Examples of the use of the OR instructions are given in Appendix A.
2. The instruction OR may be used to set a bit to one.
3. Accesses to the first operand of OI and 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, the instruction OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown in the section "Multiprocessing Examples," in Appendix A.

**PACK**



The format of the second operand is changed from zoned to packed, and the result is placed in the first-operand location. The zoned and packed formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the zoned format. All zones are ignored, except the zone in the rightmost byte, which is treated as a sign. The sign is placed in the rightmost four bits of the rightmost byte, and the digits are placed adjacent to the sign

and to each other in the remainder of the result field. The sign and digits are moved unchanged to the first operand and 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 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 the necessary operand bytes are fetched. 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.

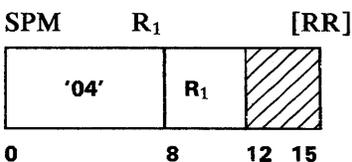
**Program Exceptions:**

Access (fetch, operand 2; store, operand 1)

**Programming Notes**

1. An example of the use of PACK is given in Appendix A.
2. The PACK instruction may be used to interchange the two hexadecimal digits in one byte by specifying a zero in the L<sub>1</sub> and L<sub>2</sub> fields and the same address for both operands.
3. To remove the zones of all bytes of a field, including the rightmost byte, both operands must be extended on the right with a dummy byte, which subsequently is ignored in the result field.

**SET PROGRAM MASK**



The contents of the general register specified by the R<sub>1</sub> field are used to set the condition code and the program mask of the current PSW. Bits 12-15 of the instruction are ignored.

Bits 2 and 3 of the register specified by the R<sub>1</sub> field replace the condition code, and bits 4-7 replace

the program mask. Bits 0, 1, and 8-31 of the register specified by the R<sub>1</sub> field are ignored.

**Resulting Condition Code:**

- 0 Bit 2 is zero, and bit 3 is zero
- 1 Bit 2 is zero, and bit 3 is one
- 2 Bit 2 is one, and bit 3 is zero
- 3 Bit 2 is one, and bit 3 is one

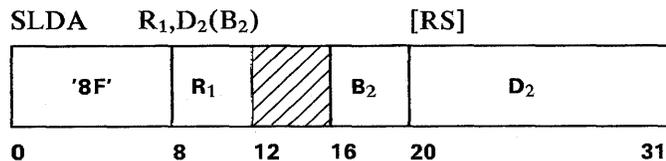
**Program Exceptions:**

None.

**Programming Notes**

1. Bits 2-7 of the general register may have been loaded from the PSW by BRANCH AND LINK.
2. The instruction permits setting of the condition code and the mask bits in either the problem or 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 exponent-underflow and significance masks also determine the result which is obtained.

**SHIFT LEFT DOUBLE**



The double-length numeric part of the first operand is shifted left the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The R<sub>1</sub> field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R<sub>1</sub> is odd, a specification exception is recognized.

The second-operand address is not used to address data; its low-order 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 position of the even register remains unchanged. The leftmost position of the odd 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 register positions on the right.

If one or more bits unlike the sign bit are shifted out of bit position 1 of the even register, an overflow occurs. The overflow causes a program interruption when the fixed-point-overflow mask bit is one.

**Resulting Condition Code:**

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Overflow

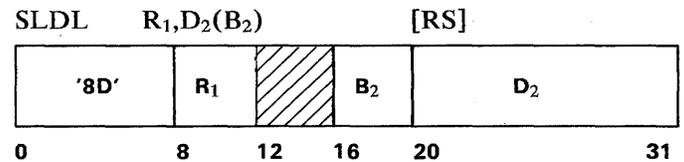
**Program Exceptions:**

Fixed-Point Overflow Specification

**Programming Notes**

1. An example of the use of SHIFT LEFT DOUBLE is given in Appendix A.
2. The eight shift instructions provide the following three pairs of alternatives: left or right, single or double, and signed or logical. 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. A zero in the B<sub>2</sub> field indicates the absence of indirect shift specification.

**SHIFT LEFT DOUBLE LOGICAL**



The double-length first operand is shifted left the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The R<sub>1</sub> field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When R<sub>1</sub> is odd, a specification exception is recognized.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

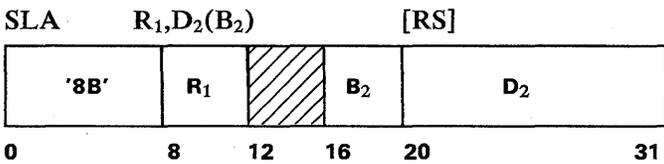
All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 0 of the even-numbered register are not inspected and are lost. Zeros are supplied to the vacated register positions on the right.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Specification

**SHIFT LEFT SINGLE**



The numeric part of the first operand is shifted left the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The second-operand address is not used to address data; its low-order 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 32-bit signed binary integer. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the left shift. Zeros are supplied to the vacated register positions on the right.

If one or more bits unlike the sign bit are shifted out of bit position 1, an overflow occurs. The overflow causes a program interruption when the fixed-point-overflow mask bit is one.

**Resulting Condition Code:**

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Overflow

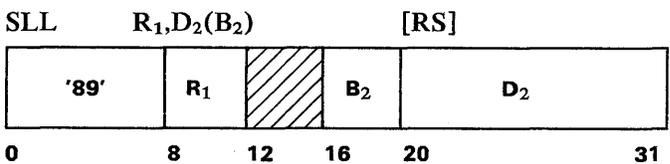
**Program Exceptions:**

Fixed-Point Overflow

**Programming Notes**

1. An example of the use of SHIFT LEFT SINGLE is given in Appendix A.
2. For numbers with an absolute value of less than  $2^{30}$ , a left shift of one bit position is equivalent to multiplying the number by two.
3. 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.

**SHIFT LEFT SINGLE LOGICAL**



The first operand is shifted left the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

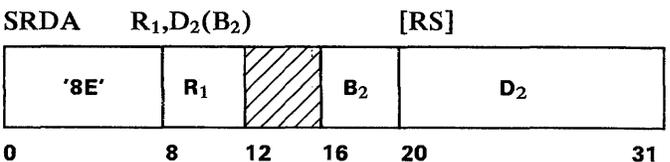
All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 0 are not inspected and are lost. Zeros are supplied to the vacated register positions on the right.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

None.

**SHIFT RIGHT DOUBLE**



The double-length numeric part of the first operand is shifted right the number of places specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The  $R_1$  field of the instruction specifies an even-odd pair of general registers and must designate an

even-numbered register. When  $R_1$  is odd, a specification exception is recognized.

The second-operand address is not used to address data; its low-order 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 position of the even register remains unchanged. The leftmost position of the odd 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 31 of the odd-numbered register are not inspected and are lost. Bits equal to the sign are supplied to the vacated register positions on the left.

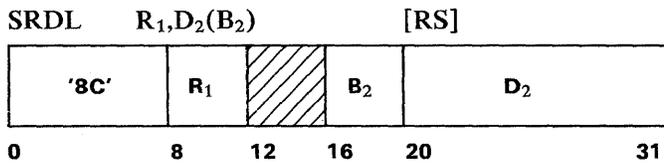
**Resulting Condition Code:**

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

**Program Exceptions:**

Specification

**SHIFT RIGHT DOUBLE LOGICAL**



The double-length first operand is shifted right the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The  $R_1$  field of the instruction specifies an even-odd pair of general registers and must designate an even-numbered register. When  $R_1$  is odd, a specification exception is recognized.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

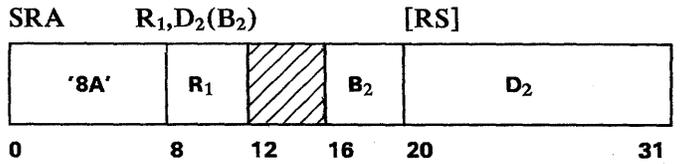
All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 31 of the odd-numbered register are not inspected and are lost. Zeros are supplied to the vacated register positions on the left.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Specification

**SHIFT RIGHT SINGLE**



The numeric part of the first operand is shifted right the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The second-operand address is not used to address data; its low-order 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 32-bit signed binary integer. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the right shift. Bits shifted out of bit position 31 are not inspected and are lost. Bits equal to the sign are supplied to the vacated register positions on the left.

**Resulting Condition Code:**

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

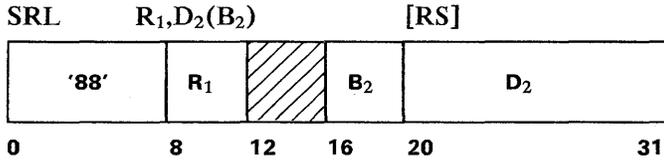
**Program Exceptions:**

None.

**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. 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.

## SHIFT RIGHT SINGLE LOGICAL



The first operand is shifted right the number of bits specified by the second-operand address. Bits 12-15 of the instruction are ignored.

The second-operand address is not used to address data; its low-order six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

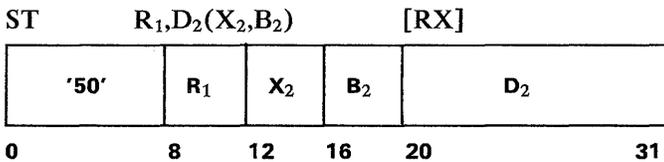
All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 31 are not inspected and are lost. Zeros are supplied to the vacated register positions on the left.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

None.

## STORE



The first operand is stored at the second-operand location.

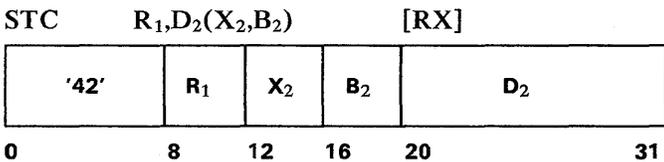
The 32 bits in the general register are placed unchanged at the second-operand location.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Access (store, operand 2)

## STORE CHARACTER



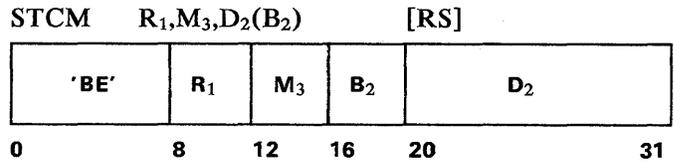
Bits 24-31 of the general register designated by the  $R_1$  field are placed unchanged at the second-operand location. The second operand is one byte in length.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Access (store, operand 2)

## STORE CHARACTERS UNDER MASK



Bytes selected from the first operand under control of a mask are placed in contiguous byte locations beginning at the second-operand address.

The contents of the  $M_3$  field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of the general register designated by the  $R_1$  field. The bytes corresponding to ones in the mask are placed in the same order in 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, and no access exceptions are recognized.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Access (store, operand 2)

## Programming Notes

1. An example of the use of STORE CHARACTERS UNDER MASK is given in Appendix A.
2. STCM with a mask of 0111 may be used to store a three-byte address, for example, in modifying the address in a CCW.
3. STCM with a mask of 1111, 0011, or 0001



specified by  $R_1$  and continuing up to and including the register specified by  $R_3$ , with register 0 following register 15.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

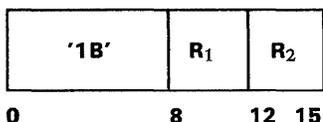
Access (store, operand 2)

**Programming Note**

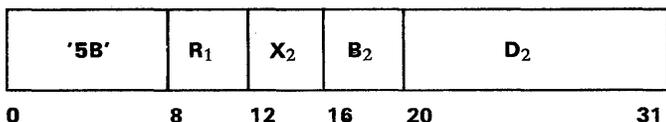
An example of the use of STORE MULTIPLE is given in Appendix A.

**SUBTRACT**

SR  $R_1, R_2$  [RR]



S  $R_1, D_2(X_2, B_2)$  [RX]



The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The operands and the difference are treated as 32-bit signed binary integers.

An overflow causes a program interruption when the fixed-point-overflow mask bit is one.

**Resulting Condition Code:**

- 0 Difference is zero
- 1 Difference is less than zero
- 2 Difference is greater than zero
- 3 Overflow

**Program Exceptions:**

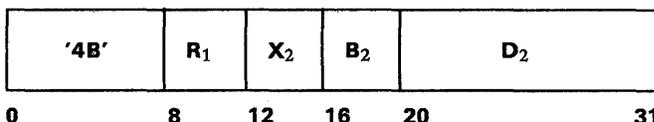
Access (fetch, operand 2 of S only)  
Fixed-Point Overflow

**Programming Notes**

1. When, in the RR format, the  $R_1$  and  $R_2$  fields designate the same register, subtracting is equivalent to clearing the register.
2. Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

**SUBTRACT HALFWORD**

SH  $R_1, D_2(X_2, B_2)$  [RX]



The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand and the difference are treated as 32-bit signed binary integers.

An overflow causes a program interruption when the fixed-point-overflow mask bit is one.

**Resulting Condition Code:**

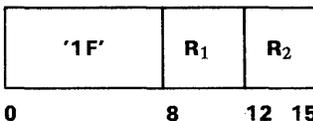
- 0 Difference is zero
- 1 Difference is less than zero
- 2 Difference is greater than zero
- 3 Overflow

**Program Exceptions:**

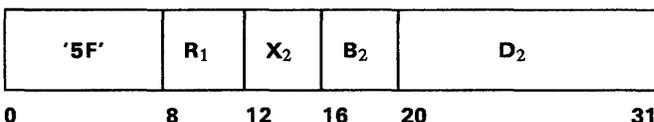
Access (fetch, operand 2)  
Fixed-Point Overflow

**SUBTRACT LOGICAL**

SLR  $R_1, R_2$  [RR]



SL  $R_1, D_2(X_2, B_2)$  [RX]



The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The operands and the difference are treated as 32-bit unsigned binary integers.

**Resulting Condition Code:**

- 0 -
- 1 Difference is not zero, with no carry
- 2 Difference is zero, with carry
- 3 Difference is not zero, with carry

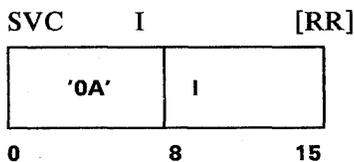
**Program Exceptions:**

Access (fetch, operand 2 of SL only)

**Programming Notes**

1. Logical subtraction is performed by adding the one's complement of the second operand and a low-order one to the first operand. The use of the one's complement and the low-order one instead of the two's complement of the second operand results in a carry when subtracting zero.
2. SUBTRACT LOGICAL differs from SUBTRACT only in the meaning of the condition code and in the absence of the interruption for overflow.
3. A zero difference is always accompanied by a carry out of the high-order bit position.
4. The condition-code setting for SUBTRACT LOGICAL can also be interpreted as indicating the presence and absence of a borrow, as follows:
  - 1 Difference is not zero, with borrow
  - 2 Difference is zero, with no borrow
  - 3 Difference is not zero, with no borrow

**SUPERVISOR CALL**



The instruction causes a supervisor-call interruption, with the I field of the instruction providing the interruption code.

Bits 8-15 of the instruction, with eight high-order zeros appended, are placed in the supervisor-call interruption code that is stored in the course of the interruption. The old PSW is stored at location 32, and a new PSW is obtained from location 96. The

instruction is valid in both the problem and supervisor states.

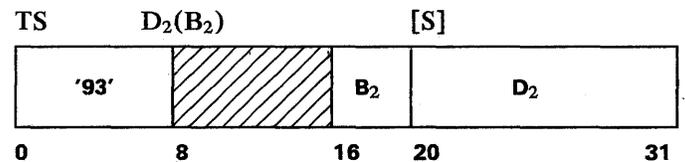
A serialization function is performed. CPU operation is delayed until all previous storage accesses by this CPU to storage have been completed, as observed by channels and other CPUs. No subsequent instructions or their operands are accessed by this CPU until the execution of this instruction is completed.

**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 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 position 0. No access by another CPU to this location is permitted between the moment of fetching and the moment of storing all ones.

A serialization function is performed before the byte is fetched and again after the storing of all ones. CPU operation is delayed until all previous accesses by this CPU to storage have been completed, as observed by channels and other CPUs, and then the byte is fetched. No subsequent instructions or their operands are accessed by this CPU until the all-ones value has been placed in storage, as observed by channels and other CPUs.

**Resulting Condition Code:**

- 0 Leftmost bit of byte specified was zero
- 1 Leftmost bit of byte specified was one
- 2 -
- 3 -

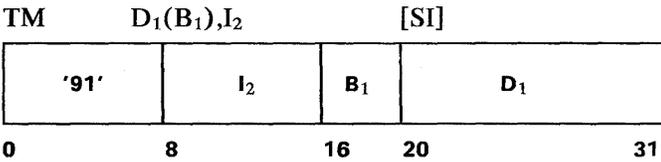
**Program Exceptions:**

Access (fetch and store, operand 2)

**Programming Notes**

1. TEST AND SET may be used for controlled sharing of a common storage area by more than one program. To accomplish this, bit position 0 of a byte must be designated as the control bit. The desired interlock can be achieved by establishing a program convention in which a zero in the bit position indicates that the common area is available but a one means that the area is being used. Each using program then must examine this byte by means of TEST AND SET before making access to the common area. If the test sets condition code 0, the area is available for use; if it sets condition code 1, the area cannot be used. Because TEST AND SET permits no other CPU access to the test byte between the moment of fetching (for testing) and the moment of storing all ones (setting), the possibility is eliminated of a second program testing the byte before the first program is able to set it.
2. It should be noted that TEST AND SET does not interlock against storage accesses by channels.

**TEST UNDER MASK**



A mask is used to select bits of the first operand, and the result is indicated in the condition code.

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, the code is set to 1.

Access exceptions associated with the storage operand are recognized for one byte, even when the mask is all zeros.

**Resulting Condition Code:**

- 0 Selected bits all zeros; or the mask is all zeros
- 1 Selected bits mixed zeros and ones
- 2 -
- 3 Selected bits all ones

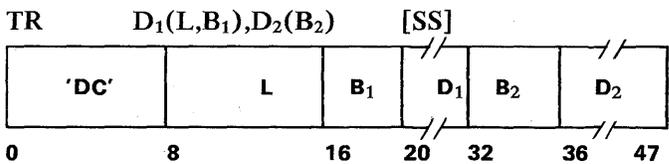
**Program Exceptions:**

Access (fetch, operand 1)

**Programming Note**

An example of the use of TEST UNDER MASK is given in Appendix A.

**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 designates 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 high-order zeros. 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 the corresponding function byte is fetched.

Access exceptions are recognized only for those bytes in the second operand which are actually required.

**Condition Code:** The code remains unchanged.

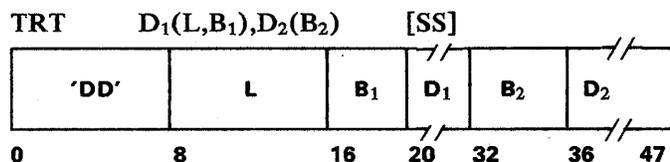
**Program Exceptions:**

Access (fetch, operand 2; fetch and store, operand 1)

### Programming Notes

1. An example of the use of TRANSLATE is given in Appendix A.
2. The instruction 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. The fetch and subsequent store accesses to a particular byte in the first-operand field do not necessarily occur one immediately after the other.
5. 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.
6. Because of pretesting, significant performance degradation is possible when the second-operand address of TRANSLATE designates a location less than 256 bytes to the left of a 2,048-byte boundary.

### 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 nonzero function byte is inserted in general register 2, and the related argument address in general register 1.

The L field designates the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding from left to right. The first operand remains unchanged in storage. Fetching of the function byte from the list is performed as in TRANSLATE. 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 related argument address is inserted in the rightmost 24 bits of general register 1. This address points to the argument byte last translated. The function byte is inserted in the low-order eight bits of general register 2. Bits 0-7 of register 1 and bits 0-23 of register 2 remain unchanged.

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.

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 -

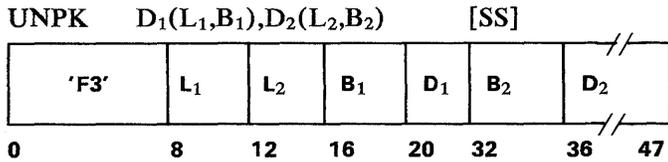
#### Program Exceptions:

Access (fetch, operands 1 and 2)

### Programming Notes

1. An example of the use of TRANSLATE AND TEST is given in Appendix A.
2. The instruction 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.

## UNPACK



The format of the second operand is changed from packed to zoned, and the result is placed in the first-operand location. The packed and zoned formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as having the packed format. Its digits and sign are placed unchanged in the first-operand location, using the zoned format. Zones with coding of 1111 are supplied for all bytes except the low-order byte, 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

each result byte were stored immediately after the necessary operand byte is fetched. 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 the leftmost four bits of the byte remain available and are not refetched. Thus, two result bytes are 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 UNPACK is given in Appendix A.
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 position of the first operand must be to the right of the rightmost position 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 low-order positions of the two operands may coincide.

# Chapter 8. Decimal Instructions

## Contents

Data Format 8-1  
 Zoned Format 8-1  
 Packed Format 8-1  
 Number Representation 8-2  
 Instructions 8-2  
 ADD DECIMAL 8-3  
 COMPARE DECIMAL 8-4  
 DIVIDE DECIMAL 8-4

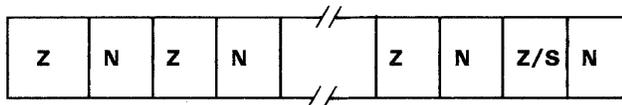
EDIT 8-5  
 EDIT AND MARK 8-8  
 MULTIPLY DECIMAL 8-8  
 SHIFT AND ROUND DECIMAL 8-8  
 SUBTRACT DECIMAL 8-10  
 ZERO AND ADD 8-10

Decimal instructions provide arithmetic, shifting, and editing operations on decimal data.

### Data Format

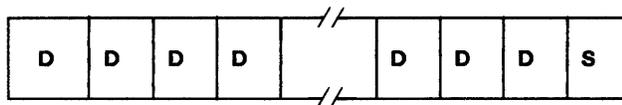
Decimal operands reside in storage and may be in either the zoned or packed format.

#### Zoned Format



In the zoned format, the rightmost four bits of a byte are called the numeric (N) and normally comprise a code representing a decimal digit. The leftmost four bits of a byte are called the zone (Z), except for the rightmost byte of the field, where these bits may be treated either as a zone or as a sign (S) code.

#### Packed Format



In the packed format, each byte contains two decimal digits (D), except for the rightmost byte, which contains a sign to the right of a decimal digit. The digit and sign codes each comprise four bits.

Arithmetic and shifting are performed with operands in the packed format and generate results in the packed format. Decimal numbers in the zoned format are represented as part of an alphanumeric character set, which includes also alphabetic and

special characters. The zoned format is usually produced by input devices, such as keyboards, and is usually used for displaying or printing decimal data on output devices.

The instructions MOVE ZONES and MOVE NUMERICS are provided for operating on data in the zoned format. Two instructions are provided for converting data between the zoned and packed formats: the PACK instruction transforms zoned data into packed data, and UNPACK performs the reverse transformation. Two instructions are provided for conversion between the packed-decimal and binary formats. The CONVERT TO BINARY instruction converts packed decimal to binary, and CONVERT TO DECIMAL converts binary to packed decimal. These six instructions are not considered to be decimal instructions and are described in Chapter 7, "General Instructions." The instructions EDIT and EDIT AND MARK may also be used to change data from the packed to the zoned format.

Decimal operands occupy fields in storage that start on a byte boundary. For all decimal instructions other than EDIT and EDIT AND MARK, the operands are in the packed format and are composed of one to sixteen 8-bit bytes. For the two editing instructions, operands of up to 256 bytes in length can be designated.

For the decimal-arithmetic instructions, the lengths of the two operands specified in the instruction need not be the same. If necessary, the operands are considered to be extended with zeros to the left of the high-order digit. Results, however, never exceed the first-operand field length as specified in the instruction. When a carry or high-order significant digits are lost because the first-operand field is too small, a program interruption for decimal overflow

occurs, provided the decimal-overflow mask bit is one. For the two editing instructions, only one operand (the pattern) has an explicitly specified length; the other operand (the source) is considered to have as many digits as necessary for the completion of the operation.

The operand fields in decimal instructions, other than EDIT and EDIT AND MARK, should not overlap at all or should have coincident rightmost bytes. In ZERO AND ADD, the field may also overlap in such a manner that the rightmost byte of the first operand 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 code configurations for digits and signs are verified during the performance of the arithmetic, improperly overlapping fields are recognized as data exceptions. In editing, overlapping operands yield unpredictable results.

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, intermediate values may be placed in the result field that may differ from the original operand and final result values. Thus, in a multiprocessing system, 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.

## Number Representation

Packed decimal numbers are represented as right-aligned true integers with a plus or minus sign.

The digits 0-9 have the binary encoding 0000-1001. The codes 1010-1111 are invalid as digit codes and are interpreted as sign codes, with 1010, 1100, 1110, and 1111 recognized as plus and with 1011 and 1101

recognized as minus. The codes 0000-1001 are invalid as sign codes. A data exception is recognized when an invalid code is detected. The operation is terminated, except when the sign position contains an invalid sign code, in which case the operation is suppressed.

Although alternate encoding of the sign in an operand is accepted, the preferred sign codes are always generated for the results of decimal arithmetic and shifting operations (for the first-operand field of ADD DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, SHIFT AND ROUND DECIMAL, SUBTRACT DECIMAL, and ZERO AND ADD). These codes are plus, 1100, and minus, 1101. They are provided even when the operand value is otherwise unchanged, such as when adding zero to a number or when shifting the field by a zero amount. The editing instruction, as well as UNPACK, generates the zone code 1111.

## Instructions

The decimal instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of Decimal Instructions." The figure also indicates when the condition code is set 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.*

Name	Mnemonic	Characteristics							Code
ADD DECIMAL	AP	SS	C	A		D	DF	ST	FA
COMPARE DECIMAL	CP	SS	C	A		D			F9
DIVIDE DECIMAL	DP	SS		A	SP	D	DK	ST	FD
EDIT	ED	SS	C	A		D		ST	DE
EDIT AND MARK	EDMK	SS	C	A		D		R ST	DF
MULTIPLY DECIMAL	MP	SS		A	SP	D		ST	FC
SHIFT AND ROUND DECIMAL	SRP	SS	C	A		D	DF	ST	F0
SUBTRACT DECIMAL	SP	SS	C	A		D	DF	ST	FB
ZERO AND ADD	ZAP	SS	C	A		D	DF	ST	F8

**Explanation:**

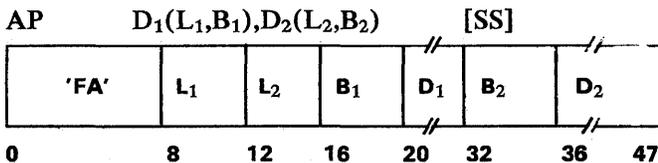
- A Access exceptions
- C Condition code is set
- D Data exception
- DF Decimal-overflow exception
- DK Decimal-divide exception
- R PER general-register-alteration event
- SP Specification exception
- SS SS instruction format
- ST PER storage-alteration event

Summary of Decimal Instructions

**Programming Note**

The moving and logical-comparing instructions may also be used in decimal calculations.

**ADD DECIMAL**



The second operand is added to the first operand, and the sum is placed in the first-operand location.

Addition is algebraic, taking into account the signs and all digits of both operands. All sign and digit codes are checked for validity. If necessary, high-order zeros are supplied for either operand. When the first-operand field is too short to contain all significant digits of the sum, a decimal overflow occurs, and a program interruption is taken, provided that the decimal-overflow mask bit is one.

Overflow has two possible causes. The first occurs when a carry out of the high-order digit position of the result field is lost. The second occurs when the second-operand field is longer than the first-operand field and significant result digits are lost. The field lengths alone are not an indication of overflow.

The first-operand and second-operand fields may overlap when their low-order bytes coincide; therefore, it is possible to add a number to itself.

The sign of the sum is determined by the rules of algebra. When the operation is completed without an overflow, a zero sum has a positive sign, but when high-order digits are lost because of an overflow, a zero sum may be either positive or negative, as determined by what the sign of the correct sum would have been.

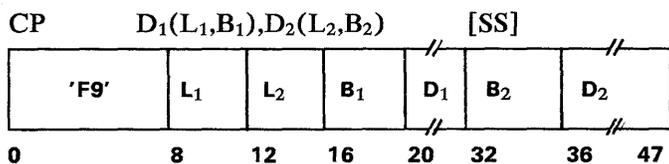
**Resulting Condition Code:**

- 0 Sum is zero
- 1 Sum is less than zero
- 2 Sum is greater than zero
- 3 Overflow

**Program Exceptions:**

- Access (fetch, operand 2; fetch and store, operand 1)
- Data
- Decimal Overflow

## COMPARE DECIMAL



The first operand is compared with the second, and the condition code indicates the comparison result.

Comparison is algebraic, taking into account the sign and all digits of both operands. All sign and digit codes are checked for validity, and any valid plus or minus sign is considered equal to any other valid plus or minus sign, respectively. If the fields are unequal in length, the shorter is extended with high-order zeros. A field with a zero value and positive sign is considered equal to a field with a zero value but negative sign. Neither operand is changed as a result of the operation. Overflow cannot occur in this operation.

The first-operand and second-operand fields may overlap when their low-order bytes coincide. It is possible, therefore, to compare a number with itself.

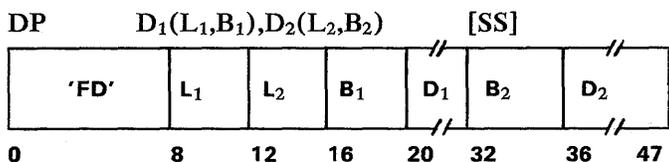
### Resulting Condition Code:

- 0 Operands equal
- 1 First operand is low
- 2 First operand is high
- 3 -

### Program Exceptions:

Access (fetch, operands 1 and 2)  
Data

## DIVIDE DECIMAL



The dividend (the first operand) is divided by the divisor (the second operand) and replaced by the quotient and remainder.

The quotient field is placed leftmost in the first-operand field. The remainder field is placed rightmost in the first-operand field and has a length equal to the divisor length. Together, the quotient and remainder occupy the entire dividend field; therefore, the address of the quotient field is the

address of the first operand. The length of the quotient field in bytes is  $L_1 - L_2$ , and the length code for this field is one less ( $L_1 - L_2 - 1$ ). When the divisor length code is greater than seven (15 digits and sign) or greater than or equal to the dividend length code, a specification exception is recognized. The operation is suppressed, and a program interruption occurs.

The dividend, divisor, quotient, and remainder are all signed integers, 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 dividend and divisor signs. The sign of the remainder has the same value as the dividend sign. These rules are true even when the quotient or remainder is zero.

Overflow cannot occur. If the quotient is too large to be represented by the number of digits allowed, a decimal-divide exception is recognized. The operation is suppressed, and a program interruption occurs. The divisor and dividend remain unchanged in their storage locations. The decimal-divide exception can be indicated only if the digit or digits used in establishing the exception are valid.

The divisor and dividend fields may overlap only if their low-order bytes coincide.

**Condition Code:** The code remains unchanged.

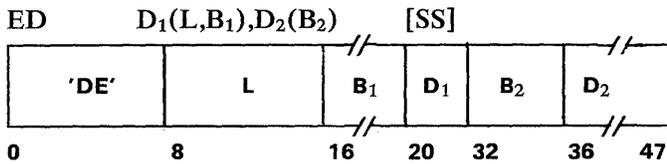
### Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)  
Data  
Decimal Divide  
Specification

### Programming Notes

1. The maximum dividend size is 31 digits and sign. Since the smallest remainder size is one digit and sign, the maximum quotient size is 29 digits and sign.
2. The condition for a decimal divide exception can be determined by a trial subtraction. The leftmost digit of the divisor field is aligned with the leftmost-less-one digit of the dividend field. When the divisor, so aligned, is less than or equal to the dividend, a divide exception is indicated.
3. A decimal-divide exception occurs if the dividend does not have at least one leading zero.

## EDIT



The format of the source (the second operand) is changed from packed to zoned, and is modified under control of the pattern (the first operand). The edited result replaces the pattern.

Editing includes sign and punctuation control, and the suppressing and protecting of leading zeros. 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.

The length field applies to the pattern (the first operand). The pattern may contain any bit combination.

The length of the source (the second operand) is determined by the operation according to the contents of the pattern. Access exceptions are recognized only for those bytes in the second operand which are actually required.

The source has the packed format. The leftmost four bits of a source byte must specify a decimal digit code (0000-1001); a sign code (1010-1111) is recognized as a data exception and causes a program interruption. The rightmost four bits may specify either a sign or a decimal digit.

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 character 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 zoned format.
3. It is replaced by the first character in the pattern, called the fill character.

Which of the three actions takes place is determined by one or more of the following: the type of the pattern character, the state of the significance indicator, and whether the source digit examined is zero.

**Pattern Characters:** There are four types of pattern characters: digit selector, significance starter, field separator, and message character. Their coding is as follows:

Name	Code
Digit selector	0010 0000
Significance starter	0010 0001
Field separator	0010 0010
Message character	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 character, as appropriate, is selected to replace the pattern character. 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 character, and the significance indicator is always off after the field separator is encountered.

Message characters in the pattern are either replaced by the fill character 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 Character:** The fill character is obtained from the pattern as part of the editing operation. The first character of the pattern is used as the fill character. The fill character can have any code and may concurrently specify a control function. If this character is a digit selector or significance starter, the indicated editing action is taken after the code has been assigned to the fill character.

**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. The source digit either is given a zone and replaces the pattern character or is disregarded.

The source digits are selected one byte at a time, and a source byte is fetched for inspection only once during an editing operation. Each source digit is examined only once for a zero value. 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 character that calls for a digit examination. When the leftmost four bits contain an invalid digit code, the operation is 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 four rightmost bit positions, these bits are not treated as a decimal-digit code, and a new source byte is fetched from storage for the next pattern character that calls for a source-digit examination.

When the source digit is stored in the result, its code is expanded from the packed to the zoned format by attaching the zone code 1111.

**Significance Indicator:** The significance indicator, by its on or off state, indicates the significance or nonsignificance, respectively, of subsequent source digits or message characters. Significant source digits replace their corresponding digit selectors or significance starters in the result. Significant message characters remain unchanged in the result.

The significance indicator, by its on or off state, indicates also the negative or positive value, respectively, of the source and is used as one factor in the setting of the condition code.

The indicator is set to the off state, if not already so set, 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. Any of the codes 1010, 1100, 1110, and 1111 is considered a plus code.

The indicator is set to the on state, if not already so set, 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, and if in both instances the source byte does not have a plus code in the rightmost four bit positions.

In all other situations, the indicator is not changed. A minus sign code has no effect on the significance indicator.

**Result Characters:** The field resulting from an editing operation replaces and is equal in length to the pattern. It is composed from pattern characters, fill characters, and zoned source digits.

If the pattern character is a message character and the significance indicator is on, the message character remains unchanged in the result. If the pattern character is a field separator or if the significance indicator is off when a message character is encountered in the pattern, the fill character replaces the pattern character 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 character replaces the pattern character. If a digit selector or significance starter is encountered with either the significance indicator on or with a nonzero decimal source digit, the source digit is considered significant, is zoned, and replaces the pattern character in the result.

**Result Condition:** All digits examined are tested for the code 0000. The sign of the last field edited and whether all source digits in the field contain zeros are recorded in the condition code at the completion of the editing operation.

The condition code is made 0 when the last field is zero, that is, when all source digits examined since the last field separator are zeros. When the pattern has no digit selectors or significance starters, the source is not examined, and the condition code is made 0. Similarly, the condition code is made 0 when the last character in the pattern is a field separator or when no digit selector or significance starter is encountered beyond the last field separator.

When the last field edited is nonzero and the significance indicator is on, the condition code is made 1 to indicate a result field less than zero.

When the last field edited is nonzero and the significance indicator is off, the condition code is made 2 to indicate a result field is greater than zero.

The figure "Summary of EDIT Functions" summarizes the functions of the editing operation. 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 character placed in the result field and the new setting of the significance indicator.

#### **Resulting Condition Code:**

- 0 Last field is zero
- 1 Last field is less than zero
- 2 Last field is greater than zero
- 3 —

#### **Program Exceptions:**

Access (fetch, operand 2; fetch and store, operand 1)  
Data

**Programming Notes**

1. As a rule, the source is shorter than the pattern because for each source digit a zone and numeric are inserted in the result.
2. The total number of digit selectors and significance starters in the pattern must equal the number of source digits to be edited.
3. If the fill character is a blank, if no significance starter appears in the pattern, and if the source is all zeros, the editing operation blanks the result field.

4. The resultant 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 low-order digit position. For multiple-field editing operations, the condition code reflects the sign and value only of the field following the last field separator.

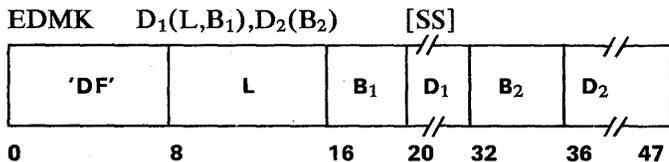
Conditions				Results	
Pattern Character	Previous State of Significance Indicator	Source Digit	Low-Order Source Digit Is a Plus Sign	Result Character	State of Significance Indicator at End of Digit Examination
Digit selector	Off	0	*	Fill character	Off
		1-9	No	Source digit	On
	On	1-9	Yes	Source digit	Off
		0-9	No	Source digit	On
Significance starter	Off	0-9	Yes	Source digit	Off
		0	No	Fill character	On
		0	Yes	Fill character	Off
	On	1-9	No	Source digit	On
		1-9	Yes	Source digit	Off
		0-9	No	Source digit	On
0-9	Yes	Source digit	Off		
Field separator	*	**	**	Fill character	Off
Message character	Off	**	**	Fill character	Off
	On	**	**	Message character	On

**Explanation:**

- \* No effect on result character and new state of significance indicator
- \*\* Not applicable because source digit not examined

**Summary of EDIT Functions**

## EDIT AND MARK



The format of the source (the second operand) is changed from packed to zoned and is modified under control of the pattern (the first operand).

The address of each first significant result character is recorded in general register 1. The edited result replaces the pattern.

The instruction EDIT AND MARK is identical to EDIT, except for the additional function of inserting the address of the result character in bit positions 8-31 of general register 1 whenever the result character is a zoned source digit and the significance indicator was off before the examination. The use of general register 1 is implied. The contents of bit positions 0-7 of the register are not changed.

### Resulting Condition Code:

- 0 Last field is zero
- 1 Last field is less than zero
- 2 Last field is greater than zero
- 3 -

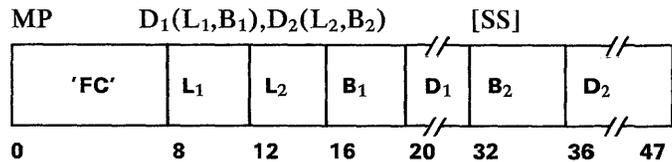
### Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)  
Data

### Programming Notes

- The instruction EDIT AND MARK facilitates the programming of floating currency-symbol insertion. The character address inserted in general register 1 is one more than the address where a floating currency-sign would be inserted. The instruction BRANCH ON COUNT (BCTR), with zero in the R<sub>2</sub> field, may be used to reduce the inserted address by one.
- The character address is not stored when significance is forced. To ensure that general register 1 contains a valid address when significance is forced, it is necessary to place into the register beforehand the address of the pattern character that immediately follows the significance starter.

## MULTIPLY DECIMAL



The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

The multiplier length is limited to 15 digits and sign and must be less than the multiplicand length. If the length code L<sub>2</sub> is greater than 7, or greater than or equal to the length code L<sub>1</sub>, a specification exception is recognized. The operation is suppressed, and a program interruption occurs.

The multiplicand must have at least as many bytes of high-order zeros as the multiplier length in bytes; otherwise, a data exception is recognized, the operation is terminated, and a program interruption occurs. This definition of the multiplicand field ensures that no product overflow can occur. The maximum product length is 31 digits. At least one high-order digit of the product field is zero.

All operands and results are treated as signed integers, right-aligned in their field. All sign and digit codes of the multiplier and multiplicand 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.

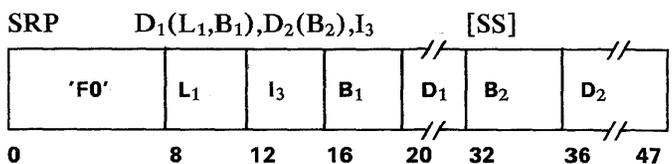
The multiplier and product fields may overlap only if their low-order bytes coincide.

**Condition Code:** The code remains unchanged.

### Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)  
Data  
Specification

## SHIFT AND ROUND DECIMAL

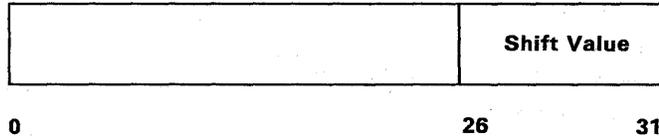


The first operand is shifted in the direction and for the number of digit positions specified by the

second-operand address, and, when shifting to the right is specified, is rounded by the rounding factor,  $I_3$ .

The second-operand address, specified by the  $B_2$  and  $D_2$  fields, is not used to address data; its low-order six bits are the shift value, and the remainder of the address is ignored.

#### Second-Operand Address



The shift value is a six-bit signed binary integer, indicating the direction and the number of 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:

<i>Shift Value</i>	<i>Amount and Direction</i>
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

The  $L_1$ ,  $B_1$ , and  $D_1$  fields are interpreted in the same manner as in the SS format with two length fields. The result replaces the first operand and is not stored outside the field specified by the address and length.

The first operand is considered to be in the 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.

For right shift, the contents of the  $I_3$  field, bit positions 12-15, are used as a rounding factor. The first operand is rounded by decimally adding the rounding factor to the leftmost digit to be shifted out and by propagating the carry, if any, to the left. The result of this addition is then shifted right. Both the first operand and the rounding factor are considered positive quantities for the purpose of this addition. No overflow results from the propagation of a carry since all digits resulting from the addition participate in the shift. Except for validity checking and the

participation in rounding, the digits shifted out of the low-order digit position are ignored and are lost.

In the absence of overflow, the sign of a zero result is made positive. Otherwise, the sign of the result is the same as the original sign, but the code is 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 factor does not have a valid digit code. The validity of first-operand codes is checked even when no shift is specified, and the validity of the rounding factor is checked even when no addition for rounding takes place. The operation is terminated, except when the sign position contains an invalid sign code, in which case the operation is suppressed.

When one or more significant digits are shifted out of the high-order digit positions during left shift, a decimal overflow occurs and results in a program interruption, provided that the decimal-overflow mask bit is one. Overflow cannot occur on right shift or when no shifting is specified.

#### **Resulting Condition Code:**

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Overflow

#### **Program Exceptions:**

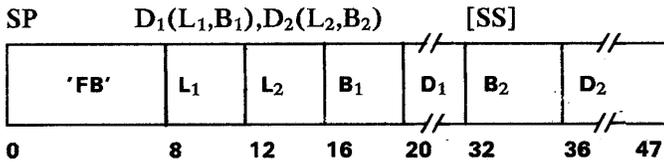
- Access (fetch and store, operand 1)
- Data
- Decimal Overflow

#### **Programming Note**

SHIFT AND ROUND 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 field even when rounding in right shift is specified.

Note that when the  $B_2$  field is zero, the six-bit shift value, bits 26-31 of the second-operand address, are obtained directly from bits 42-47 of the instruction.

## SUBTRACT DECIMAL



The second operand is subtracted from the first operand, and the difference is placed in the first-operand location.

Subtraction is algebraic, taking into account the signs and all digits of both operands. The execution of SUBTRACT DECIMAL is identical to that of ADD DECIMAL, except that the sign of the second operand, if negative, is treated as positive, and, if positive, is treated as negative.

The sign of the difference is determined by the rules of algebra. When the operation is completed without an overflow, a zero difference has a positive sign, but when high-order digits are lost because of an overflow, a zero difference may be either positive or negative, as determined by what the sign of the correct difference would have been.

### Resulting Condition Code:

- 0 Difference is zero
- 1 Difference is less than zero
- 2 Difference is greater than zero
- 3 Overflow

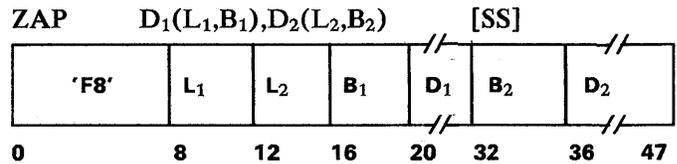
### Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)  
Data  
Decimal Overflow

### Programming Note

The operands of SUBTRACT DECIMAL may overlap when their low-order bytes coincide, even when their lengths are unequal. This property may be used to set to zero an entire field or the low-order part of a field.

## ZERO AND ADD



The second operand is placed in the first-operand location.

The operation is equivalent to an addition to zero. A zero result is positive. When high-order digits are lost because of overflow, a zero result has the sign of the second operand.

Only the second operand is checked for valid sign and digit codes. Extra high-order zeros are supplied if needed. When the first-operand field is too short to contain all significant digits of the second operand, a decimal overflow occurs and results in a program interruption, provided that the decimal-overflow mask bit is one.

The first-operand and second-operand fields may overlap when the rightmost byte of the first-operand field 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.

### Resulting Condition Code:

- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Overflow

### Program Exceptions:

Access (fetch, operand 2; store, operand 1)  
Data  
Decimal Overflow

# Chapter 9. Floating-Point Instructions

## Contents

Data Format	9-1
Guard Digit	9-2
Number Representation	9-3
Normalization	9-3
Instructions	9-4
ADD NORMALIZED	9-6
ADD UNNORMALIZED	9-7
COMPARE	9-8
DIVIDE	9-8
HALVE	9-9
LOAD	9-10

LOAD AND TEST	9-10
LOAD COMPLEMENT	9-11
LOAD NEGATIVE	9-11
LOAD POSITIVE	9-11
LOAD ROUNDED	9-12
MULTIPLY	9-12
STORE	9-14
SUBTRACT NORMALIZED	9-14
SUBTRACT UNNORMALIZED	9-15

The floating-point instructions are used to perform calculations on operands with a wide range of magnitude and to yield results scaled to preserve precision.

A floating-point number consists of a signed exponent, represented by the characteristic, and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess-64 binary notation (see the section "Number Representation" in this chapter); the fraction is expressed as a hexadecimal number having a radix point to the left of the high-order digit.

To avoid unnecessary storing and loading operations for results and operands, four floating-point registers are provided. The floating-point instructions provide for the loading, rounding, adding, subtracting, comparing, multiplying, dividing, and storing, as well as the sign control, of short, long, and extended operands. Short operands generally provide faster processing and require less storage than long or extended operands. On the other hand, long and extended operands provide greater precision in computation. Operations may be either register to register or storage to register.

For addition, subtraction, multiplication, and division, instructions are provided that generate normalized results. Normalized results preserve the highest precision in the operation. For addition and subtraction, instructions are also provided that generate unnormalized results. Normalized and unnormalized operands may be used in any floating-point operation.

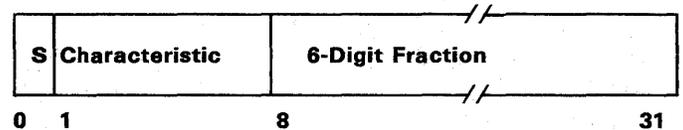
The condition code is set as a result of all sign-control, add, subtract, and compare operations.

### Data Format

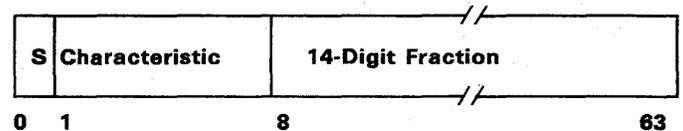
Floating-point data occupies a fixed-length format, which may be either a four-byte (short) format, an eight-byte (long) format, or a 16-byte (extended) format. The short and long formats may be designated as operands both in main storage and in the floating-point registers, whereas the extended formats can be designated only in the floating-point registers.

The floating-point registers are numbered 0, 2, 4, and 6. Designation of an odd-numbered register in the R<sub>1</sub> or R<sub>2</sub> field of a floating-point instruction causes the operation to be suppressed and a program interruption for specification exception to occur.

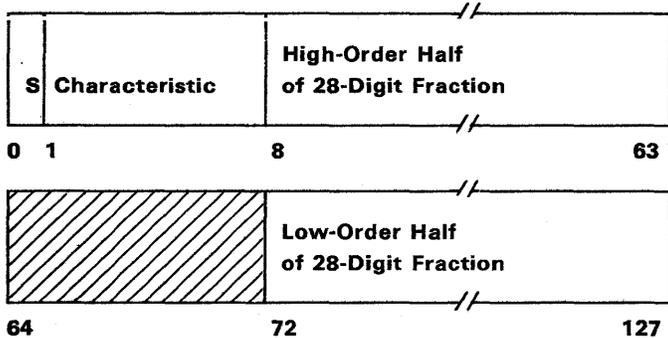
#### Short Floating-Point Number



#### Long Floating-Point Number



## Extended Floating-Point Number



In the short and long formats, the first bit is the sign bit (S). The subsequent seven bit positions are occupied by the characteristic. The following field contains the fraction, which, depending on the format, consists of six or 14 hexadecimal digits.

Short floating-point numbers occupy only the leftmost 32 bit positions of a floating-point register. When a floating-point register is used as the source of a short floating-point number, the rightmost 32 bit positions of the register are ignored. When a floating-point register is used as the destination of a short floating-point number, the rightmost 32 bit positions of the register remain unchanged.

An extended floating-point number has a 28-digit fraction and consists of two long floating-point numbers in consecutive floating-point registers. Two pairs of floating-point registers can be used as sources of extended operands or destinations of extended results: registers 0, 2 and registers 4, 6. The designation of any other register pair causes the operation to be suppressed and a program interruption for a specification exception to occur.

The two long floating-point numbers comprising an extended floating-point number are called the high-order and low-order parts. The high-order part may be any long floating-point number. If it is normalized, the extended number is considered normalized. The characteristic of the high-order part is the characteristic of the extended number, and the sign of the high-order part is the sign of the extended number.

The fraction field of the low-order part contains the 14 low-order hexadecimal digits of the 28-digit extended fraction. The sign and characteristic of the low-order part of an extended operand are ignored, the value of the number being assumed such as if the sign of the low-order part were the same as that of the high-order part, and the characteristic of the low-order part were 14 less than that of the high-order part. In extended results, 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 causes the low-order characteristic to become less than zero, it is made 128 larger than its correct value. Exponent underflow is indicated only when the high-order characteristic underflows.

The entire set of floating-point functions is available for short and long operands. These instructions generate a result that has the same format as the sources, except that in the case of MULTIPLY, a long product is produced from a short multiplier and multiplicand. For extended operands, instructions are provided for normalized addition, subtraction, and multiplication. Additionally, two multiplication instructions are provided that generate an extended product from a long multiplier and multiplicand. The rounding instructions provide for rounding from extended to long format and from long to short format.

### Programming Note

A long floating-point number can be extended to the extended format by appending any long floating-point number having a zero fraction, including a true zero. Conversion from the extended to the long format can be accomplished by truncation or by means of LOAD ROUNDED.

In the absence of an exponent overflow or exponent underflow, the long floating-point 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 relation is true 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 addressed as a long floating-point number, does not have the correct characteristic value.

### Guard Digit

Although final results have six fraction digits in the short format, 14 fraction digits in the long format, and 28 fraction digits in the extended format, intermediate results in ADD NORMALIZED, SUBTRACT NORMALIZED, ADD UNNORMALIZED, SUBTRACT UNNORMALIZED, COMPARE, HALVE, and MULTIPLY may have one additional low-order digit. This low-order digit, the guard digit, increases the precision of the final result.

## Number Representation

The fraction of a floating-point number is expressed in hexadecimal digits. The radix point of the fraction is assumed to be immediately to the left of the high-order fraction digit. The fraction is considered to be multiplied by a power of 16. The characteristic portion, bits 1-7 of the floating-point formats, indicates this power. The bits within the characteristic field can represent numbers from 0 through 127. To accommodate large and small magnitudes, the characteristic is formed by adding 64 to the actual exponent. The range of the exponent is thus -64 through +63. This technique produces a characteristic in excess-64 notation.

Both positive and negative quantities have a true fraction, the sign being indicated by the sign bit. The number is positive or negative, depending on whether the sign bit is zero or one, respectively.

The range covered by the magnitude (M) of a normalized floating-point number is:

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:

$$5.4 \times 10^{-79} \leq M \leq 7.2 \times 10^{75}$$

A number with a zero characteristic, zero fraction, and plus sign is called a true zero. When an extended result is made a true zero, both the high-order and low-order parts are made true zero.

A true zero may arise as the result of an arithmetic operation because of the particular magnitude of the operands. A result is forced to be true zero when

1. An exponent underflow occurs and the exponent-underflow mask bit in the PSW is zero,
2. The result fraction of an addition or subtraction operation is zero and the significance mask bit in the PSW is zero, or
3. The operand of HALVE, one or both operands of MULTIPLY, or the dividend in DIVIDE has a zero fraction.

When a program interruption due to exponent underflow occurs, a true zero fraction is not forced; instead, the fraction and sign remain correct, and the characteristic is 128 too large. When a program interruption due to the significance exception occurs, the fraction remains zero, the sign is positive, and the

characteristic remains correct. The exponent-overflow and exponent-underflow exceptions do not cause a program interruption when the result has a zero fraction. When a divisor has a zero fraction, division is omitted, and a program interruption for a floating-point-divide exception occurs. In addition and subtraction, an operand with a zero fraction or characteristic participates as a normal number.

The sign of a sum, difference, product, or quotient with zero fraction is positive. The sign of a zero fraction resulting from other operations is established by the rules of algebra from the operand signs.

## Normalization

A quantity can be represented with the greatest precision by a floating-point number of given fraction length when that number is normalized. A normalized floating-point number has a nonzero high-order hexadecimal fraction digit. If one or more high-order fraction digits are zeros, the number is said to be unnormalized. The process of normalization consists in shifting the fraction left, one digit at a time, until the high-order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. For extended results, the entire fraction participates in the normalization; therefore, the low-order part may or may not appear to be a normalized long number, depending on the value of the fraction. A number with a zero fraction cannot be normalized, and its characteristic therefore remains unchanged when normalization is called for.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result. This function is called postnormalization. In performing multiplication and division, the operands are normalized before the arithmetic process. This function is called prenormalization.

Floating-point operations may be performed with or without normalization. Most operations are performed only with normalization. Addition and subtraction with short or long operands may be specified either way.

When an operation is performed without normalization, high-order zeros in the result fraction are not eliminated. The result may or may not be normalized, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. Also, intermediate fraction results are shifted right when an overflow occurs, and the intermediate

fraction result is truncated to the final result length after the shifting, if any

#### **Programming Note**

Since normalization applies to hexadecimal digits, the three high-order bits of the fraction of a normalized number may be zero.

### **Instructions**

The floating-point instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of Floating-Point Instructions." The figure also indicates when the condition code is set 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 a register-to-register operation using LOAD (short), for example, LER is the mnemonic and R<sub>1</sub>,R<sub>2</sub> the operand designation.*

Mnemonics for the floating-point instructions have an "R" as the last letter when the instruction is in the RR format. For instructions where all operands are the same length, certain letters are used to represent operand-format length and normalization, as follows:

E	short normalized
U	short unnormalized
D	long normalized
W	long unnormalized
X	extended normalized

Name	Mnemonic	Characteristics							Code
ADD NORMALIZED (extended)	AXR	RR	C		SP	U	EO	LS	36
ADD NORMALIZED (long)	ADR	RR	C		SP	U	EO	LS	2A
ADD NORMALIZED (long)	AD	RX	C	A	SP	U	EO	LS	6A
ADD NORMALIZED (short)	AER	RR	C		SP	U	EO	LS	3A
ADD NORMALIZED (short)	AE	RX	C	A	SP	U	EO	LS	7A
ADD UNNORMALIZED (long)	AWR	RR	C		SP		EO	LS	2E
ADD UNNORMALIZED (long)	AW	RX	C	A	SP		EO	LS	6E
ADD UNNORMALIZED (short)	AUR	RR	C		SP		EO	LS	3E
ADD UNNORMALIZED (short)	AU	RX	C	A	SP		EO	LS	7E
COMPARE (long)	CDR	RR	C		SP				29
COMPARE (long)	CD	RX	C	A	SP				69
COMPARE (short)	CER	RR	C		SP				39
COMPARE (short)	CE	RX	C	A	SP				79
DIVIDE (long)	DDR	RR			SP	U	EO	FK	2D
DIVIDE (long)	DD	RX		A	SP	U	EO	FK	6D
DIVIDE (short)	DER	RR			SP	U	EO	FK	3D
DIVIDE (short)	DE	RX		A	SP	U	EO	FK	7D
HALVE (long)	HDR	RR			SP	U			24
HALVE (short)	HER	RR			SP	U			34
LOAD (long)	LDR	RR			SP				28
LOAD (long)	LD	RX		A	SP				68
LOAD (short)	LER	RR			SP				38
LOAD (short)	LE	RX		A	SP				78
LOAD AND TEST (long)	LTDR	RR	C		SP				22
LOAD AND TEST (short)	LTER	RR	C		SP				32
LOAD COMPLEMENT (long)	LCDR	RR	C		SP				23
LOAD COMPLEMENT (short)	LCER	RR	C		SP				33
LOAD NEGATIVE (long)	LNDR	RR	C		SP				21
LOAD NEGATIVE (short)	LNER	RR	C		SP				31
LOAD POSITIVE (long)	LPDR	RR	C		SP				20
LOAD POSITIVE (short)	LPER	RR	C		SP				30
LOAD ROUNDED (extended to long)	LRDR	RR			SP		EO		25
LOAD ROUNDED (long to short)	LRER	RR			SP		EO		35
MULTIPLY (extended)	MXR	RR			SP	U	EO		26
MULTIPLY (long)	MDR	RR			SP	U	EO		2C
MULTIPLY (long)	MD	RX		A	SP	U	EO		6C
MULTIPLY (long to extended)	MXDR	RR			SP	U	EO		27
MULTIPLY (long to extended)	MXD	RX		A	SP	U	EO		67
MULTIPLY (short to long)	MER	RR			SP	U	EO		3C
MULTIPLY (short to long)	ME	RX		A	SP	U	EO		7C
STORE (long)	STD	RX		A	SP			ST	60
STORE (short)	STE	RX		A	SP			ST	70
SUBTRACT NORMALIZED (extended)	SXR	RR	C		SP	U	EO	LS	37
SUBTRACT NORMALIZED (long)	SDR	RR	C		SP	U	EO	LS	2B
SUBTRACT NORMALIZED (long)	SD	RX	C	A	SP	U	EO	LS	6B
SUBTRACT NORMALIZED (short)	SER	RR	C		SP	U	EO	LS	3B
SUBTRACT NORMALIZED (short)	SE	RX	C	A	SP	U	EO	LS	7B
SUBTRACT UNNORMALIZED (long)	SWR	RR	C		SP		EO	LS	2F
SUBTRACT UNNORMALIZED (long)	SW	RX	C	A	SP		EO	LS	6F
SUBTRACT UNNORMALIZED (short)	SUR	RR	C		SP		EO	LS	3F
SUBTRACT UNNORMALIZED (short)	SU	RX	C	A	SP		EO	LS	7F

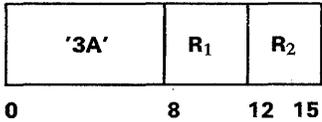
**Explanation:**

A	Access exceptions	RR	RR instruction format
C	Condition code is set	RX	RX instruction format
EO	Exponent-overflow exception	SP	Specification exception
FK	Floating-point-divide exception	ST	PER storage-alteration event
LS	Significance exception	U	Exponent-underflow exception

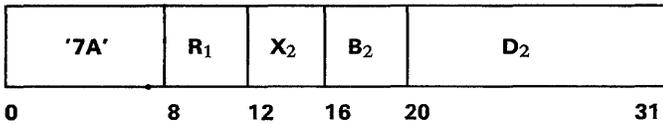
Summary of Floating-Point Instructions

## ADD NORMALIZED

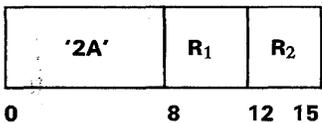
AER  $R_1, R_2$   
[RR, Short Operands]



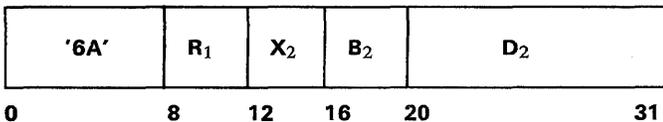
AE  $R_1, D_2(X_2, B_2)$   
[RX, Short Operands]



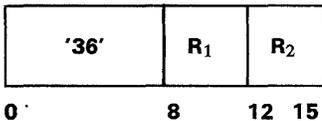
ADR  $R_1, R_2$   
[RR, Long Operands]



AD  $R_1, D_2(X_2, B_2)$   
[RX, Long Operands]



AXR  $R_1, R_2$   
[RR, Extended Operands]



The second operand is added to the first operand, and the normalized sum is placed in the first-operand location.

Addition of two floating-point numbers consists in characteristic comparison and fraction addition. The characteristics of the two operands are compared, and the fraction accompanying the smaller characteristic is shifted right, with its characteristic increased by one for each hexadecimal digit of shift until the two characteristics agree.

When an operand is shifted right during alignment, the leftmost hexadecimal digit of the field shifted out is retained as a guard digit. The operand that is not shifted is considered to be extended with a low-order zero. Both operands are considered to be extended with low-order zeros when no alignment shift occurs. The fractions are then added algebraically to form an intermediate sum.

The short intermediate-sum fraction consists of seven hexadecimal digits and a possible carry. The long intermediate-sum fraction consists of 15 hexadecimal digits and a possible carry. The extended intermediate-sum fraction consists of 29 hexadecimal digits and a possible carry. If a carry is present, the sum is shifted right one digit position, and the characteristic is increased by one.

After the addition, the intermediate sum is shifted left as necessary to form a normalized number, provided the fraction is not zero. Vacated low-order digit positions are filled with zeros, and the characteristic is reduced by the number of hexadecimal digits of shift. The intermediate-sum fraction is subsequently truncated to the proper result-fraction length.

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 exponent-overflow exception is recognized when a carry from the high-order position of the intermediate-sum fraction causes the characteristic of the normalized sum to exceed 127. The operation is completed by making the characteristic 128 less than the correct value, and a program interruption for exponent overflow occurs. The result is normalized, the sign and fraction remain correct, and, for AXR, the low-order characteristic remains correct.

An exponent-underflow exception exists when the characteristic of the normalized sum is less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent underflow then takes place. When exponent underflow occurs and the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the result a true zero. For AXR, exponent underflow is not recognized when the low-order characteristic is less than zero, but the high-order characteristic is zero or above.

A significance exception exists when the intermediate-sum fraction, including the guard digit, is zero. If the significance mask bit is one, the intermediate-sum characteristic remains unchanged and becomes the characteristic of the result. No normalization occurs, and a program interruption for significance takes place. If the significance mask bit is zero, the program interruption does not occur; instead, the result is made a true zero.

The  $R_1$  field for AER, AE, ADR, and AD, and the  $R_2$  field for AER and ADR must designate register 0, 2, 4, or 6. The  $R_1$  and  $R_2$  fields for AXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

**Resulting Condition Code:**

- 0 Result fraction is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

**Program Exceptions:**

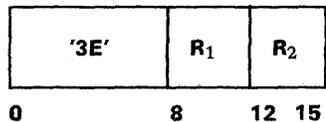
- Access (fetch, operand 2 of AE and AD only)
- Exponent Overflow
- Exponent Underflow
- Significance
- Specification

**Programming Note**

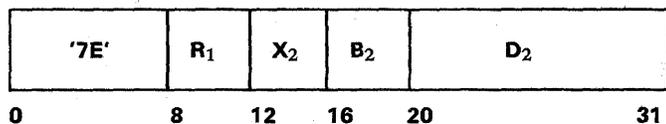
Interchanging the two operands in a floating-point addition does not affect the value of the sum.

**ADD UNNORMALIZED**

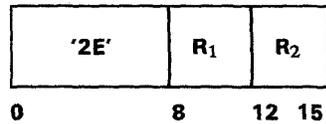
AUR  $R_1, R_2$   
[RR, Short Operands]



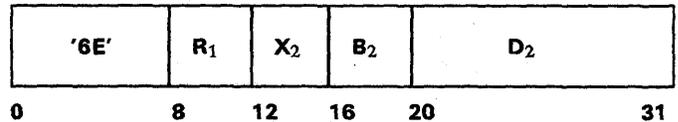
AU  $R_1, D_2(X_2, B_2)$   
[RX, Short Operands]



AWR  $R_1, R_2$   
[RR, Long Operands]



AW  $R_1, D_2(X_2, B_2)$   
[RX, Long Operands]



The second operand is added to the first operand, and the unnormalized sum is placed in the first-operand location.

The execution of ADD UNNORMALIZED is identical to that of ADD NORMALIZED, except that, after the addition, the intermediate-sum fraction is truncated to the proper result-fraction length without performing normalization. Leading zeros are not eliminated in the result fraction, exponent underflow cannot occur, and the guard digit does not participate in the recognition of significance exception. A significance exception is recognized when the intermediate-sum fraction, not including the guard digit, is zero.

The  $R_1$  and  $R_2$  fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

**Resulting Condition Code:**

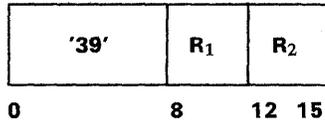
- 0 Result fraction is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

**Program Exceptions:**

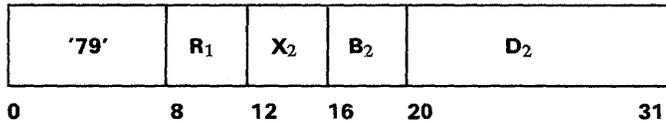
- Access (fetch, operand 2 of AU and AW only)
- Exponent Overflow
- Significance
- Specification

## COMPARE

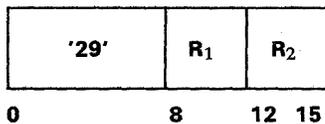
CER  $R_1, R_2$   
[RR, Short Operands]



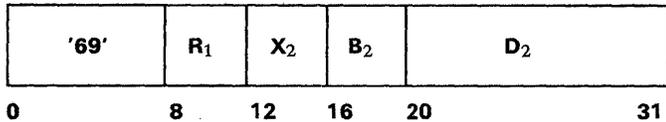
CE  $R_1, D_2(X_2, B_2)$   
[RX, Short Operands]



CDR  $R_1, R_2$   
[RR, Long Operands]



CD  $R_1, D_2(X_2, B_2)$   
[RX, Long Operands]



The first operand is compared with the second operand, and the condition code is set to indicate the result.

Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. An equality is established by following the rules for normalized floating-point subtraction. When the intermediate sum, including the guard digit, is zero, the operands are equal. An exponent inequality is not decisive for magnitude determination since the fractions may have different numbers of leading zeros. Neither operand is changed as a result of the operation.

An exponent-overflow, exponent-underflow, or significance exception cannot occur.

The  $R_1$  and  $R_2$  fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

## Resulting Condition Code:

- 0 Operands are equal
- 1 First operand is low
- 2 First operand is high
- 3 -

## Program Exceptions:

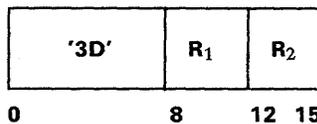
Access (fetch, operand 2 of CE and CD only)  
Specification

## Programming Note

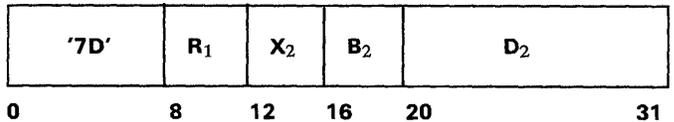
Numbers with zero fractions compare equal even when they differ in sign or characteristic.

## DIVIDE

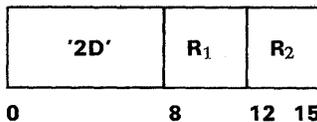
DER  $R_1, R_2$   
[RR, Short Operands]



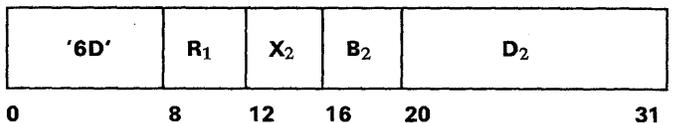
DE  $R_1, D_2(X_2, B_2)$   
[RX, Short Operands]



DDR  $R_1, R_2$   
[RR, Long Operands]



DD  $R_1, D_2(X_2, B_2)$   
[RX, Long Operands]



The first operand (the dividend) is divided by the second operand (the divisor) and replaced by the normalized quotient. No remainder is preserved.

Floating-point division consists in characteristic subtraction and fraction division. The operands are prenormalized, and the difference between the dividend and divisor characteristics of the normalized operands, plus 64, is used as the characteristic of the intermediate quotient.

All dividend and divisor fraction digits participate in forming the fraction of the quotient. Postnormalizing the intermediate quotient is never necessary, but a right-shift of one digit position may be called for. The intermediate-quotient characteristic is adjusted for the shift. The intermediate-quotient fraction is subsequently truncated to the proper result-fraction length.

The sign of the quotient is determined by the rules of algebra, unless the quotient is made a true zero, in which case the sign is made plus.

An exponent-overflow exception is recognized when the final-quotient characteristic exceeds 127 and the fraction is not zero. The operation is completed by making the characteristic 128 less than the correct value. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent overflow occurs.

An exponent-underflow exception exists when the characteristic of the normalized quotient is less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the quotient a true zero.

Exponent underflow is not signaled when an operand characteristic becomes less than zero during prenormalization or the intermediate-quotient characteristic is less than zero, but the final quotient can be expressed without encountering exponent underflow.

A floating-point divide exception is recognized when the divisor fraction is zero. The operation is suppressed, and a program interruption for floating-point divide occurs.

When the dividend fraction is zero, the quotient is made a true zero, and a possible exponent overflow or exponent underflow is not recognized. A division of zero by zero, however, causes the operation to be suppressed and an interruption for floating-point divide to occur.

The  $R_1$  and  $R_2$  fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

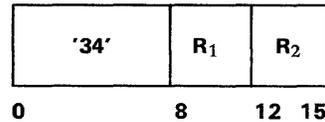
**Condition Code:** The code remains unchanged.

**Program Exceptions:**

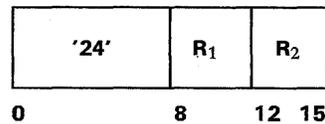
- Access (fetch, operand 2 of DD and DE only)
- Exponent Overflow
- Exponent Underflow
- Floating-Point Divide Specification

**HALVE**

HER  $R_1, R_2$   
[RR, Short Operands]



HDR  $R_1, R_2$   
[RR, Long Operands]



The second operand is divided by 2, and the normalized quotient is placed in the first-operand location.

The fraction of the second operand is shifted right one bit position, placing the contents of the low-order bit position into the high-order bit position of the guard digit and introducing a zero into the high-order bit position of the fraction. The intermediate result is subsequently normalized, and the normalized quotient is placed in the first-operand location. The guard digit participates in the normalization.

The sign of the quotient is the same as that of the second operand, unless the quotient is made a true zero, in which case the sign is made plus.

An exponent-underflow exception exists when the characteristic of the normalized quotient is less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, a program interruption occurs. The result is normalized, the sign and fraction remain correct, and the

characteristic is made 128 greater than the correct value. If the exponent underflow mask bit is zero, program interruption does not take place; instead, the operation is completed by making the quotient a true zero.

When the fraction of the second operand is zero, the result is made a true zero, and no exceptions are recognized.

The R<sub>1</sub> and R<sub>2</sub> fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

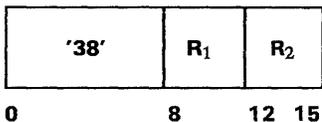
Exponent Underflow  
Specification

**Programming Notes**

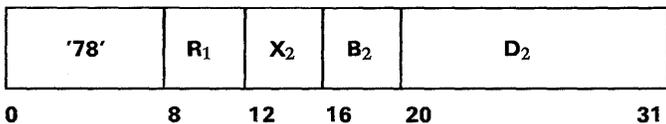
1. With short and long operands, the halve operation is identical to a divide operation with the number 2 as divisor. Similarly, the result of HDR is identical to that of MD or MDR with one-half as a multiplier. No multiply operation corresponds to HER, since no multiply operation produces short results.
2. The result of HALVE is replaced by a true zero only when the second-operand fraction is zero, or when exponent underflow occurs with the exponent-underflow mask set to zero. When the fraction of the second operand is zero, except for the low-order bit position, the low-order one is shifted into the guard-digit position and participates in the postnormalization.

**LOAD**

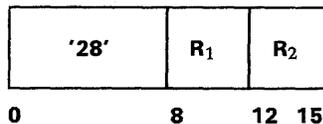
LER R<sub>1</sub>,R<sub>2</sub>  
[RR, Short Operands]



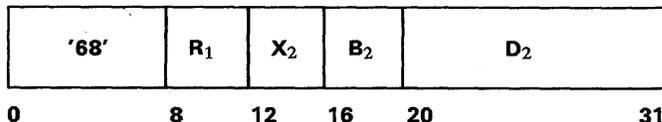
LE R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
[RX, Short Operands]



LDR R<sub>1</sub>,R<sub>2</sub>  
[RR, Long Operands]



LD R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
[RX, Long Operands]



The second operand is placed unchanged in the first-operand location.

The R<sub>1</sub> and R<sub>2</sub> fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

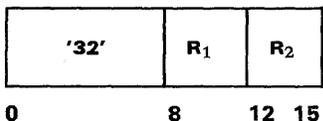
**Condition Code:** The code remains unchanged.

**Program Exceptions:**

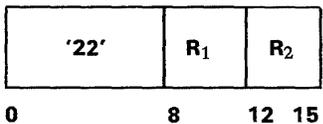
Access (fetch, operand 2 of LE and LD only)  
Specification

**LOAD AND TEST**

LTER R<sub>1</sub>,R<sub>2</sub>  
[RR, Short Operands]



LTDR R<sub>1</sub>,R<sub>2</sub>  
[RR, Long Operands]



The second operand is placed unchanged in the first-operand location, and its sign and magnitude are tested to determine the setting of the condition code.

The R<sub>1</sub> and R<sub>2</sub> fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

**Resulting Condition Code:**

- 0 Result fraction is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

**Program Exceptions:**

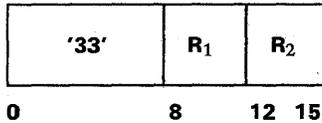
Specification

**Programming Note**

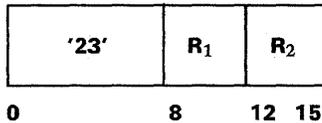
When the same register is specified as the first-operand and second-operand location, the operation is equivalent to a test without data movement.

**LOAD COMPLEMENT**

LCER R<sub>1</sub>,R<sub>2</sub>  
[RR, Short Operands]



LCDR R<sub>1</sub>,R<sub>2</sub>  
[RR, Long Operands]



The second operand is placed in the first-operand location with the sign changed to the opposite value.

The sign bit is inverted, even if the fraction is zero. The characteristic and fraction are not changed.

The R<sub>1</sub> and R<sub>2</sub> fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

**Resulting Condition Code:**

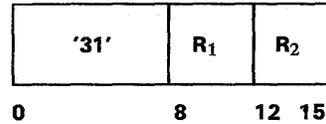
- 0 Result fraction is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

**Program Exceptions:**

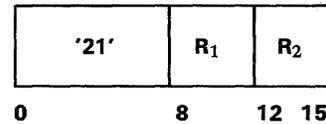
Specification

**LOAD NEGATIVE**

LNER R<sub>1</sub>,R<sub>2</sub>  
[RR, Short Operands]



LNDR R<sub>1</sub>,R<sub>2</sub>  
[RR, Long Operands]



The second operand is placed in the first-operand location with the sign made minus.

The sign bit is made one, even if the fraction is zero. The characteristic and fraction are not changed.

The R<sub>1</sub> and R<sub>2</sub> fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

**Resulting Condition Code:**

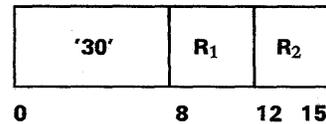
- 0 Result fraction is zero
- 1 Result is less than zero
- 2 -
- 3 -

**Program Exceptions:**

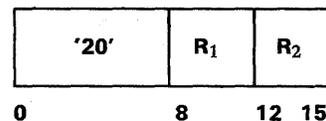
Specification

**LOAD POSITIVE**

LPER R<sub>1</sub>,R<sub>2</sub>  
[RR, Short Operands]



LPDR R<sub>1</sub>,R<sub>2</sub>  
[RR, Long Operands]



The second operand is placed in the first-operand location with the sign made plus.

The sign bit is made zero. The characteristic and fraction are not changed.

The R<sub>1</sub> and R<sub>2</sub> fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

**Resulting Condition Code:**

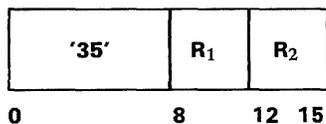
- 0 Result fraction is zero
- 1 -
- 2 Result is greater than zero
- 3 -

**Program Exceptions:**

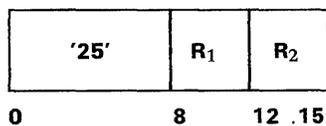
Specification

**LOAD ROUNDED**

LRER R<sub>1</sub>,R<sub>2</sub>  
[RR, Long Operand 2, Short Operand 1]



LRDR R<sub>1</sub>,R<sub>2</sub>  
[RR, Extended Operand 2, Long Operand 1]



The second operand is rounded to the next smaller format, and the result is placed in the first-operand location.

Rounding consists in adding a one in bit position 32 or 72 of the long or extended second operand, respectively, and propagating the carry, if any, to the left. For both cases, the sign of the fraction is ignored, and addition is performed as if the fractions were positive.

If rounding causes a carry out of the high-order digit position of the fraction, the fraction is shifted right one digit position, and the characteristic is increased by one.

The sign of the result is the same as the sign of the second operand. No normalization takes place.

An exponent-overflow exception is recognized when shifting the fraction right causes the characteristic to exceed 127. The operation is completed by loading a

number whose characteristic is 128 less than the correct value, and a program interruption for exponent overflow occurs. The result is normalized, and the sign and fraction remain correct.

Exponent-underflow and significance exceptions cannot occur.

The R<sub>1</sub> field must designate register 0, 2, 4, or 6; the R<sub>2</sub> field of LRER must designate register 0, 2, 4, or 6; and the R<sub>2</sub> field of LRDR must designate register 0 or 4. Otherwise, a specification exception is recognized.

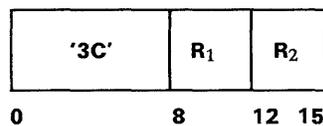
**Condition Code:** The code remains unchanged.

**Program Exceptions:**

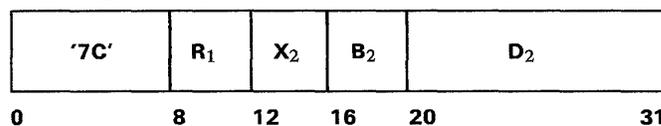
Exponent Overflow  
Specification

**MULTIPLY**

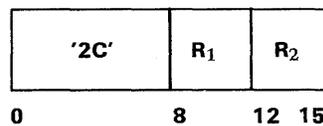
MER R<sub>1</sub>,R<sub>2</sub>  
[RR, Short Multiplier and Multiplicand, Long Product]



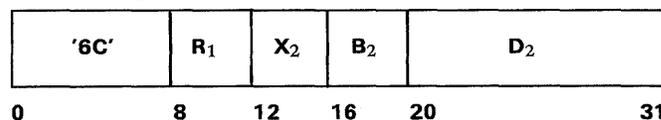
ME R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
[RX, Short Multiplier and Multiplicand, Long Product]



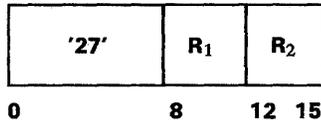
MDR R<sub>1</sub>,R<sub>2</sub>  
[RR, Long Operands]



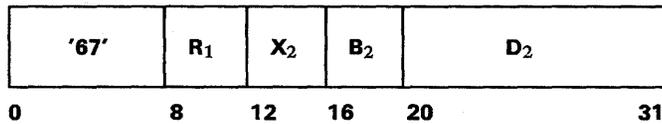
MD R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
[RX, Long Operands]



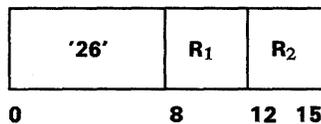
**MXDR**  $R_1, R_2$   
 [RR, Long Multiplier and Multiplicand, Extended Product]



**MXD**  $R_1, D_2(X_2, B_2)$   
 [RX, Long Multiplier and Multiplicand, Extended Product]



**MXR**  $R_1, R_2$   
 [RR, Extended Operands]



The normalized product of the second operand (the multiplier) and the first operand (the multiplicand) is placed in the first-operand location.

Multiplication of two floating-point numbers consists in exponent addition and fraction multiplication. The operands are prenormalized, and the sum of the characteristics of the normalized operands, less 64, is used as the characteristic of the intermediate product.

The product of the fractions is developed such that the result has the exact fraction product truncated to the proper result-fraction length. When the result is normalized without requiring any postnormalization, the intermediate-product fraction is truncated to the result-fraction length, and the intermediate-product characteristic becomes the final product characteristic. When the intermediate-product fraction has one leading zero digit, it is shifted left one digit position, bringing the contents of the guard-digit position into the low-order position of the result fraction, and the intermediate-product characteristic is reduced by one. The intermediate-product fraction is subsequently truncated to the result-fraction length.

For MER and ME, the multiplier and multiplicand have six-digit fractions, and the product fraction has the full 14 digits of the long format, with the two low-order fraction digits always zeros. For MDR and

MD, the multiplier and multiplicand fractions have 14 digits, and the result product fraction is truncated to 14 digits. For MXDR and MXD, the multiplier and multiplicand fractions have 14 digits, with the multiplicand occupying the high-order part of the first operand; the result 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 result product fraction is truncated to 28 digits.

The sign of the product is determined by the rules of algebra, unless all digits of the product fraction are zeros, in which case the sign is made plus.

An exponent-overflow exception is recognized when the characteristic of the normalized product exceeds 127 and the fraction of the product is not zero. The operation is completed by making the characteristic 128 less than the correct value. If, for extended results, the low-order characteristic also exceeds 127, it, too, is decreased by 128. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent overflow then occurs.

Exponent overflow is not recognized if the intermediate-product characteristic exceeds 127 but is brought within range by normalization.

An exponent-underflow exception exists when the characteristic of the normalized product is less than zero and the fraction of the product is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, program interruption does not take place; instead, the operation is completed by making the product a true zero. For extended results, 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.

Exponent underflow is not recognized when the characteristic of an operand becomes less than zero during prenormalization, but the characteristic of the normalized product is within range.

When either or both operand fractions are zero, the result is made a true zero, and no exceptions are recognized.

The  $R_1$  field for MER, ME, MDR, and MD, and the  $R_2$  field for MER, MDR, and MXDR must designate register 0, 2, 4, or 6. The  $R_1$  field for MXDR, MXD, and MXR, and the  $R_2$  field for MXR

must designate register 0 or 4. Otherwise, a specification exception is recognized.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

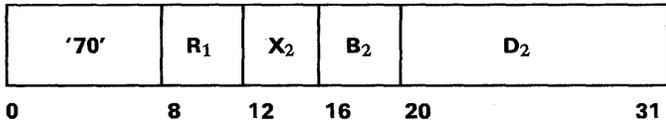
Access (fetch, operand 2 of ME, MD, and MXD only)  
 Exponent Overflow  
 Exponent Underflow  
 Specification

**Programming Note**

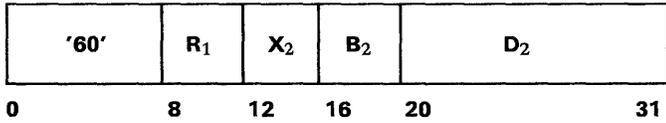
Interchanging the two operands in a floating-point multiplication does not affect the value of the product.

**STORE**

STE R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
 [RX, Short Operands]



STD R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
 [RX, Long Operands]



The first operand is placed unchanged at the second-operand location.  
 The R<sub>1</sub> field must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

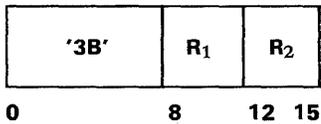
**Condition Code:** The code remains unchanged.

**Program Exceptions:**

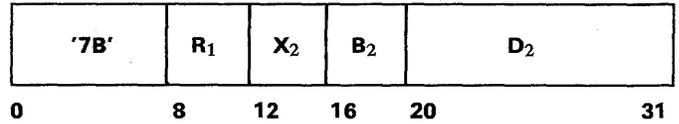
Access (store, operand 2)  
 Specification

**SUBTRACT NORMALIZED**

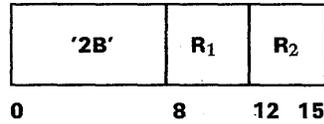
SER R<sub>1</sub>,R<sub>2</sub>  
 [RR, Short Operands]



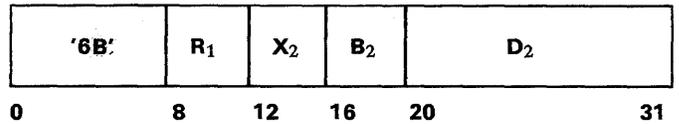
SE R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
 [RX, Short Operands]



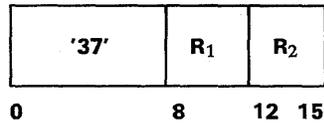
SDR R<sub>1</sub>,R<sub>2</sub>  
 [RR, Long Operands]



SD R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
 [RX, Long Operands]



SXR R<sub>1</sub>,R<sub>2</sub>  
 [RR, Extended Operands]



The second operand is subtracted from the first operand, and the normalized difference is placed in 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.

The R<sub>1</sub> field of SER, SE, SDR, and SD, and the R<sub>2</sub> field of SER and SDR must designate register 0, 2, 4, or 6. The R<sub>1</sub> and R<sub>2</sub> fields of SXR must designate register 0 or 4. Otherwise, a specification exception is recognized.

**Resulting Condition Code:**

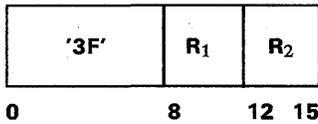
- 0 Result fraction is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

**Program Exceptions:**

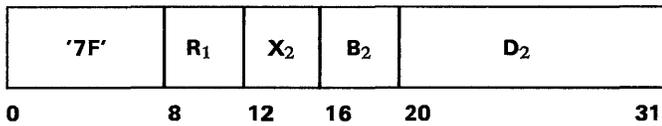
Access (fetch, operand 2 of SE and SD only)  
Exponent Overflow  
Exponent Underflow  
Significance  
Specification

**SUBTRACT UNNORMALIZED**

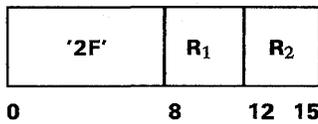
SUR R<sub>1</sub>,R<sub>2</sub>  
[RR, Short Operands]



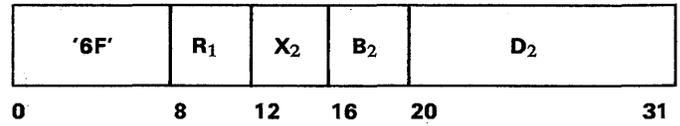
SU R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
[RX, Short Operands]



SWR R<sub>1</sub>,R<sub>2</sub>  
[RR, Long Operands]



SW R<sub>1</sub>,D<sub>2</sub>(X<sub>2</sub>,B<sub>2</sub>)  
[RX, Long Operands]



The second operand is subtracted from the first operand, and the unnormalized difference is placed in 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.

The R<sub>1</sub> and R<sub>2</sub> fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

**Resulting Condition Code:**

- 0 Result fraction is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 -

**Program Exceptions:**

Access (fetch, operand 2 of SU and SW only)  
Exponent Overflow  
Significance  
Specification

## Chapter 10. Control Instructions

### Contents

CLEAR PAGE	10-3	SET CLOCK	10-9
CONNECT PAGE	10-3	SET CLOCK COMPARATOR	10-9
DECONFIGURE PAGE	10-4	SET CPU TIMER	10-10
DIAGNOSE	10-4	SET PAGE BITS	10-10
DISCONNECT PAGE	10-5	SET PSW KEY FROM ADDRESS	10-10
INSERT PAGE BITS	10-6	SET STORAGE KEY	10-11
INSERT PSW KEY	10-5	SET SYSTEM MASK	10-11
INSERT STORAGE KEY	10-6	STORE CAPACITY COUNTS	10-11
LOAD CONTROL	10-6	STORE CLOCK COMPARATOR	10-12
LOAD FRAME INDEX	10-6	STORE CONTROL	10-12
LOAD PSW	10-7	STORE CPU ID	10-12
MAKE ADDRESSABLE	10-7	STORE CPU TIMER	10-13
MAKE UNADDRESSABLE	10-8	STORE THEN AND SYSTEM MASK	10-13
RESET REFERENCE BIT	10-8	STORE THEN OR SYSTEM MASK	10-14
RETRIEVE STATUS AND PAGE	10-8		

The control instructions include all privileged instructions, except the input/output instructions, which are described in Chapter 12, "Input/Output Operations."

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 control instructions and their mnemonics, formats, and operation codes are listed in the figure "Control Instructions." The figure also indicates

when the condition code is set 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 LOAD PSW, for example, LPSW is the mnemonic and  $D_2(B_2)$  the operand designation.*

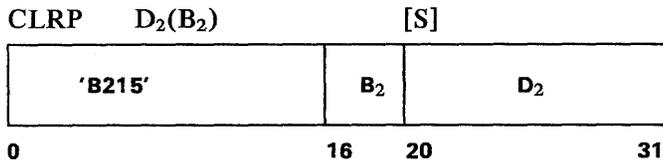
Name	Mnemonic	Characteristics						Code	
CLEAR PAGE	CLRP	S		M	AP		PS	ST	B215
CONNECT PAGE	CTP	RS	C	M	AD	SP	PT	R	B0
DECONFIGURE PAGE	DEP	S		M	AD	SP	PT		B21B
DIAGNOSE				M	DM				83
DISCONNECT PAGE	DCTP	S	C	M	AD	SP	PT		B21C
INSERT PAGE BITS	IPB	RS		M	AD			R	B4
INSERT PSW KEY	IPK	S		M				R	B20B
INSERT STORAGE KEY	ISK	RR		M	AD	SP		R	09
LOAD CONTROL	LCTL	RS		M	A	SP			B7
LOAD FRAME INDEX	LFI	RS	C	M				R	B8
LOAD PSW	LPSW	S	L	M	A	SP	\$		82
MAKE ADDRESSABLE	MAD	S	C	M	AD		PT		B21D
MAKE UNADDRESSABLE	MUN	S	C	M	AD	SP	PT		B21E
RESET REFERENCE BIT	RRB	S	C	M	AD				B213
RETRIEVE STATUS AND PAGE	RSP	SS	C	M	A			ST	D8
SET CLOCK	SCK	S	C	M	A	SP			B204
SET CLOCK COMPARATOR	SCKC	S		M	A	SP			B206
SET CPU TIMER	SPT	S		M	A	SP			B208
SET PAGE BITS	SPB	RS	C	M	AD				B5
SET PSW KEY FROM ADDRESS	SPKA	S		M					B20A
SET STORAGE KEY	SSK	RR		M	AD	SP			08
SET SYSTEM MASK	SSM	S		M	A	SP	SO		80
STORE CAPACITY COUNTS	STCAP	S		M	A			ST	B21F
STORE CLOCK COMPARATOR	STCKC	S		M	A	SP		ST	B207
STORE CONTROL	STCTL	RS		M	A	SP		ST	B6
STORE CPU ID	STIDP	S		M	A	SP		ST	B202
STORE CPU TIMER	STPT	S		M	A	SP		ST	B209
STORE THEN AND SYSTEM MASK	STNSM	SI		M	A			ST	AC
STORE THEN OR SYSTEM MASK	STOSM	SI		M	A	SP		ST	AD

**Explanation:**

\$ Causes serialization	PT Page-transition exception
A Access exceptions	R PER general-register-alteration event
AD Addressing exception	RR RR instruction format
AP Addressing and protection exceptions	RS RS instruction format
C Condition code is set	S S instruction format
DM DIAGNOSE may generate various program exceptions and may change the condition code	SI SI instruction format
L New condition code loaded	SO Special-operation exception
M Privileged-operation exception	SP Specification exception
PS Page-state exception	SS SS instruction format
	ST PER storage-alteration event

**Control Instructions**

## CLEAR PAGE



The storage page designated by the second-operand address is cleared, which is equivalent to storing 2,048 zero bytes at that location. The page is validated.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

The page may be addressable or connected; if the page is disconnected, a page-state exception is raised, and the operation is suppressed.

**Condition Code:** The code remains unchanged.

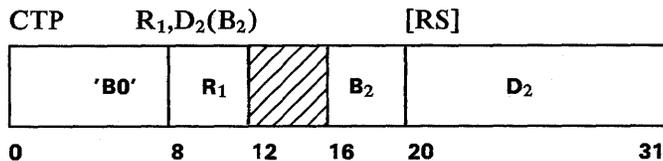
### Program Exceptions:

Addressing (operand 2)  
Page State  
Privileged Operation  
Protection (store, operand 2)

### Programming Note

Page 0 may be cleared, but it can only be in the addressable state.

## CONNECT PAGE



If disconnected, the storage page designated by the second-operand address enters the connected state. If already connected, the page remains in the connected state. The frame index of the page frame that is connected to the page is returned in the general register designated by the R<sub>1</sub> field.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored. Bits 12-15 of the instruction are ignored.

If bits 8-20 of the second-operand address are zeros, that is, page 0 is specified, a specification exception is recognized, and the operation is suppressed.

If the page is in the addressable state, a page-transition exception is recognized, and the operation is suppressed.

If the operation is not successful, because the page is disconnected but no page frame is free for connection (free-frame-capacity count is zero), the R<sub>1</sub> register remains unchanged, and condition code 2 is set.

If the operation is successful, the condition code indicates whether the page was connected (1) or disconnected (0) at the start of the operation. The frame index, which is an unsigned binary integer, is loaded right-aligned in the R<sub>1</sub> register, and the remaining high-order bits of the register are set to zeros. The frame index is unique and may have any value from zero to EFCC - 1, where EFCC is the existing-frame-capacity count.

If the page was disconnected before and the operation is successful, the value of the free-frame-capacity count is decreased by one.

The contents of a newly connected page frame are unpredictable.

### Resulting Condition Code:

- 0 Successful, page was disconnected, index returned
- 1 Page was already connected, index returned
- 2 Not successful, index not returned
- 3 -

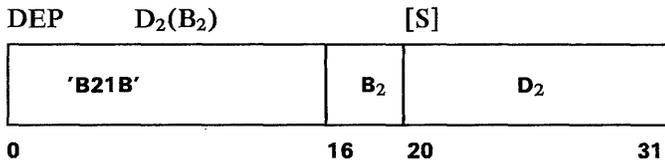
### Program Exceptions:

Addressing (operand 2)  
Page Transition  
Privileged Operation  
Specification

### Programming Notes

1. The storage key and the reference, change, and page bits of a page are not changed when the page is connected.
2. The frame index of the page frame connected to the specified page remains unchanged until that page is disconnected. The value of the frame index to be assigned by CONNECT PAGE to a previously disconnected page is unpredictable.

## DECONFIGURE PAGE



If connected, the storage page designated by the second-operand address enters the disconnected state. The page frame that was connected to the page becomes unavailable; that is, it will no longer be available for connection to any page. The reference and change bits of the page are set to zeros.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

If bits 8-20 of the second-operand address are zeros, that is, page 0 is specified, a specification exception is recognized, and the operation is suppressed.

The page must be in the connected state at the start of the operation; otherwise, a page-transition exception is recognized, and the operation is suppressed.

The value of the available-frame-capacity count is decreased by one. The values of the free-frame and existing-frame-capacity counts remain unchanged.

**Condition Code:** The code remains unchanged.

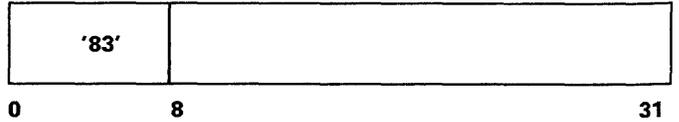
### Program Exceptions:

Addressing (operand 2)  
Page Transition  
Privileged Operation  
Specification

### Programming Notes

1. DECONFIGURE PAGE allows a program to put a page frame out of operation. This may be desirable when the page frame is indicated as defective by a machine check which is caused by a storage access to the page connected to that frame or by an access to the associated storage key. The frame may become available again during a subsequent manual clear-reset operation.
2. The instruction cannot be used on the frame connected to page 0 because page 0 cannot be in the disconnected state.

## DIAGNOSE



The CPU performs built-in diagnostic functions or other implementation-dependent functions. The purpose of the diagnostic functions is to verify that CPU equipment is operating properly and to locate any faulty components. Other implementation-dependent functions may include reconfiguration of storage and channels.

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.

**Condition Code:** The code is unpredictable.

### Program Exceptions:

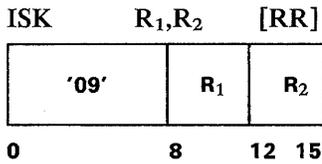
Privileged Operation  
Depending on the function, other exceptions may be recognized.

### Programming Notes

1. Since the instruction is not intended for problem-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 machine operation, including instruction execution and channel 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 machine may be left in such a condition that a power-on reset or initial microprogram loading (IML) must be performed.



## INSERT STORAGE KEY



The storage key associated with the page that is addressed by the contents of the general register designated by the R<sub>2</sub> field is inserted in the general register designated by the R<sub>1</sub> field.

Bits 8-20 of the register designated by the R<sub>2</sub> field designate the page. Bits 0-7 and 21-27 of the register are ignored. Bits 28-31 of the register must be zeros; otherwise, a specification exception is recognized, and the operation is suppressed.

The execution of the instruction depends on whether the PSW specifies the EC or BC mode. In the EC mode, the seven-bit storage key is inserted in bit positions 24-30 of the register designated by the R<sub>1</sub> field, and bit 31 is set to zero. In the BC mode, bits 0-4 of the storage key are placed in bit positions 24-28 of that register, and bits 29-31 of the register are set to zeros. In both modes, the contents of bit positions 0-23 of the register remain unchanged.

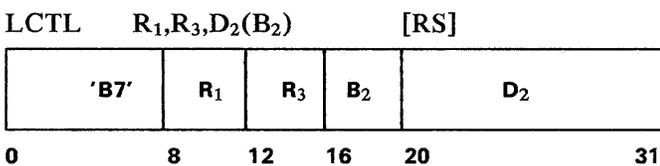
The reference to the storage key is not subject to a protection exception. The storage key can be accessed regardless of the state of the addressed page.

**Condition Code:** The code remains unchanged.

### Program Exceptions:

Addressing (operand 2)  
Privileged Operation  
Specification

## LOAD CONTROL



The set of control registers starting with the control register designated by the R<sub>1</sub> field and ending with the control register designated by the R<sub>3</sub> field is loaded from the locations designated by the second-operand address.

The storage area from which the contents of the control registers are obtained starts at the location designated by the second-operand address and continues through as many storage words as the number of control registers specified. The control registers are loaded in ascending order of their addresses, starting with the control register designated by the R<sub>1</sub> field and continuing up to and including the control register designated by the R<sub>3</sub> field, with control register 0 following control register 15. The second operand remains unchanged.

A word of the operand is fetched from storage for each of the designated control registers, regardless of whether the control register has any assigned positions. Whenever the storage reference causes an access exception, the exception is indicated. The information fetched for unassigned control-register positions may be loaded or ignored.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized, and the operation is suppressed.

**Condition Code:** The code remains unchanged.

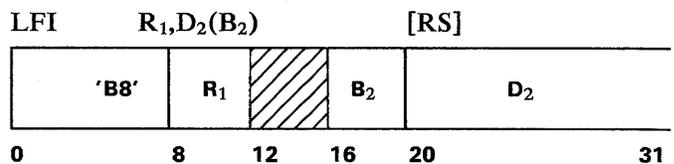
### Program Exceptions:

Access (fetch, operand 2)  
Privileged Operation  
Specification

### Programming Note

To ensure that existing programs run if and when new facilities using additional control-register positions are defined, only zeros should be loaded in unassigned control-register positions.

## LOAD FRAME INDEX



The frame index of the page frame that is connected to the storage page designated by the second-operand address is returned in the general register designated by the R<sub>1</sub> field.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored. Bits 12-15 of the instruction are ignored.

The frame index is an unsigned binary integer. It is right-aligned in the R<sub>1</sub> register, and the remaining

high-order bits of the register are set to zeros. The frame index is unique and may have any value from zero to EFCC - 1, where EFCC is the existing-frame-capacity count.

The frame index is returned only when the page is connected or addressable. When the page is disconnected or not provided (condition code 2 or 3), the R<sub>1</sub> register remains unchanged.

Condition code 0, 1, or 2 is set when the page is addressable, connected, or disconnected, respectively. Condition code 3 is set when the address is invalid, that is, the value of bits 8-20 of the second-operand address equals or exceeds the page-capacity count.

**Resulting Condition Code:**

- 0 Index returned, page is addressable
- 1 Index returned, page is connected
- 2 Index not returned, page is disconnected
- 3 Index not returned, address is invalid

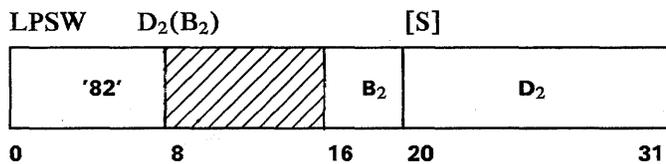
**Program Exceptions:**

Privileged Operation

**Programming Note**

The instruction may be used to test the page address and state of a page and return its frame index, if any, without raising an access exception.

**LOAD PSW**



The current PSW is replaced by the contents of the doubleword at the location designated by the second-operand address.

If the new PSW specifies the BC mode, information in bit positions 16-33 of the new PSW is not retained as the PSW is loaded. When the PSW is subsequently stored, these bit positions contain the new interruption code and the instruction-length code.

A serialization function is performed.

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on protection and addressing exceptions.

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 the newly loaded PSW specifies the EC mode and the contents of bit positions 0, 2-5, 16-17, and 24-39 are not all zeros. In these cases, the operation is completed, and the resulting instruction-length code is zero.

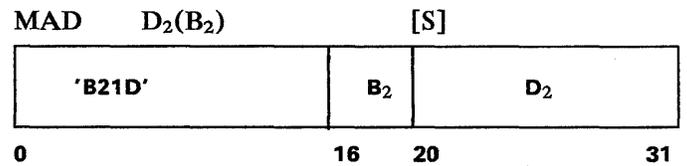
Bits 8-15 of the instruction are ignored.

**Resulting Condition Code:** The code is that specified in the new PSW loaded.

**Program Exceptions:**

Access (fetch, operand 2)  
Privileged Operation  
Specification

**MAKE ADDRESSABLE**



If connected, the storage page designated by the second-operand address enters the addressable state. If already addressable, the page remains in the addressable state.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

If the page is in the disconnected state, a page-transition exception is recognized, and the operation is suppressed.

The condition code indicates whether the page was addressable (1) or connected (0) before.

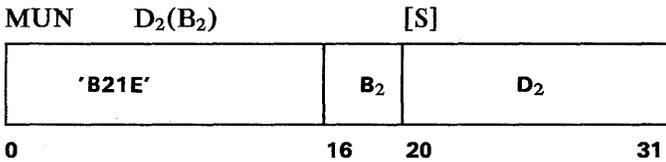
**Resulting Condition Code:**

- 0 Page was connected
- 1 Page was already addressable
- 2 -
- 3 -

**Program Exceptions:**

Addressing (operand 2)  
Page Transition  
Privileged Operation

## MAKE UNADDRESSABLE



If addressable, the storage page designated by the second-operand address enters the connected state. If already connected, the page remains in the connected state.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

If bits 8-20 of the second-operand address are zeros, that is, page 0 is specified, a specification exception is recognized, and the operation is suppressed.

If the page is in the disconnected state, a page-transition exception is recognized, and the operation is suppressed.

The condition code indicates whether the page was addressable (0) or connected (1) before.

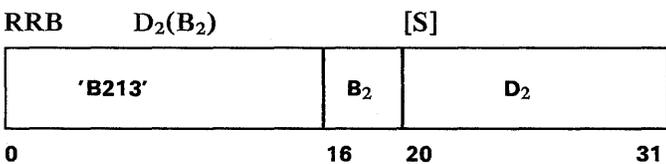
### Resulting Condition Code:

- 0 Page was addressable
- 1 Page was already connected
- 2 -
- 3 -

### Program Exceptions:

Addressing (operand 2)  
Page Transition  
Privileged Operation  
Specification

## RESET REFERENCE BIT



The reference bit in the storage key associated with the storage page that is designated by the second-operand address is set to zero.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored.

The condition code is set to reflect the state of the reference and change bits before the reference bit is set to zero.

The references to the storage key are not subject to protection exceptions. The storage key can be accessed regardless of the state of the addressed page. The values of the remaining bits of the storage key, including the change bit, are not affected.

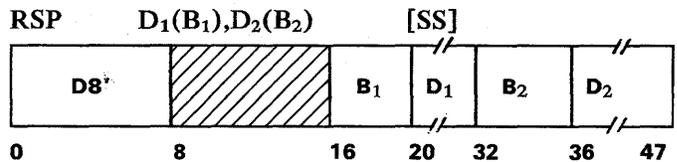
### Resulting Condition Code:

- 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 (operand 2)  
Privileged Operation

## RETRIEVE STATUS AND PAGE



The saved machine status is retrieved and stored at the first-operand location. The contents of the saved page are retrieved and stored at the second-operand location.

The saved machine status, as retrieved, consists of 256 bytes reflecting the state of the machine at the last time that the manual machine-save operation was performed. (See the figure "Machine Status, Retrieval Format" in Chapter 4, "Control," for the contents.) The saved page consists of the contents at that time of page 0. The storage key, page bits, and frame index for the saved page are contained in the machine status.

If the two operands overlap, the results are unpredictable.

If the saved information is valid, condition code 0 is set. If the saved information is invalid, neither storage operand is accessed, no access exceptions are recognized, and condition code 3 is set.

The saved machine status and page remain unchanged.

**Resulting Condition Code:**

- 0 Save information is valid
- 1 -
- 2 -
- 3 Save information is invalid

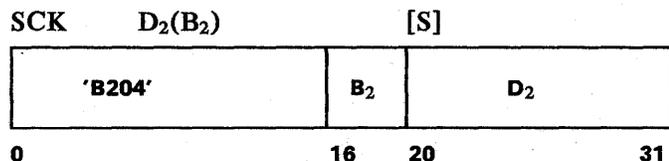
**Program Exceptions:**

Access (store, operand 1 and 2)  
Privileged Operation

**Programming Notes**

1. The saved information may be found invalid if a partially performed machine save was canceled by resetting the machine. The saved information is invalid if a clear reset has been performed since the last machine save. RETRIEVE STATUS AND PAGE will indicate an invalid save until another machine save is performed.
2. Two executions of RETRIEVE STATUS AND PAGE will retrieve the same status and page information, as long as the information has not been invalidated by a reset and no machine save has intervened.

**SET CLOCK**



The current value of the time-of-day clock is replaced by the contents of the doubleword designated by the second-operand address, and the clock enters the set 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.

The value of the clock is changed and the clock is placed in the set state only if the manual TOD-clock control is set to enable-set. If the TOD-clock control is set to secure, the value and the state of the clock are not changed. The two results are 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

setting of the TOD-clock control, and condition code 3 is set.

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. Access exceptions are recognized regardless of the state of the clock and the setting of the TOD-clock control.

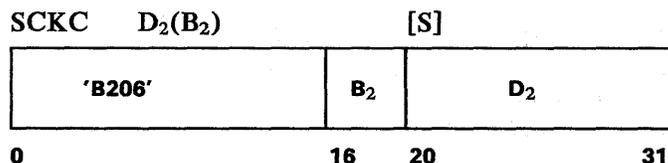
**Resulting Condition Code:**

- 0 Clock value set
- 1 Clock value secure
- 2 -
- 3 Clock in not-operational state

**Program Exceptions:**

Access (fetch, operand 2)  
Privileged Operation  
Specification

**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 time-of-day clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock comparator.

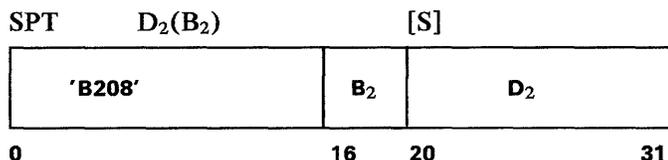
The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on protection and addressing exceptions.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Access (fetch, operand 2)  
Privileged Operation  
Specification

## 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 which are updated by the CPU timer; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the CPU timer.

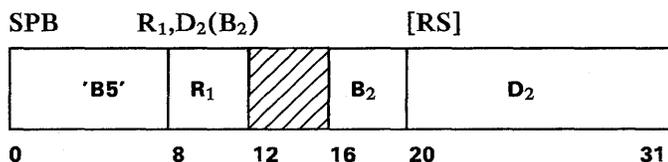
The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on protection and addressing exceptions.

**Condition Code:** The code remains unchanged.

### Program Exceptions:

Access (fetch, operand 2)  
Privileged Operation  
Specification

## SET PAGE BITS



The current settings of the three programmable page bits and the reference and change bits that are associated with the storage page designated by the second-operand address are replaced by the contents of the general register designated by the R<sub>1</sub> field.

Bits 8-20 of the second-operand address designate the page. Bits 0-7 and 21-31 of the address are ignored. Bits 12-15 of the instruction are ignored.

The condition code is set to reflect the state of the reference and change bits before these bits are modified.

The new values of the three page bits are obtained from bit positions 25-27, and the reference and change bits from bit positions 29-30 of the register designated by the R<sub>1</sub> field. The contents of bit positions 0-24, 28, and 31 of the register are ignored.

The references to the page bits and to the reference and change bits are not subject to a protection exception. These bits can be accessed regardless of the state of the addressed page.

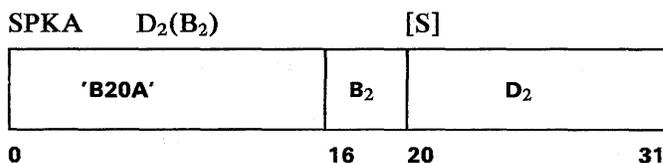
### Resulting Condition Code:

- 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 (operand 2)  
Privileged Operation

## SET PSW KEY FROM ADDRESS



The four-bit PSW key, bits 8-11 of the current PSW, is replaced by bits 24-27 of the second-operand address.

The second-operand address is not used to address data; instead, bits 24-27 of the address form the new PSW key. Bits 8-23 and 28-31 of the second-operand address are ignored.

**Condition Code:** The code remains unchanged.

### Program Exceptions:

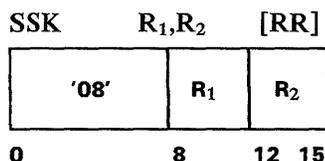
Privileged Operation

### Programming Notes

- The format of the SET PSW KEY FROM ADDRESS instruction permits the program to set the PSW key either from the general register designated by the B<sub>2</sub> field or from the D field in the instruction itself.
- When a problem program requests a control program to access a location specified by the problem program, the SET PSW KEY FROM ADDRESS instruction can be used by the control program to verify that the problem program is authorized to make this access, provided the storage location of the control program is not protected against fetching. The control program can perform the verification by replacing the

PSW key of the control program with the problem-program PSW key before making the access and subsequently restoring the control-program PSW key to its original value.

### SET STORAGE KEY



The storage key associated with the page that is addressed by the contents of the general register designated by the R<sub>2</sub> field is replaced by the contents of the general register designated by the R<sub>1</sub> field.

Bits 8-20 of the register designated by the R<sub>2</sub> field designate the page. Bits 0-7 and 21-27 of the register are ignored. Bits 28-31 of the register must be zeros; otherwise, a specification exception is recognized, and the operation is suppressed.

The new seven-bit storage-key value is obtained from bit positions 24-30 of the register designated by the R<sub>1</sub> field. The contents of bit positions 0-23 and 31 of that register are ignored.

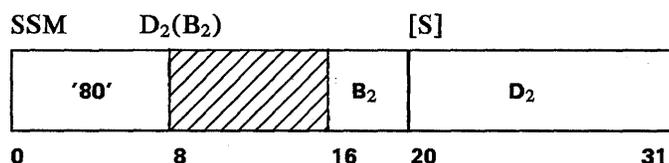
The reference to the storage key is not subject to a protection exception. The storage key can be accessed regardless of the state of the addressed page.

**Condition Code:** The code remains unchanged.

#### Program Exceptions:

Addressing (operand 2)  
Privileged Operation  
Specification

### SET SYSTEM MASK



Bits 0-7 of the current PSW are replaced by the byte at the location designated by the second-operand address.

When the SSM-suppression bit, bit 1 of control register 0, is one and the CPU is in the supervisor

state, a special-operation exception is recognized, and the operation is suppressed.

The operation is suppressed on protection and addressing exceptions.

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 CPU is in the EC mode and the contents of bit positions 0 and 2-5 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2.

Bits 8-15 of the instruction are ignored.

**Condition Code:** The code remains unchanged.

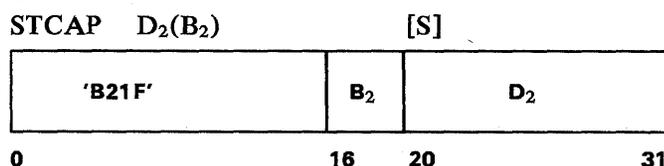
#### Program Exceptions:

Access (fetch, operand 2)  
Privileged Operation  
Special Operation  
Specification (EC mode only)

#### Programming Note

The SSM instruction is frequently used in the BC mode to disable or enable the CPU for I/O or external interruptions. Hence, suppressing the execution of the SSM instruction by means of the SSM-suppression bit, bit 1 of control register 0, may be useful when converting a program written for a BC-mode PSW to operate with an EC-mode PSW.

### STORE CAPACITY COUNTS



The current values of the page-capacity (PCC), existing-frame-capacity (EFCC), available-frame-capacity (AFCC), and free-frame-capacity (FFCC) counts are stored at the 16-byte location designated by the second-operand address. The counts are stored as 32-bit unsigned binary integers in the order, from left to right, of PCC, EFCC, AFCC, and FFCC.

**Condition Code:** The code remains unchanged.

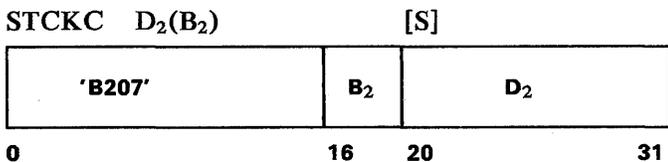
**Program Exceptions:**

Access (store, operand 2)  
Privileged Operation

**Programming Notes**

1. The instruction allows the program to display the current values of the PCC, EFCC, AFCC, and FFCC for initialization purposes at IPL time and for the management of virtual storage and real storage.
2. The high-order 16 bits of each counter value, as stored, are always zeros. The counter values cannot exceed 65,535.

**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 time-of-day clock.

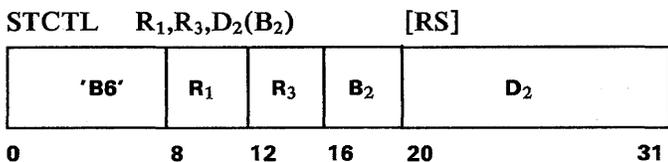
The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed. The operation is suppressed on protection and addressing exceptions.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Access (store, operand 2)  
Privileged Operation  
Specification

**STORE CONTROL**



The set of control registers starting with the control register designated by the R<sub>1</sub> field and ending with the control register designated by the R<sub>3</sub> field is

stored at the locations designated by 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 as the number of control registers specified. The contents of the control registers are stored in ascending order of their addresses, starting with the control register designated by the R<sub>1</sub> field and continuing up to and including the control register designated by the R<sub>3</sub> field, with control register 0 following control register 15. The contents of the control registers remain unchanged.

A word is stored for each of the designated control registers, regardless of whether the control register has any assigned positions. Whenever the storage reference causes an access exception, the exception is indicated. The information stored for unassigned control-register positions is unpredictable.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized, and the operation is suppressed.

**Condition Code:** The code remains unchanged.

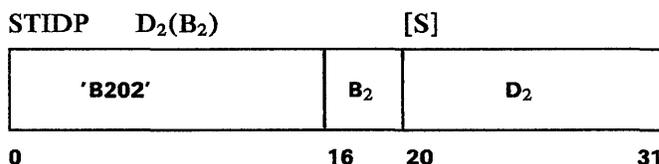
**Program Exceptions:**

Access (store, operand 2)  
Privileged Operation  
Specification

**Programming Note**

Although STORE CONTROL may provide zeros in the bit positions corresponding to unassigned register positions, the program should not depend on such zeros.

**STORE CPU ID**



Information identifying the CPU is stored at the doubleword location designated by the second-operand address.



**Condition Code:** The code remains unchanged.

**Program Exceptions:**

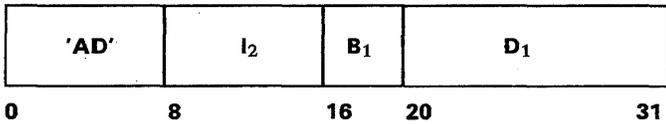
Access (store, operand 1)  
Privileged Operation

**Programming Note**

The STORE THEN AND SYSTEM MASK instruction permits the program to set selected bits in the system mask to zeros while retaining the original contents for later restoration.

**STORE THEN OR SYSTEM MASK**

STOSM D<sub>1</sub>(B<sub>1</sub>),I<sub>2</sub> [SI]



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.

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 CPU is in the EC mode and the contents of bit positions 0 and 2-5 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2.

The operation is suppressed on protection and addressing exceptions.

**Condition Code:** The code remains unchanged.

**Program Exceptions:**

Access (store, operand 1)  
Privileged Operation  
Specification (EC mode only)

**Programming Note**

The STORE THEN OR SYSTEM MASK instruction permits the program to set selected bits in the system mask to ones while retaining the original contents for later restoration.

# Chapter 11. Machine-Check Handling

## Contents

Machine-Check Detection	11-1	Interruption Action	11-4
Correction of Machine Malfunctions	11-1	Point of Interruption	11-5
Handling of Machine Checks	11-2	Machine-Check-Interruption Code	11-6
Invalid CBC in Storage	11-2	Subclass	11-7
Invalid CBC in Page Descriptions	11-2	Auxiliary Bits	11-8
Invalid CBC in Registers	11-3	Machine-Check Interruption-Code Validity Bits	11-8
Usage of Validation	11-3	Machine-Check Extended Interruption Information	11-9
Check-Stop State	11-3	Register-Save Areas	11-9
Machine-Check Interruption	11-4	Failing-Storage Address	11-9
Exigent Conditions	11-4	Machine-Check Masking	11-9
Repressible Conditions	11-4		

The machine-check-handling mechanism provides machine-malfunction detection to ensure the integrity of machine operation and permit automatic recovery from some malfunctions. Machine malfunctions are reported by machine-check interruptions to assist in program-damage assessment and recovery.

### 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 will 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 and is less than or equal to 2,048 bytes of data.

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 entire contents of a checking block, including the redundancy, is called a 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

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 take two 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.

Some models have the capability of correcting intermittent errors by retrying CPU operations. When a malfunction is detected, recovery is attempted by returning the CPU state to that existing at the checkpoint when information about the CPU state was last saved, and proceeding from that point.

Machine failures which are corrected successfully may be reported by machine-check interruptions as system-recovery conditions. This permits the program to note the cause of CPU delay and to keep a log of such incidents.

## Handling of Machine Checks

A machine check can be caused only by a machine malfunction and never 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 page descriptions, and in all available page frames.

Specification of an unavailable component, such as a channel 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 has been configured out of the system results in an addressing exception and does not generate a machine-check condition, even though the storage location or its associated storage key has invalid CBC.

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.

When an operation alters a checking block, invalid CBC is preserved as invalid unless the contents of the entire checking block are replaced in the operation.

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. A malfunction detected as part of an I/O operation may cause a machine-check 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 a CCW or data with invalid CBC is fetched from storage but is not used in an I/O operation, the condition may or may not be reported.

When the machine reports a failing storage location detected during an I/O operation, both I/O-error and machine-check conditions are presented. The I/O-error condition is the primary indication to the program. The machine-check condition is a secondary report, which is indicated as system recovery together with a failing-storage address.

A malfunction, detected as part of an operation that is not I/O, can only cause a machine-check condition. Machine-check conditions may be reported as machine-check interruptions, or they may cause the CPU to enter the check-stop state.

## Invalid CBC in Storage

An attempt to store into a checking block with invalid CBC, without replacing the entire checking block, leaves the data in the checking block (including the check bits) unchanged.

When the checking block consists of multiple bytes and contains invalid CBC, special procedures are necessary to place new information into the checking block. Placing valid CBC in storage is called storage validation.

Storage validation is provided as a program function and is also provided with the manual clear-reset function. Programmed storage validation is done, one page at a time, by executing the privileged instruction CLEAR PAGE. Manual storage validation by clear reset validates all pages.

## Invalid CBC in Page Descriptions

When invalid CBC is detected in a page description, a machine-check interruption may occur; depending on the circumstances, the machine-check condition may be system damage, instruction-processing damage, system recovery, or external damage. The machine-check condition may or may not be accompanied by a storage-key-error indication. Also, if invalid CBC in a page description is detected during an I/O operation, a channel-control check is normally indicated at the end of the I/O operation.

In addition to internal storage for page descriptions, some models may have a separate lookaside storage for the storage keys of connected or addressable pages. Each entry of such a lookaside is associated with a page frame, whereas each page description is associated with a page. A storage-key error may be indicated only when invalid CBC is detected in the lookaside storage during a reference to the storage key of a page that is in the connected or addressable state.

A storage-key error is not indicated when:

- Invalid CBC is detected in the storage key of a disconnected page
- Invalid CBC is detected in the page bits, the page state, or the frame index of a page, whether disconnected or not
- No lookaside storage is provided for storage keys

All parts of the page descriptions are validated manually by clear reset. On models which provide lookaside storage with a separate checking block for the storage key of each connected or addressable page, executing the instruction SET STORAGE KEY sets new values for and validates the storage key after a storage-key error has been indicated. The

instruction CONNECT PAGE may validate the lookaside entry of a page frame which previously had invalid CBC by using the values of the storage key from the page-description entry.

No storage-key-error indication is given when a machine check occurs during the execution of DECONFIGURE PAGE, DISCONNECT PAGE, LOAD FRAME INDEX, MAKE ADDRESSABLE, and MAKE UNADDRESSABLE.

Any machine-check condition which would otherwise be indicated as a storage-key error is ignored if the access key is zero when a fetch operation takes place. Depending on the model, a storage-key error may or may not be ignored if the access key is zero when a store operation takes place or when the instruction CLEAR PAGE is executed.

The CPU enters the check-stop state when invalid CBC is detected in the page description for page 0, and also when a page description is left in an inconsistent state after an error occurs while the page description is being updated.

#### **Programming Note**

Recovery from a storage-key error which cannot be successfully removed by issuing SET STORAGE KEY may be attempted by issuing DECONFIGURE PAGE to delete the page frame and CONNECT PAGE to use another page frame. The previous contents of the page are lost.

#### **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, and control registers, the current PSW, the time-of-day clock, the CPU timer, and the clock comparator.

When a machine-check interruption occurs that is due to invalid CBC in a CPU register, the following actions are taken as part of the interruption.

1. As for all machine-check interruptions, the contents of the CPU registers, other than the time-of-day-clock, are saved in assigned storage locations. The type of register that is in error is identified, unless it is the time-of-day clock, by a corresponding validity bit of zero in the machine-check-interruption code. Register saving does not result in additional machine-check-interruption conditions; instead, the accuracy of all the information stored is indicated by the appropriate setting of the validity bits.

2. CPU registers with invalid CBC, other than the time-of-day clock, are then validated, their actual contents being unpredictable.

CPU registers other than the time-of-day clock are also validated manually by the clear-reset function; programmed validation is not provided.

The time-of-day clock enters the error state when a malfunction is detected in the clock. It is validated by programming when a SET CLOCK instruction changes the state of the clock from the error state to the set state. The clock is also validated manually by a power-on reset.

#### **Usage of Validation**

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.

Validating a checking block does not ensure a valid CBC. If the failure is solid, the checking block will still contain an invalid CBC after validation. For an intermittent failure, however, validation is useful to restore a valid CBC such that a subsequent partial store into the checking block (a store into a checking block without replacing the entire checking block) by either the CPU or a channel will be permitted.

#### **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 when:

- 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
- A machine-check interruption cannot be taken because of a storage error in page 0

- Invalid CBC is detected in the page description for page 0
- An error occurs while a page description is being updated, leaving the page description in an inconsistent 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 interval timer is not updated, and channel operations may be stopped. The time-of-day 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. The start key and stop key are not effective in this state.

The CPU may be removed from the check-stop state by program reset.

## **Machine-Check Interruption**

The machine-check interruption provides a means of reporting equipment malfunctions and certain external disturbances, and it supplies the program with information about the extent of the resultant damage and the location and nature of the cause. There are two major types of conditions which can cause a machine-check interruption: exigent conditions and repressible conditions.

### ***Exigent Conditions***

Exigent machine-check-interruption conditions are those in which direct damage has occurred to CPU operation, so that the current instruction or interruption cannot safely continue. Exigent conditions are divided into two subclasses: instruction-processing damage and system damage. Malfunctions which cannot be isolated to a specific function are indicated as system damage.

### ***Repressible Conditions***

Repressible machine-check-interruption conditions are those in which the sequential processing capability of the CPU has not been damaged. 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 classes: recovery, alert, and repressible damage. Each class has one or more subclasses.

A malfunction in the CPU, storage, channels, or operator facilities, which has been successfully corrected or circumvented internally without loss of machine integrity, 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 the loss of integrity of a machine function but not directly affecting sequential CPU operation is called a repressible-damage condition. Repressible-damage conditions are divided into three subclasses, according to the function affected: interval-timer damage, timing-facility damage, and external damage.

### **Programming Note**

Classification of a damage condition as repressible does not imply that the damage is necessarily less severe than damage classified as an exigent condition. 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; 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, a channel encounters an independent storage error, which is an external-damage condition, 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 that damaged area of storage 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.

### ***Interruption Action***

A machine-check interruption causes the PSW reflecting the point of interruption to be stored as the machine-check old PSW at location 48. The contents of other registers are stored in register save areas at locations 216-231 and 352-511. A failing-storage address, if any, is stored at location 248. Then a machine-check-interruption code (MCIC) of eight

bytes is placed at location 232. The new PSW is fetched from location 112.

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 cause a machine-check interruption only if both PSW bit 13 and the associated subclass mask bit in control register 14 are ones. 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 repressible machine-check condition 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. Only one repressible condition is held pending for each subclass, regardless of the number of conditions that may have been detected.

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 in the same interruption code, even though the CPU is disabled for those subclasses. All indicated conditions are then cleared.

If 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 can cause a machine-check interruption only when PSW bit 13 is one. When it occurs, 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 steps, 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 condition is detected, the CPU enters the check-stop state. If an exigent machine-check

condition 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 in both the nonwait and wait states. In the wait state, a machine-check condition causes an immediate interruption if the CPU is enabled for that condition.

Machine checks which occur while the rate control is set to instruction step are handled in the same manner as when the control is set to process; that is, recovery and machine-check interruptions occur when allowed.

Every reasonable attempt is made to limit the side effects of any machine-check condition and the associated interruption. Normally, I/O and external interruptions, as well as the progress of I/O data transfer and the updating of the CPU timer, 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 may terminate the process of switching PSWs that is due to another type of interruption. In these cases, system damage will be indicated.

### ***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.

Only certain points in the processing may be used as a point of interruption. For repressible machine checks, 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 can occur at any point during instruction processing. The point of interruption may be after the unit of operation is completed but before any associated program or supervisor-call interruption occurs. In this case, a valid PSW is defined as that which would have been stored in the old PSW for the program or supervisor-call interruption. Even though all status may be indicated as valid, damage has occurred because the associated interruption is lost.

When the point of interruption for an exigent machine-check condition occurs after an associated



### Programming Note

The program should not depend on unassigned bits in the machine-check-interruption code being zeros, so as to ensure that existing programs run if and when new facilities using these bits are defined.

### Subclass

Bits 0-5, 7, and 8 identify the type of machine-check condition causing the interruption. At least one of the defined subclass bits is stored as a one. When multiple errors have occurred, several of the defined bits may be set to ones.

**System Damage (SD):** Bit 0, when one, indicates that damage has occurred which cannot be isolated to one or more of the less severe machine-check damage subclasses. System damage is an exigent condition.

**Instruction-Processing Damage (PD):** Bit 1, when one, indicates that a malfunction has been detected in the processing of an instruction. Detection of such damage is synchronous with CPU operation.

For damage to be indicated as instruction-processing damage, the damaged instruction and the point of interruption must not be separated by an interruption or by a LOAD PSW instruction, and the extent of the damage must fall within one or more of the following categories:

1. The damaged area still contains invalid CBC.
2. The damaged area lies within the destination operand of the instruction.
3. The damaged area lies within the general registers, floating-point registers, control registers, or PSW.

Instruction-processing damage is an exigent condition.

**System Recovery (SR):** Bit 2, when one, indicates that malfunctions were detected but have been successfully corrected or circumvented without the loss of machine integrity. CPU-detected malfunctions are reported as system recovery only if the CPU successfully completes the operation or unit of operation in which the malfunction was detected. Some I/O-detected malfunctions may result in a system-recovery condition in addition to an I/O-error condition. 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. The presence and extent of the system-recovery capability depend on the model.

System recovery is a repressible condition.

**Interval-Timer Damage (TD):** Bit 3, when one, indicates that damage has occurred to the interval timer or to storage location 80. Interval-timer damage is a repressible condition.

**Timing-Facility Damage (CD):** Bit 4, when one, indicates that damage has occurred to the time-of-day clock, the CPU timer, or the clock comparator. The timing-facility-damage machine-check condition is set whenever any of the following occurs:

1. The time-of-day clock enters the error or not-operational state.
2. The CPU timer is damaged, and the CPU is enabled for CPU-timer external interruptions.
3. The CPU timer is damaged, and SET CPU TIMER or STORE CPU TIMER is executed. This condition also sets instruction-processing damage.
4. The clock comparator is damaged, and the CPU is enabled for clock-comparator external interruptions.
5. The clock comparator is damaged, and SET CLOCK COMPARATOR or STORE CLOCK COMPARATOR is executed. This condition also sets instruction-processing damage.

Timing-facility damage is a repressible condition.

**External Damage (ED):** Bit 5, when one, indicates that damage has occurred to a channel or to storage during operations not directly associated with processing the current instruction. Channel malfunctions are reported as external damage only when the channel is unable to report the malfunctions by an I/O interruption. Depending on the model and on the type and extent of the error, an external-damage condition may be indicated as system damage instead of external damage.

External damage is a repressible condition.

**Degradation (DG):** Bit 7, 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. The presence and extent of the degradation-report capability depends on the model.

Degradation is a repressible condition.

**Warning (W):** Bit 8, 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.

Warning is a repressible condition.

#### Programming Notes

1. On some models with CPU retry, a CPU-retry operation may delay the response to a concurrent I/O operation, thus causing an I/O overrun or chaining check and, hence, an I/O-interruption condition. Consequently, when a machine-check-interruption condition indicating system recovery and such an I/O-interruption condition occur at approximately the same time, the overrun may be due to a CPU delay rather than to timing problems with the I/O device and channel.
2. Timing-facility-damage interruptions for the CPU timer and the clock comparator are disallowed 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 submask bits, bits 21 and 20 of control register 0, are zero), and when the corresponding set and store instructions are not being issued. Timing-facility-damage interruptions due to damage to the time-of-day clock cannot be so disallowed.

#### Auxiliary Bits

Bits 15, 16, and 18 of the machine-check-interruption code may occur together with one or more of the bits in the subclass field to indicate a delayed condition, an uncorrected storage error, and an uncorrected key error, respectively.

**Delayed (D):** Bit 15, when one, indicates that one or more of the repressible machine-check conditions being reported were delayed because, at the time a particular error was detected, the CPU was disabled for that type of interruption. The bit does not apply to exigent conditions, which cannot be delayed.

**Storage Error Uncorrected (SE):** Bit 16, when one, indicates invalid CBC for the contents of a storage page that is in the connected or addressable state.

**Storage-Key Error Uncorrected (KE):** Bit 18, when one, indicates invalid CBC for the storage key in lookaside storage which is associated with a storage page that is in the connected or addressable state.

#### Programming Note

The storage-error and storage-key-error bits do not in themselves indicate the occurrence of damage because the error detected may not have affected a result. The accompanying subclass bits of the interruption code indicate the area affected by the error.

#### Machine-Check Interruption-Code Validity Bits

Bits 20-24, 27-29, 31, 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 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 will attempt, when possible, to place valid CBC in the storage field and thus reduce the possibility of additional machine checks being caused.

**PSW EMWP Validity (WP):** Bit 20, when one, indicates that the EMWP bits (bits 12-15) of the machine-check old PSW are correct.

**PSW Mask and Key Validity (MS):** Bit 21, when one, indicates that the system mask and the PSW key (bits 0-11) of the machine-check old PSW are correct.

**PSW-Program-Mask and Condition-Code Validity (PM):** Bit 22, when one, indicates that the program mask and condition code of the machine-check old PSW are correct.

**PSW-Instruction-Address Validity (IA):** Bit 23, when one, indicates that the instruction address (bits 40-63) of the machine-check old PSW is correct.

**Failing-Storage-Address Validity (FA):** Bit 24, when one, indicates that a correct failing-storage address has been placed at location 248 after an uncorrected

storage error or storage-key error. When no such errors are reported, that is, bits 16 and 18 of the machine-check-interruption code are zeros, the failing-storage address is meaningless, even though it may be indicated as valid.

**Floating-Point-Register Validity (FP):** Bit 27, when one, indicates that the contents of the floating-point-register save area at locations 352-383 reflect the correct state of the floating-point registers at the point of interruption.

**General-Register Validity (GR):** Bit 28, when one, indicates that the contents of the general-register save area at locations 384-447 reflect the correct state of the general registers at the point of interruption.

**Control-Register Validity (CR):** Bit 29, when one, indicates that the contents of the control-register save area at locations 448-511 reflect the correct state of the control registers at the point of interruption.

**Storage Logical Validity (ST):** Bit 31, when one, indicates that the contents of those storage locations 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.

**CPU-Timer Validity (CT):** Bit 46, when one, indicates that the CPU timer is not in error and that the contents of the CPU-timer save area at location 216 reflect the correct state of the CPU timer at the time the interruption occurred.

**Clock-Comparator Validity (CC):** Bit 47, when one, indicates that the clock comparator is not in error and that the contents of the clock-comparator save area at location 224 reflect the correct state of the clock comparator.

## **Machine-Check Extended Interruption Information**

During a machine-check interruption, the current PSW is stored as the machine-check old PSW at storage location 48. The current contents of the other CPU registers, except for the time-of-day clock, are stored in five register-save areas assigned in storage. When an uncorrected storage error or storage-key error is indicated, the failing-storage address is also saved.

Each of these fields has associated with it one or more validity bits in the machine-check-interruption code. If, for any reason, the machine cannot store one of these fields or cannot store the field validly, the associated validity bit is set to zero.

### **Register-Save Areas**

The following are the five sets of registers and the locations in storage where their contents are saved during a machine check:

<i>Locations</i>	<i>Registers</i>
216-223	CPU timer
224-231	Clock comparator
352-383	Floating-point registers 0, 2, 4, 6
384-447	General registers 0-15
448-511	Control registers 0-15

The information stored for unassigned control-register positions is unpredictable.

### **Failing-Storage Address**

When an uncorrected storage error or storage-key error is indicated in the machine-check-interruption code, the associated address, called the failing-storage address, is stored in bits 8-31 of the word at location 248. Bits 0-7 of that word are set to zeros.

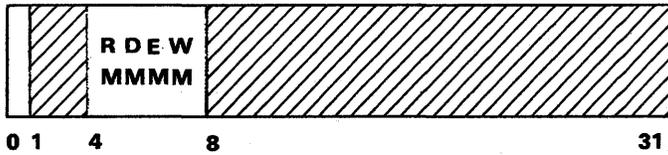
The failing-storage address may be the address of any location within the page that is in error or that is associated with the storage key in error. When an error is detected in more than one location before the interruption, the failing-storage address may point to any of the failing locations.

## **Machine-Check Masking**

The exigent machine-check conditions (system damage and instruction-processing damage) are controlled only by the machine-check mask, PSW bit 13. When the mask bit is one, an exigent condition causes a machine-check interruption. When the mask is zero, the occurrence of an exigent machine-check condition causes the CPU to enter the check-stop state.

The repressible machine-check conditions (system recovery, interval-timer damage, timing-facility damage, external damage, degradation, and warning) are controlled both by the machine-check mask, PSW bit 13, and by four 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 the subclass mask is zero, the associated condition does not initiate an interruption, but the condition may be presented with another condition which initiates the interruption. All conditions presented are then cleared.

Control Register 14



Bits 4-7 of control register 14 are the submasks for repressible machine-check conditions. In addition,

bit 0 is initialized to one, but it is otherwise ignored by the machine. All other bits of control register 14 are unassigned.

**Recovery Report Mask (RM):** Bit 4 of control register 14 controls system-recovery-interruption conditions. This bit is initialized to zero.

**Degradation-Report Mask (DM):** Bit 5 of control register 14 controls degradation-interruption conditions. This bit is initialized to zero.

**External-Damage-Report Mask (EM):** Bit 6 of control register 14 controls the following machine-check-interruption conditions: interval-timer damage, timing-facility damage, and external damage. This bit is initialized to one.

**Warning Mask (WM):** Bit 7 of control register 14 controls warning conditions. This bit is initialized to zero.

# Chapter 12. Input/Output Operations

## Contents

Attachment of Input/Output Devices	12-2
Input/Output Devices	12-2
Control Units	12-2
Channels	12-3
Modes of Operation	12-3
Types of Channels	12-4
I/O-System Operation	12-5
Compatibility of Operation	12-7
Control of Input/Output Devices	12-7
Input/Output Device Addressing	12-7
States of the Input/Output System	12-8
Resetting of the Input/Output System	12-11
I/O System Reset	12-11
I/O Selective Reset	12-11
Effect of Reset on a Working Device	12-11
Reset Upon Malfunction	12-12
Condition Code	12-12
Instruction Formats	12-14
Instructions	12-15
CLEAR I/O	12-16
HALT DEVICE	12-18
HALT I/O	12-21
START I/O	12-23
START I/O FAST RELEASE	12-23
STORE CHANNEL ID	12-26
TEST CHANNEL	12-27
TEST I/O	12-27
Input/Output-Instruction-Exception Handling	12-30
Execution of Input/Output Operations	12-30
Blocking of Data	12-30
Channel-Address Word	12-30
Channel-Command Word	12-31
Command Code	12-32
Designation of Storage Area	12-32
Chaining	12-33
Data Chaining	12-35
Command Chaining	12-36
Skipping	12-36
Program-Controlled Interruption	12-36
Commands	12-37
Write	12-39
Read	12-39
Read Backward	12-39
Control	12-40
Sense	12-40
Transfer in Channel	12-42
Command Retry	12-42
Conclusion of Input/Output Operations	12-43
Types of Conclusion	12-43
Conclusion at Operation Initiation	12-43
Immediate Operations	12-44
Conclusion of Data Transfer	12-44
Termination by HALT I/O or HALT DEVICE	12-45
Termination by CLEAR I/O	12-46
Termination Due to Equipment Malfunction	12-47
Input/Output Interruptions	12-47
Interruption Conditions	12-47
Channel-Available Interruption	12-48
Priority of Interruptions	12-48
Interruption Action	12-49
Channel-Status Word	12-49
Unit Status	12-50
Attention	12-50
Status Modifier	12-50
Control-Unit End	12-51
Busy	12-51
Channel End	12-53
Device End	12-53
Unit Check	12-53
Unit Exception	12-54
Channel Status	12-55
Program-Controlled Interruption	12-55
Incorrect Length	12-55
Program Check	12-55
Protection Check	12-56
Channel-Data Check	12-56
Channel-Control Check	12-56
Interface-Control Check	12-57
Chaining Check	12-57
Contents Of Channel-Status Word	12-57
Information Provided by Channel-Status Word	12-57
Subchannel Key	12-58
CCW Address	12-58
Count	12-59
Status	12-60
Channel Logout	12-63
I/O Communications Area	12-63

The transfer of information to or from main storage, other than to or from the central processing unit, is referred to as an input or output operation. An input/output (I/O) operation involves the use of an I/O device. Input/output devices perform I/O operations under control of control units, which are attached to the central processing unit (CPU) by means of channels.

This portion of the publication describes the programmed control of I/O devices by the channels and by the CPU. Formats are defined for the various types of I/O control information. The formats apply to all I/O operations and are independent of the type of I/O device, its speed, and its mode of operation.

The formats described include provisions for functions unique to some I/O device types, such as an erase gap on a magnetic-tape unit. The way in which a device makes use of the format is defined in the System Library (SL) publication for the particular device.

All main-storage references for I/O operations are references to virtual storage. Unless indicated otherwise, "storage" means virtual storage, and "address" means virtual address. The terms "I/O address," "channel address," and "device address" are never abbreviated to "address" in this publication.

## **Attachment of Input/Output Devices**

### ***Input/Output Devices***

Input/output devices provide external storage and a means of communication between data-processing systems or between a system and its environment. Input/output devices include such equipment as card readers, card punches, magnetic-tape units, direct-access-storage devices (disks and drums), display units, typewriter-keyboard devices, printers, teleprocessing devices, and sensor-based equipment.

Most types of I/O devices, such as printers, card equipment, or tape devices, deal directly with external media, and these devices are physically distinguishable and identifiable. Other types consist only of electronic equipment and do not directly handle physical recording media. The channel-to-channel adapter, for example, provides a channel-to-channel data-transfer path, and the data never reaches a physical recording medium outside main storage. Similarly, a transmission-control unit handles transmission of information between the

data-processing system and a remote station, and its input and output are signals on a transmission line. An I/O device may be physically distinct equipment, or it may time-share equipment with other I/O devices.

An input/output device ordinarily is attached to one control unit and is accessible from one channel. Switching equipment is available to make some devices accessible to two or more channels by switching devices between control units and control units between channels. The time required for switching occurs during device-selection time and may be ignored.

### ***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 to the standard form of control provided by the channel.

The control unit accepts control signals from the channel, controls the timing of data transfer, 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 the recording medium and recording data. To accomplish these functions, the device needs detailed signal sequences peculiar to the type of device. The control unit decodes the commands received from the channel, interprets them for the particular type of device, and provides the signal sequence required for execution of the operation.

A control unit may be housed separately, or it may be physically and logically integral with the I/O device or the 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 contain. These electromechanical devices often are of a type where only one device of a group attached to a control unit is required to operate 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 does not make specific mention of the control unit function; the execution of I/O operations is described as if the I/O devices communicated directly with the channel. Reference is made to the control unit only when emphasizing a function performed by it or when describing how sharing of the control unit among a number of devices affects the execution of I/O operations.

### **Channels**

A channel directs the flow of information between I/O devices and main storage. It relieves the CPU of the task of communicating directly with the devices and permits data processing to proceed concurrently with I/O operations.

A channel provides a means for connecting different types of I/O devices to the CPU and to storage. It accepts control information from the CPU in the format supplied by the program and changes it into a sequence of signals acceptable to a control unit. After the operation with the device has been initiated, the CPU is released for other work, and the channel assembles or disassembles data and synchronizes the transfer of data bytes with storage cycles. To accomplish this, the channel maintains and updates an address and a count that describe the destination or source of data in storage. Similarly, when an I/O device provides signals that should be brought to the attention of the program, the channel transforms the signals to information that can be used in the CPU.

A channel 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 the CPU. The only storage cycles required during I/O operations in such channels are those needed to transfer data and control information to or from the final locations in storage. These cycles do not interfere with the CPU program, except when both the CPU and the channel concurrently attempt to refer to the same storage area.

If separate equipment is not provided, facilities of the CPU are used for controlling I/O devices. When the CPU and channels, or the CPU, channels, and control units, share common facilities, I/O operations cause interference to the CPU, varying in intensity

from occasional delay of a CPU cycle to a complete lockout of CPU activity. The intensity depends on the extent of sharing and on the I/O data rate. The sharing of the facilities, however, is accomplished automatically, and the program is not affected by CPU delays, except for an increase in execution time. The effects of sharing on the CPU timer and interval timer are described in Chapter 4, "Control."

### **Modes of Operation**

An I/O operation occurs in one of two modes: burst or byte multiplex.

In burst mode, the I/O device monopolizes the channel and stays logically connected to the channel for the transfer of a burst of information. No other device can communicate with the channel 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 which monopolizes the channel.

Some channels can tolerate an absence of data transfer during a burst-mode operation, such as occurs when reading a long gap on magnetic tape, for not more than approximately 1/2 minute. Equipment malfunction may be indicated when an absence of data transfer exceeds this time.

In byte-multiplex mode, the facilities in the channel 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. During such an interval, only one device is logically connected to the channel. The intervals associated with the concurrent operation of multiple I/O devices are sequenced in response to demands from the devices. The channel controls are occupied with 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 initiation of a new operation.

Operation in burst and byte-multiplex modes is differentiated because of the way the channels respond to I/O instructions. A channel operating a device in the burst mode appears busy to new I/O instructions, whereas a channel operating one or more devices in the byte-multiplex mode is available for initiating an operation on another device. If a channel that can operate in either mode happens to be communicating with an I/O device at the instant a new I/O instruction is issued, action on the

instruction is delayed by the channel until the current mode of operation is established. Furthermore, the new I/O operation is initiated only after the channel has serviced all outstanding requests for data transfer from devices previously placed in operation.

The distinction between a short burst of data occurring in the byte-multiplex mode and an operation in the burst mode is in the length of the bursts of data. A channel that can operate in either mode determines its mode of operation by "time-out." Whenever the burst causes the device to be connected to the channel for more than approximately 100 microseconds, the channel is considered to be operating in the burst mode.

Ordinarily, devices with a high data-transfer rate operate with the channel in burst mode, and slower devices run in byte-multiplex mode. Some control units have a manual switch for setting the mode of operation.

### Types of Channels

A system can be equipped with three types of channels: selector, byte multiplexer, and block multiplexer.

The channel facilities required for sustaining a single I/O operation are termed a subchannel. The subchannel consists of internal storage used for recording the addresses, count, and any status and control information associated with the I/O operation. The capability of a channel to permit multiplexing depends upon whether it has more than one subchannel.

A selector channel, which contains a minimum of facilities, has one subchannel and always forces the I/O device to transfer data in the burst mode. The burst extends over the whole block of data, or, when command chaining is specified, over the whole sequence of blocks. A selector channel cannot perform any multiplexing and therefore can be involved in only one I/O operation or chain of operations at a time. In the meantime, other I/O devices attached to the channel can be executing previously initiated operations that do not involve communication with the channel, such as backspacing tape. When the selector channel is not executing an operation or a chain of operations and is not processing an interruption, it monitors the attached devices for status information.

A byte-multiplexer channel contains multiple subchannels and can operate in either byte-multiplex or burst mode. The channel operates most efficiently when running I/O devices that are designed to

operate in byte-multiplex mode. The mode of operation is determined by the I/O device, and the mode can change at any time. The data transfer associated with an operation can occur partially in the byte-multiplex mode and partially in the burst mode.

A block-multiplexer channel contains multiple subchannels and can only operate in burst mode. The channel operates most efficiently when running devices that are designed to operate in burst mode. When multiplexing is not inhibited, the channel permits multiplexing during operations with an I/O device, between bursts, or when command retry is performed. On most models, the burst is forced to extend over the block of data, and multiplexing is permitted between blocks of data when command chaining is specified. Whether or not multiplexing occurs depends on the design of the channel and I/O device and on the state of the block-multiplexing-control bit.

When the block-multiplexing-control bit, bit 0 of control register 0, is zero, multiplexing is inhibited; when it is one, multiplexing is allowed.

Whether a block-multiplexer channel executes an I/O operation with multiplexing inhibited or allowed is determined by the state of the block-multiplexing-control bit at the time the operation is initiated by START I/O or START I/O FAST RELEASE and applies to that operation until the involved subchannel becomes available.

Both byte-multiplexer and block-multiplexer channels vary in the number of subchannels they contain. When multiplexing, they can sustain concurrently one I/O operation per subchannel, provided that the total load on the channel does not exceed its capacity. Each subchannel appears to the program as an independent selector channel, except in those aspects of communication that pertain to the physical channel (for example, individual subchannels on a multiplexer channel are not distinguished as such by the TEST CHANNEL instruction or by the masks controlling I/O interruptions from the channel). When a multiplexer channel is not servicing an I/O device, it monitors its devices for data and for status information.

Subchannels on a multiplexer channel may be either nonshared or shared.

A subchannel is referred to as nonshared if it is associated with and can be used only by a single I/O device. A nonshared subchannel is used with devices that do not have any restrictions on the concurrency of channel-program operations, such as the IBM 3211

Printer Model 1 or one drive of an IBM 3330 Disk Storage.

A subchannel is referred to as shared if data transfer to or from a set of devices implies the use of the same subchannel. Only one device associated with a shared subchannel may be involved in data transmission at a time. Shared subchannels are used with devices, such as magnetic-tape units or some disk-access mechanisms, that share a control unit. For such devices, the sharing of the subchannel does not restrict the concurrency of I/O operations since the control unit permits only one device to be involved in a data-transfer operation at a time. I/O devices may share a control unit without necessarily sharing a subchannel. For example, each transmission line attached to the IBM 2702 Transmission Control is assigned a nonshared subchannel, although all of the transmission lines share the common control unit.

#### **Programming Notes**

A block-multiplexer channel can be made to operate as a selector channel by the appropriate setting of the block-multiplexing-control bit. However, since a block-multiplexer channel inherently can interleave the execution of multiple I/O operations and since the state of the block-multiplexing-control bit can be changed at any time, it is possible to have one or more operations that permit multiplexing and an operation that inhibits multiplexing being executed simultaneously by a channel.

Therefore, to ensure complete compatibility with selector channel operation, all operational subchannels on the block-multiplexer channel must be available or operating with multiplexing inhibited when the use of that channel as a selector channel is begun. All subsequent operations should then be initiated with the block-multiplexing-control bit inhibiting multiplexing.

#### ***I/O-System Operation***

Input/output operations are initiated and controlled by information with two types of formats: instructions and channel-command words (CCWs). Instructions are decoded by the CPU and are part of the CPU program. CCWs are decoded and executed by the channels and I/O devices and initiate I/O operations, such as reading and writing. One or more CCWs arranged for sequential execution form a channel program. Both instructions and CCWs are fetched from storage and their formats are common for all types of I/O devices, although the modifier

bits in the command code of a CCW may specify device-dependent operations.

The CPU program initiates I/O operations with the instruction START I/O or START I/O FAST RELEASE. These instructions identify the channel and device and cause the channel to fetch the channel-address word (CAW) from a fixed location in storage. The CAW contains the subchannel key and designates the location in storage from which the channel subsequently fetches the first CCW. The CCW specifies the command to be executed and the storage area, if any, to be used.

When the CAW has been fetched, some channels consider the execution of START I/O FAST RELEASE complete. The results of the execution of the instruction to that point are indicated by setting the condition code in the program-status word (PSW) and, in certain situations, by storing pertinent information in the channel-status word (CSW).

If the channel is not operating in burst mode and if the subchannel associated with the addressed I/O device is available, the channel attempts to select the device by sending the address of the device to all control units attached to the channel. A control unit that recognizes the address connects itself logically to the channel and responds to its selection by returning the address of the selected device. The channel subsequently sends the command-code part of the CCW to the control unit, and the device responds with a status byte indicating whether it can execute the command.

At this time, the execution of START I/O and of START I/O FAST RELEASE, if not previously considered complete, is completed. The results of the attempt to initiate the execution of the command are indicated by setting the condition code in the PSW and, in certain situations, by storing pertinent information in the CSW.

If the operation is initiated at the device and its execution involves transfer of data, the subchannel is set up to respond to service requests from the device and assumes further control of the operation. In the case of operations that do 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.

An I/O operation may involve transfer of data to one storage area, designated by a single CCW, or to a number of noncontiguous storage areas. In the latter case, generally a list of CCWs is used for execution of the I/O operation, each CCW designating a contiguous storage area, and the CCWs are said to be

coupled by data chaining. Data chaining is specified by a flag in the CCW and causes the channel 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 request an I/O interruption as soon as possible, thereby notifying the CPU program that chaining has progressed at least as far as that CCW.

The conclusion of an I/O operation normally is indicated by channel end and device end. Channel end indicates that the I/O device has received or provided all data associated with the operation and no longer needs channel facilities. Device end indicates that the I/O device has concluded execution of the operation. Device end can occur concurrently with channel end or later.

Operations that keep the control unit busy after releasing channel facilities may, in some situations, cause a third indication called control-unit end. Control-unit end may occur only concurrently with or after channel end and indicates that the control unit has become available for initiation of another operation.

Concurrent with channel end, both the channel and the device can provide indications of unusual situations. Control-unit end and device end can be accompanied by error indications from the device.

The indication of the conclusion of an I/O operation can be brought to the attention of the program by I/O interruptions or, when the CPU is disabled for I/O interruptions from the channel, by programmed interrogation of the I/O device. An indication that will result in an interruption or that can be observed through interrogation is called an interruption condition. In either case, a CSW is stored, which contains additional information concerning the execution of the operation. When channel end is indicated in the CSW, the CSW identifies the last CCW used and provides its residual byte count, thus indicating the extent of storage used.

Facilities are provided for the program to initiate the execution of a chain of I/O operations with a single START I/O or START I/O FAST RELEASE. When the chaining flags in the current CCW specify command chaining and no unusual conditions have been detected in the operation, the receipt of the device-end signal causes the channel to fetch a new

CCW and to initiate a new command at the device. A chained command is initiated in the same way as the first operation. Channel end and device end are not presented to the program when chaining causes another operation to follow. However, unusual situations can cause premature termination of command chaining and generation of an interruption condition.

Activities that cause I/O-interruption conditions are asynchronous to activity in the CPU, and more than one interruption condition can exist at the same time. The channel and the CPU establish priority among the conditions so that only one condition is presented to the CPU at a time. The conditions are preserved in the I/O devices or subchannels until accepted by the CPU.

The execution of an I/O operation or chain of operations thus involves up to four levels of participation:

1. Except for the effects caused by the integration of CPU and channel equipment, the CPU is busy for the duration of execution of START I/O or START I/O FAST RELEASE, which lasts at most until the addressed I/O device responds to the first command.
2. The subchannel is busy with the execution from the initiation of the operation at the I/O device until the interruption condition caused by the signal that terminates the last operation of the command chain is accepted by the CPU.
3. The control unit may remain busy after the subchannel has been released and may generate control-unit end when it becomes free.
4. The I/O device is busy from the initiation of the first operation until the interruption condition caused by the device end associated with the operation is accepted or cleared by the CPU.

An interruption condition caused by device end makes the device appear busy, but normally does not affect the state of any other part of the system. An interruption condition caused by control-unit end normally blocks communications through the control unit to any device attached to it, and an interruption condition caused by channel end normally blocks all communications through the subchannel.

### ***Compatibility of Operation***

The organization of the I/O system provides for a uniform method of controlling I/O operations. The capability of a channel, however, depends on its use and on the CPU model to which it is attached. Channels 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 that can accommodate at least this data rate.

The data rate a channel can accommodate depends also on the way the I/O operation is programmed. The channel can sustain its highest data rate when no data chaining is specified. Data chaining reduces the maximum allowable rate, and the extent of the reduction depends on the frequency at which new CCWs are fetched and on the address resolution of the first byte in each new storage area. Furthermore, since a channel shares storage with the CPU and other channels, activity in the rest of the system affects the accessibility of storage and, hence, the instantaneous load the channel can sustain.

In view of the dependence of channel capacity on programming and on activity in the rest of the system, an evaluation of the ability of elements in a specific I/O configuration to function concurrently must be based on a consideration of both the data rate and the way the I/O operations are programmed. Two systems employing identical complements of I/O devices may be able to execute certain programs in common, but it is possible that other programs requiring, for example, data chaining, may not run on one of the systems because of the increased load caused by the data chaining.

### **Control of Input/Output Devices**

The CPU controls I/O operations by means of eight I/O instructions: START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, HALT DEVICE, TEST CHANNEL, and STORE CHANNEL ID.

The instructions TEST CHANNEL and STORE CHANNEL ID address a channel; they do not address an I/O device. The other six I/O instructions address a channel and a device on that channel.

### ***Input/Output Device Addressing***

An I/O device and the associated access path are designated by an I/O address. The 16-bit I/O address consists of two parts: a channel address in the leftmost eight bit positions and a device address in the rightmost eight bit positions.

The channel address provides for identifying up to 256 channels. Channels are numbered 0-255. Channel 0 is a byte-multiplexer channel, and each of channels 1-255 may be either a multiplexer or a selector channel.

The number and type of channels and subchannels available, as well as their address assignment, depend on the system model and the particular installation.

The device address identifies the particular I/O device and control unit on the designated channel. The address identifies, for example, a particular magnetic-tape drive, disk-access mechanism, or transmission line. Any number in the range 0-255 can be used as a device address, providing facilities for addressing up to 256 devices per channel. An exception is some multiplexer channels that provide fewer than the maximum configuration of subchannels and hence eliminate the corresponding unassignable device addresses.

Devices that do not share a control unit with other devices may be assigned any device address in the range 0-255, provided the address is not recognized by any other control unit. Logically, such devices are not distinguishable from their control unit, and both are identified by the same address.

Devices sharing a control unit (for example, magnetic-tape drives or disk-access mechanisms) are assigned addresses within sets of contiguous numbers. The size of such a set is equal to the maximum number of devices that can share the control unit, or 16, whichever is smaller. Furthermore, such a set starts with an address in which the number of low-order zeros is at least equal to the number of bit positions required for specifying the set size. The high-order bit positions of an address within such a set identify the control unit, and the low-order bit positions designate the device on the control unit.

Control units designed to accommodate more than 16 devices may be assigned nonsequential sets of addresses, each set consisting of 16, or the number required to bring the total number of assigned addresses equal to the maximum number of devices attachable to the control unit, whichever is smaller. The addressing facilities are added in increments of a set so that the number of device addresses assigned to

a control unit does not exceed the number of devices attached by more than 15.

The control unit does not respond to any address outside its assigned set or sets. For example, if a control unit is designed to control devices having only bits 0000-1001 in the low-order positions of the device address, it does not recognize addresses containing 1010-1111 in these bit positions. On the other hand, a control unit responds to all addresses in the assigned set, regardless of whether the device associated with the address is installed. If no control unit responds to an address, the I/O device appears not operational. If a control unit responds to an address for which no device is installed, the absent device appears in the not-ready state.

Input/output devices accessible through more than one channel have a distinct address for each path of communications. This address identifies the channel and the control unit. For sets of devices connected to two or more control units, the portion of the address identifying the device on the control unit is fixed, and does not depend on the path of communications.

The assignment of channel and device addresses is arbitrary, subject to the rules described and any model-dependent restrictions. The assignment is made at the time of installation, and the addresses normally remain fixed thereafter.

### ***States of the Input/Output System***

The state of the I/O system identified by an I/O address depends on the collective state of the channel, subchannel, and I/O device. Each of these components of the I/O system can have up to four states, as far as the response to an I/O instruction is concerned. These states are listed in the figure "Input/Output System States." The name of the state is followed by its abbreviation and a brief definition.

A channel, subchannel, or I/O device that is available, interruption-pending, or working is called "operational." A channel, subchannel, or I/O device that is interruption-pending, working, or not-operational is called "not available."

In the case of a multiplexer channel, the channel and subchannel are easily distinguishable and, if the channel is operational, any combination of channel and subchannel states is possible. Since the selector channel can have only one subchannel, the channel and subchannel are functionally coupled, and certain states of the channel are related to those of the subchannel. In particular, the working state can occur only concurrently in both the channel and subchannel and, whenever an interruption condition is pending in the subchannel, the channel also is in the same state. The channel and subchannel, however, are not synonymous, and an interruption condition not associated with data transfer, such as attention, does not affect the state of the subchannel. Thus, the subchannel may be available when the channel has an interruption condition pending. Consistent distinction between the subchannel and channel permits selector and multiplexer channels to be covered uniformly by a single description.

<b>Name</b>	<b>Abbreviation and Definition</b>	
<b>Channel</b>		
Available	A	None of the following states
Interruption pending	I	Interruption condition immediately available from channel
Working	W	Channel operating in burst mode
Not operational	N	Channel not operational

<b>Subchannel</b>		
Available	A	None of the following states
Interruption pending	I	Information for CSW available in subchannel
Working	W	Subchannel executing an operation
Not operational	N	Subchannel not operational

<b>I/O Device</b>		
Available	A	None of the following states
Interruption pending	I	Interruption condition in device
Working	W	Device executing an operation
Not operational	N	Device not operational

**Input/Output-System States**

The device referred to in the figure "Input/Output-System States" includes both the device proper and its control unit. For some types of devices, such as magnetic-tape units, the working and the interruption-pending states can be caused by activity in the addressed device or control unit. A "not available" shared control unit imposes its state on all devices attached to the control unit. The states of the devices are not related to those of the channel and subchannel.

When the response to an I/O instruction is determined by the state of the channel or subchannel, the components further removed are not interrogated. Thus, 10 composite states may be distinguished as conditions for the execution of I/O instructions. Each composite state is identified by three letters. The first letter specifies the state of the channel, the second letter specifies the state of the subchannel, and the third letter specifies the state of the device. Each letter may be A, I, W, or N, denoting the state of the component. The letter X indicates that the state of the corresponding component is not significant for the execution of the instruction.

**Available (AAA):** The addressed channel, subchannel, control unit, and I/O device are operational, are not engaged in the execution of any previously initiated operations, and do not contain any pending interruption conditions.

Because of internal activity, some block-multiplexer channels may at times appear to be working even though they are not engaged in the execution of a previously initiated operation and do not contain any interruption condition. This will result in a WXX state instead of the AAA state.

**Interruption Pending in Device (AAI) or Device Working (AAW):** The addressed channel and subchannel are available. The addressed control unit or I/O device is executing a previously initiated operation or contains an interruption condition. These situations are possible:

1. The device is executing an operation, such as rewinding magnetic tape or seeking on a disk file, after signaling channel end.
2. The control unit associated with the device is executing an operation, such as backspacing file on a magnetic-tape unit, after signaling channel end.
3. The device or control unit is executing an operation on another subchannel or channel.

4. The device or control unit contains the device-end, control-unit-end, or attention condition or a channel-end condition associated with a terminated operation.

**Device Not Operational (AAN):** The addressed channel and subchannel are available. The addressed I/O device is not operational. A device appears not operational when no control unit recognizes the address. 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 disconnected from the system. The not-operational state is indicated also when the control unit is provided and is designed to attach the device, but the device has not been installed and the address has not been assigned to the control unit. (See also the section "Input/Output Device Addressing" in this chapter.)

If the addressed device is not installed or has been logically removed from the control unit, but the associated control unit is operational and the address has been assigned to the control unit, the device is said to be not ready. When an instruction is addressed to a device in the not-ready state, the control unit responds to the selection and indicates unit check whenever the not-ready state precludes a successful execution of the operation. (See the section "Unit Check" in this chapter.)

**Interruption Pending in Subchannel (AIX):** The addressed channel is available. An interruption condition is pending in the addressed subchannel. The subchannel is able to provide information for a CSW. The interruption condition can indicate concluding of an operation at the addressed I/O device or at another device on the subchannel. The state of the addressed device is not significant, except when TEST I/O is addressed to the device associated with the interruption condition, in which case the CSW contains status information provided by the device.

The state AIX does not occur on the selector channel. On the selector channel, the existence of an interruption condition in the subchannel immediately causes the channel to assign to this condition the highest priority for I/O interruptions and, hence, leads to the state IIX.

**Subchannel Working (AWX):** The addressed channel is available. The addressed subchannel is executing a previously initiated operation or chain of operations

and has not yet received channel end for the last operation. The state of the addressed device is not significant, except when HALT I/O or HALT DEVICE is issued. During HALT I/O and HALT DEVICE, the state of the device may be interrogated and will then be indicated in either the CSW or the condition code.

The subchannel-working state does not occur on the selector channel since all operations on the selector channel are executed in the burst mode and cause the channel to be in the working state (WWX).

**Subchannel Not Operational (ANX):** The addressed channel is available. The addressed subchannel on the multiplexer channel is not operational. A subchannel is not operational when it is not provided in the system. This state cannot occur on the selector channel.

**Interruption Pending in Channel (IXX):** The addressed channel is not working and has established which device will cause the next I/O interruption from this channel. The state where the channel contains an interruption condition is distinguished only by the instruction TEST CHANNEL. This instruction does not cause the subchannel and I/O device to be interrogated. The other I/O instructions, with the exception of STORE CHANNEL ID, consider the channel available when it contains an interruption condition. A channel with an interruption condition may be considered to be working by the instruction STORE CHANNEL ID. When the channel assigns priority for interruptions among devices, the interruption condition is preserved in the I/O device or subchannel. (See the section "Interruption Conditions" in this chapter.)

**Channel Working (WXX):** The addressed channel is operating in the burst mode. In the case of the multiplexer channel, a burst of bytes is currently being handled. In the case of the selector channel, an operation or a chain of operations is currently being executed, and the channel end for the last operation has not yet been signaled. The states of the addressed device and, in the case of the multiplexer channel, of the subchannel are not significant. In addition, because of internal activity, some block-multiplexer channels may at times appear to be working even though they are not operating in burst mode. Depending on the model and the channel type, TEST I/O and HALT DEVICE may consider the channel to be available when the

channel is working with a device other than the addressed device.

**Channel Not Operational (NXX):** The addressed channel is not operational. A channel is not operational when it is not provided in the system, when power is off in the channel, or when it is not configured to the CPU. The states of the addressed I/O device and subchannel are not significant.

### **Resetting of the Input/Output System**

Two types of resetting can occur in the I/O system: an I/O system reset and an I/O selective reset. The response of each type of I/O device to the two kinds of reset is specified in the SL publication for the device.

#### **I/O System Reset**

I/O system reset is performed when the CPU performs a program reset, initial-program reset, clear reset, or power-on reset, and when a power-on sequence is performed by the channel.

I/O system reset causes the channel to conclude operations on all subchannels. Status information and all interruption conditions in all subchannels are reset, and all operational subchannels are placed in the available state. The channel signals system reset to all I/O devices attached to it.

#### **I/O Selective Reset**

The I/O selective reset is performed by some channels when they detect certain equipment malfunctions.

I/O selective reset causes the channel to signal selective reset to the device that is connected to the channel at the time the malfunction is detected. No subchannels are reset.

#### **Effect of Reset on a Working Device**

With either type of reset, if the device is currently communicating with a channel, the device immediately disconnects from the channel. 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 appears in the working state until the termination of 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 completing any mechanical operation.

#### **Reset Upon Malfunction**

When a malfunction occurs and the program is alerted by an I/O interruption, or when a malfunction occurs during the execution of an I/O instruction and the program is alerted by the setting of a condition code, then an I/O selective reset may have been performed. A CSW is stored identifying the cause of the malfunction.

The device addressed by the I/O instruction is not necessarily the device that is reset.

When a malfunction occurs and the program is alerted by a machine-check interruption, then an I/O selective reset may have been performed. This may or may not be accompanied by an I/O interruption. When no I/O interruption occurs, a CSW is not stored and a device is not identified.

#### **Condition Code**

The results of certain tests by the channel and device, and the original state of the addressed part of the I/O system are used during the execution of an I/O instruction to set one of four condition codes in the PSW. The condition code is set at the time the execution of the instruction is concluded, that is, the time the CPU is released to proceed with the next instruction. The condition code ordinarily indicates whether or not the function specified by the instruction has been performed and, if not, the reason for the rejection. In the case of START I/O FAST RELEASE executed independent of the device, a condition code 0 may be set that is later superseded by a deferred condition code stored in the CSW.

The figure "Condition-Code Settings for I/O States and Instructions" lists the I/O-system status and the corresponding condition codes for each I/O instruction. The I/O-system states and associated abbreviations were defined in the section "States of the Input/Output System" earlier in this chapter. The digits in the figure represent the decimal value of the code. The instructions START I/O and START I/O FAST RELEASE can set condition code 0 or 1 for the AAA state, depending on the type of operation initiated. Equipment malfunctions and programming errors generally cause condition code 1 to be set and the CSW to be stored.

I/O-System States	I/O State	Condition-Code Settings						
		SIO SIOF	TIO	CLRIO	HIO	HDV	TCH	STDC
Available	AAA	0,1* <sup>@</sup>	0	0	1*	1*	0	0
Interruption pending in device	AAI	1* <sup>@</sup>	1*	0	1*	1*	0	0
Device working	AAW	1* <sup>@</sup>	1*	0	1*	1*	0	0
Device not operational	AAN	3 <sup>@</sup>	3	0	3	3	0	0
Interruption pending in subchannel	AIX							
For the addressed device		2	1*	1*	0	0	0	0
For another device		2	2	0	0	0	0	0
Subchannel working	AWX							
With the addressed device		2	2	1*	1*#	1*#	0	0
With another device		2	2	0	1*#	0	0	0
Subchannel not operational	ANX	3	3	3	3	3	0	0
Interruption pending in channel	IXX		See Note				1	##
Channel working	WXX							
With the addressed device		2	2	***	2	+	2	##
With another device		2	2●	**	2	≠	2	##
Internal activity		2	2●	**	2	≠	2	##
Channel not operational	NXX	3	3	3	3	3	3	3

**Explanation:**

\* Whenever condition code 1 is set, the CSW or its status portion is stored at location 64 during execution of the instruction.

\*\* When CLEAR I/O encounters the WXX state, either condition code 2 is set, or the channel is treated as available and the condition code is set according to the state of the subchannel. When the channel is treated as available, the condition codes for the WXX states are the same as for the AXX states.

\*\*\*Condition code 1 (with the CSW stored) or 2 may be set, depending on the channel.

≠ The condition code depends on the state of the subchannel, the channel type, and the system model. If the subchannel is not operational, a condition code 2 or 3 is set. If the subchannel is available or working with the addressed device, condition code 2 is set. Otherwise, a condition code 0 or 2 is set.

# When a "device not operational" response is received in selecting the addressed device, condition code 3 is set.

@ START I/O FAST RELEASE may cause the same condition code to be set as for START I/O or may cause condition code 0 to be set.

+ If the channel ascertains that the device received the signal to terminate, condition code 1 is set and the CSW stored. Otherwise, condition code 2 is set.

## When the channel is unable to store the channel ID because of the working or interruption-pending state, condition code 2 is set. If the working or interruption-pending state does not preclude storing the channel ID, condition code 0 is set.

● If the subchannel is interruption pending for the addressed device, condition code 1 may be set depending on the channel type.

**Note:** For the purpose of executing START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT DEVICE, and HALT I/O, a channel containing an interruption condition appears the same as an available channel, and the condition-code setting depends on the states of the subchannel and device. The condition codes for the IXX states are the same as for the AXX states, where the Xs represent the states of the subchannel and the device. As an example, the condition code for the IAW state is the same as for AAW.

**Condition-Code Settings for I/O States and Instructions**



## Instructions

All I/O instructions cause a serialization function to be performed. See the section "Serialization" in Chapter 5, "Program Execution."

The names, mnemonics, format, and operation codes of the I/O instructions are listed in the figure "Input/Output Instructions." The figure also indicates that all I/O instructions cause a program interruption when they are encountered in the

problem state, and that all I/O instructions set the condition code.

*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. In the case of START I/O, for example, SIO is the mnemonic and D<sub>2</sub>(B<sub>2</sub>) the operand designation.*

Name	Mnemonic	Characteristics				Code
CLEAR I/O	CLRIO	S	C	M	\$	9D01*
HALT DEVICE	HDV	S	C	M	\$	9E01*
HALT I/O	HIO	S	C	M	\$	9E00*
START I/O	SIO	S	C	M	\$	9C00*
START I/O FAST RELEASE	SIOF	S	C	M	\$	9C01*
STORE CHANNEL ID	STIDC	S	C	M	\$	B203
TEST CHANNEL	TCH	S	C	M	\$	9F00‡
TEST I/O	TIO	S	C	M	\$	9000*

### Explanation:

C Condition code is set.

M Privileged-operation exception.

S S instruction format.

\* Bits 8-14 of the operation code are ignored.

‡ Bits 8-15 of the operation code are ignored.

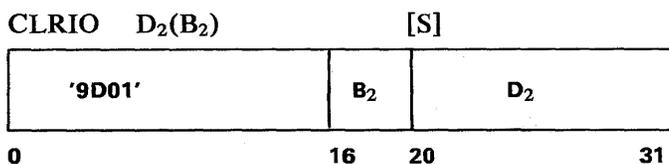
§ Causes serialization.

### Input/Output Instructions

### Programming Note

The instructions START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, HALT DEVICE, and STORE CHANNEL ID cause a CSW to be stored. To prevent the contents of the CSW stored by the instruction from being destroyed by an immediately following I/O interruption, the CPU must be disabled for all I/O interruptions before START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, HALT DEVICE, or STORE CHANNEL ID is issued and must remain disabled until the information in the CSW provided by the instruction has been acted upon or stored elsewhere for later use.

### CLEAR I/O



Either a TIO or CLRIO function is performed, depending on the block-multiplexing control, bit 0 of control register 0. The TIO function is performed when the block-multiplexing-control bit is zero.

The TIO function is described in the definition of the instruction TEST I/O.

Bits 8-14 of the instruction are ignored. Bit positions 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies.

The CLRIO function causes the current operation with the addressed device to be discontinued and the state of the operation at the time of the discontinuation to be indicated in the stored CSW.

When the subchannel is available, interruption-pending with another device, or working with another device, no channel action is taken, and condition code 0 is set. Channels not capable of determining subchannel states while in the working state may instead set condition code 2.

When the subchannel is either working with the addressed device or interruption-pending with the addressed device, the CLRIO function causes the channel to discontinue the operation with the addressed device by storing the status of the operation in the CSW and making the subchannel available. When the channel is working with the addressed device, the device is signaled to terminate

the current operation. Some channels may, instead, indicate busy and cause no channel action.

When any of the following conditions occurs, the CLRIO function causes the CSW at location 64 to be stored. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

1. The channel is available or interruption-pending, and the subchannel contains an interruption condition for the addressed device or is working with the addressed device. The subchannel-key, command-address, and count fields describe the state of the operation at the time of the execution of the instruction.
2. The channel is working with the addressed device. The subchannel-key, command-address, and count fields describe the state of the operation at the time the instruction is executed. (Some channels alternatively indicate busy under this condition.)
3. The channel is working with a device other than the one addressed, and the subchannel contains an interruption-pending condition for the addressed device or is working with the addressed device. The subchannel-key, command-address, and count fields describe the state of the operation at the time CLEAR I/O is executed. (Some channels alternatively indicate busy under these conditions.)
4. The channel detected an equipment error during the execution of the instruction. The CSW identifies the error condition. The states of the channel and the I/O operations in progress are unpredictable. The limited channel logout, if stored, indicates a sequence code of 000.

When CLEAR I/O cannot be executed because of a pending logout that affects the operational capability of the channel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending and channel-control-check bits, which are set to ones. No channel logout is associated with this status.

### Program Exceptions:

Privileged Operation

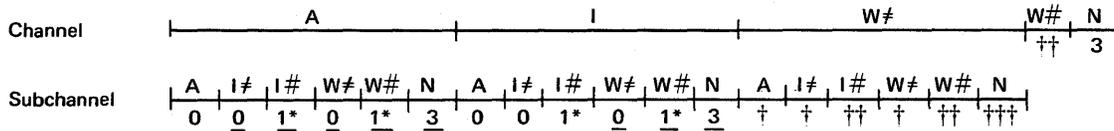
### Resulting Condition Code:

- 0 No operation in progress for the addressed device
- 1 CSW stored
- 2 Channel busy
- 3 Not operational

The condition code set by the CLRIO function for all possible states of the I/O system is shown in the

figure "Condition Codes Set By CLEAR I/O." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states. The condition code set when CLEAR

I/O causes the TIO function to be performed is shown in the figure "Condition Codes Set By TEST I/O" in the definition of the instruction TEST I/O.



- A Available
  - I Interruption pending
    - I# = Interruption pending for a device other than the one addressed
    - I# = Interruption pending for the addressed device
  - W Working
    - W# = Working with a device other than the one addressed
    - W# = Working with the addressed device
  - N Not operational
  - \* CSW stored
- † In the W#AX, W#I#X, and W#W#X states, a condition code 0 or 2 may be set, depending on the channel.
- †† In the W#I#X, W#W#X, and W#XX states, a condition code 1 (with the CSW stored) or 2 may be set, depending on the channel.
- ††† In the W#NX state, a condition code 2 or 3 may be set, depending on the channel.

**Note:** Underscored codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set By Clear I/O

## Programming Notes

1. Since some channels cause a condition code 2 to be set when the instruction is received and the channel is working, it may be useful to issue a halt instruction and then CLEAR I/O to the desired address. Using HALT DEVICE will ensure that condition code 2 is received on the CLEAR I/O only when the channel is working with a device other than the one addressed. Using HALT I/O will ensure that the current working state, if any, is terminated without regard for the address.
2. Because of the inability of CLEAR I/O to terminate operations on some channels when in the working state, the instruction is not a suitable substitute for HALT I/O or HALT DEVICE.
3. The combination of HALT DEVICE followed by CLEAR I/O can be used to clear out all activity on a channel by executing the two instructions for all device addresses on the channel.

## HALT DEVICE

HDV	D <sub>2</sub> (B <sub>2</sub> )	[S]
'9E01'	B <sub>2</sub>	D <sub>2</sub>
0	16	20
		31

The current I/O operation at the addressed I/O device is terminated. The subsequent state of the subchannel depends on the type of channel. Bits 8-14 of the instruction are ignored.

Bits 16-31 of the second-operand address identify the channel, the subchannel, and the I/O device to which the instruction applies.

When the channel is either available or interruption-pending with the subchannel available or working with the addressed device, HALT DEVICE causes the addressed device to be selected and to be signaled to terminate the current operation, if any. If the subchannel is working with the addressed device, HALT DEVICE also causes the subchannel to signal termination of the device operation the next time the device requests or offers a byte of data, if any. If chaining is indicated for the I/O operation using the subchannel, it is suppressed. If the subchannel is available, the subchannel is not affected.

When the channel is either available or interruption-pending with the subchannel either working with a device other than the one addressed or interruption-pending, no action is taken.

When the channel is working in burst mode with the addressed device, data transfer for the operation is immediately terminated, and the device immediately disconnects from the channel. If chaining is indicated for the I/O operation using the subchannel, it is suppressed.

When the channel is working in burst mode with a device other than the one addressed, and the subchannel is available, interruption-pending, or working with a device other than the one addressed, no action is taken. If the subchannel is working with the addressed device, the subchannel signals termination of the device operation the next time the device requests or offers a byte of data, if any. If chaining is indicated for the I/O operation using the subchannel, it is suppressed.

When the channel is working in burst mode with a device other than the one addressed and the subchannel is not operational, is interruption-pending, or is working with a device other than the one addressed, the resulting condition code may, in some channels, be determined by the subchannel state.

Termination of a burst operation by HALT DEVICE on a selector channel causes the channel and subchannel to be placed in the interruption-pending state. Generation of the interruption condition is not contingent on the receipt of status information from the device. When HALT DEVICE causes a burst operation on a byte-multiplexer channel to be terminated, the subchannel associated with the burst operation remains in the working state until the device provides ending status, whereupon the subchannel enters the interruption-pending state. The termination of a burst operation by HALT DEVICE on a block-multiplexer channel may, depending on the model and the type of subchannel, take place as for a selector channel or may allow the subchannel to remain in the working state until the device provides ending status.

When any of the three situations numbered below occurs, HALT DEVICE causes the 16-bit unit-status and channel-status portion of the CSW to be replaced by a new set of status bits. The contents of the other fields of the CSW are not changed. The CSW stored by HALT DEVICE pertains only to the execution of HALT DEVICE and does not describe the I/O operation, at the addressed subchannel, that is terminated. The extent of data transfer, and the status at the termination of the operation at the subchannel, are provided in the CSW associated with the interruption condition caused by the termination. The three situations are:

1. The addressed device is selected and signaled to terminate the current operation, if any. The CSW then contains zeros in the status field unless a machine malfunction is detected.
2. The control unit is busy and the device cannot be given the signal to terminate the operation. The CSW unit-status field contains ones in the busy and status-modifier bit positions. The channel-status field contains zeros unless a machine malfunction is detected.
3. The channel detects a machine malfunction during the execution of HALT DEVICE. The status bits in the CSW then identify the type of malfunction. The state of the channel and the progress of the I/O operation are unpredictable.

When HALT DEVICE cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

When HALT DEVICE causes data transfer to be terminated, the control unit associated with the operation remains not available until the data-handling portion of the operation in the control unit is concluded. Conclusion of this portion of the operation is signaled by the generation of channel end. This may occur at the normal time for the operation, or earlier, or later, depending on the operation and type of device. If the control unit is shared, all devices attached to the control unit appear in the working state on that channel until the channel-end condition is accepted by the CPU. The I/O device executing the terminated operation remains in the working state until the end of the

inherent cycle of the operation, at which time device end is generated. If blocks of data at the device are defined, as in read-type operations on magnetic tape, the recording medium is advanced to the beginning of the next block.

When HALT DEVICE is issued at a time when the subchannel is available and no burst operation is in progress, the effect of the HALT DEVICE signal depends partially on the type of device and its state. In all cases, the HALT DEVICE signal has no effect on devices that are not in the working state or are executing a mechanical operation in which data is not transferred, such as rewinding tape or positioning a disk-access mechanism. If the device is executing a type of operation that is unpredictable in duration, or in which data is transferred, the device interprets the signal as one to terminate the operation. Pending attention or device-end conditions at the device are not reset.

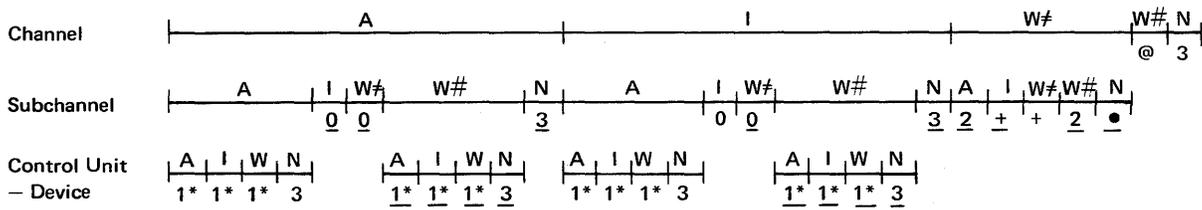
***Program Exceptions:***

Privileged Operation

***Resulting Condition Code:***

- 0 Subchannel busy with another device or interruption pending
- 1 CSW stored
- 2 Channel working
- 3 Not operational

The condition code set by HALT DEVICE for all possible states of the I/O system is shown in the figure "Condition Codes Set by HALT DEVICE." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.



- A Available
- I Interruption pending
- W Working
- W# = Working with a device other than the one addressed
- W# = Working with the addressed device
- N Not operational
- \* CSW stored
- @ In the W# XX state, either condition code 1 (with CSW stored) or condition code 2 may be set, depending on the channel. However, condition code 1 (with CSW stored) can be set only if the control unit has received the signal to terminate or if control-unit-busy status is received by the channel.
- + In the W#IX and W#W#X states, either condition code 0 or 2 may be set.
- In the W#NX state, either condition code 2 or 3 may be set, depending on the model and the channel type.

**Note:** Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

**Condition Codes Set By HALT DEVICE**

### Programming Note

The execution of HALT DEVICE always causes data transfer between the addressed device and the channel to be terminated. The condition code and the CSW (when stored) indicate whether the control unit was signaled to terminate its operation during the execution of the instruction. If the control unit was not signaled to terminate its operation, the condition code and the CSW (when stored) imply the situations under which the execution of a HALT DEVICE for the same address will cause the control unit to be signaled to terminate.

*Condition Code 0* indicates that HALT DEVICE cannot signal the control unit until an interruption condition on the same subchannel is cleared.

*Condition Code 1 with Control-Unit-Busy Status in the CSW* indicates that HALT DEVICE cannot signal the control unit until the control-unit-end status is received from that control unit.

*Condition Code 1 with Zeros in the Status Field of the CSW* indicates that the addressed device was selected and signaled to terminate the current operation, if any.

*Condition Code 2* indicates that the control unit cannot be signaled until the channel is not working. The end of the working state can be detected by noting an interruption from the channel or by noting the results of repeatedly executing HALT DEVICE.

*Condition Code 3* indicates that manual intervention is required in order to allow HALT DEVICE to signal the control unit to terminate.

### HALT I/O

HIO	D <sub>2</sub> (B <sub>2</sub> )	[S]
'9E00'	B <sub>2</sub>	D <sub>2</sub>
0	16	20
		31

Execution of the current I/O operation at the addressed I/O device, subchannel, or channel is terminated. The subsequent state of the subchannel depends on the type of channel. Bits 8-14 of the instruction are ignored.

Bits 16-31 of the second-operand address identify the channel and, when the channel is not working, identify the subchannel and the I/O device to which the instruction applies.

When the channel is either available or interruption-pending, with the subchannel either available or working, HALT I/O causes the

addressed device to be selected and to be signaled to terminate the current operation, if any. If the subchannel is available, its state is not affected. If, on the byte-multiplexer channel, the subchannel is working, data transfer is immediately terminated, but the subchannel remains in the working state until the device provides the next status byte, whereupon the subchannel is placed in the interruption-pending state.

When HALT I/O is issued to a channel operating in the burst mode, data transfer for the burst operation is terminated, and the device performing the burst operation is immediately disconnected from the channel. The subchannel and I/O-device address in the instruction, in this case, is ignored.

The termination of a burst operation by HALT I/O on the selector channel causes the channel and subchannel to be placed in the interruption-pending state. Generation of the interruption condition is not contingent on the receipt of a status byte from the device. When HALT I/O causes a burst operation on the byte-multiplexer channel to be terminated, the subchannel associated with the burst operation remains in the working state until the device signals channel end, whereupon the subchannel enters the interruption-pending state. The termination of a burst operation by HALT I/O on a block-multiplexer channel may, depending on the model and the type of subchannel, take place as for a selector channel or may allow the subchannel to remain in the working state until the device provides ending status.

On the byte-multiplexer channel operating in the byte-multiplex mode, the device is selected and the instruction executed only after the channel has serviced all outstanding requests for data transfer for previously initiated operations, including the operation to be halted. If the control unit does not accept the HALT I/O signal because it is in the not-operational or control-unit-busy state, the subchannel, if working, is set up to signal termination of device operation the next time the device requests or offers a byte of data. If command chaining is indicated in the subchannel and the device presents status next, chaining is suppressed.

When the addressed subchannel is interruption-pending, with the channel available or interruption-pending, HALT I/O does not cause any action.

When any of the following conditions occurs, HALT I/O causes the status portion, bits 32-47, of the CSW to be replaced by a new set of status bits. The contents of the other fields of the CSW are not changed. The CSW stored by HALT I/O pertains

only to the execution of HALT I/O and does not describe the I/O operation, at the addressed subchannel, that is terminated. The extent of data transfer, and the status at the termination of the operation at the subchannel, are provided in the CSW associated with the interruption condition due to the termination.

1. The addressed device was selected and signaled to terminate the current operation. The CSW contains zeros in the status field unless an equipment error is detected.
2. The channel attempted to select the addressed device, but the control unit could not accept the HALT I/O signal because it is executing a previously initiated operation or had an interruption condition associated with a device other than the one addressed. The signal to terminate the operation has not been transmitted to the device, and the subchannel, if in the working state, will signal termination the next time the device identifies itself. The CSW unit-status field contains ones in the busy and status-modifier bit positions. The channel-status field contains zeros unless an equipment error is detected.
3. The channel detected an equipment malfunction during the execution of HALT I/O. The status bits in the CSW identify the error condition. The state of the channel and the progress of the I/O operation are unpredictable.

When HALT I/O cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

When HALT I/O causes data transfer to be terminated, the control unit associated with the operation remains unavailable until the data-handling portion of the operation in the control unit is terminated. Termination of the data-transfer portion

of the operation is signaled by the generation of channel end, which may occur at the normal time for the operation, earlier, or later, depending on the operation and type of device. If the control unit is shared, all devices attached to the control unit appear in the working state until the channel-end signal is accepted by the CPU. The I/O device executing the terminated operation remains in the working state until the end of the inherent cycle of the operation, at which time device end is generated. If blocks of data at the device are defined, such as reading on magnetic tape, the recording medium is advanced to the beginning of the next block.

When HALT I/O is issued at a time when the subchannel is available and no burst operation is in progress, the effect of the HALT I/O signal depends on the type of device and its state and is specified in the SL publication for the device. The HALT I/O signal has no effect on devices that are not in the working state or are executing a mechanical operation in which data is not transferred, such as rewinding tape or positioning a disk-access mechanism. If the device is executing a type of operation that is variable in duration, the device interprets the signal as one to terminate the operation. Attention or device-end signals at the device are not reset.

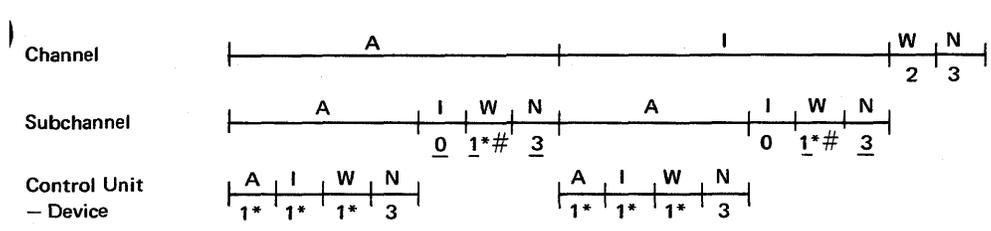
***Program Exceptions:***

Privileged Operation

***Resulting Condition Code:***

- 0 Interruption pending in subchannel
- 1 CSW stored
- 2 Burst operation terminated
- 3 Not operational

The condition code set by HALT I/O for all possible states of the I/O system is shown in the figure "Condition Codes Set By HALT I/O." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.



- A Available
- I Interruption pending
- W Working
- N Not operational
- \* CSW stored
- # When a device-not-operational response is received in selecting the addressed device, a condition code 3 is set.

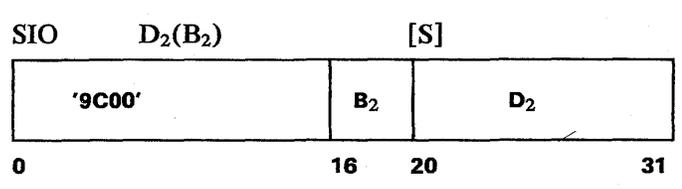
**Note:** Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by HALT I/O

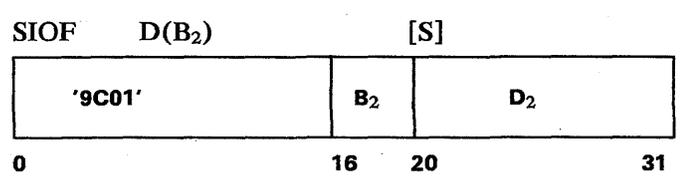
**Programming Note**

The instruction HALT I/O provides the program with a means of terminating an I/O operation before all data specified in the operation has been transferred or before the operation at the device has reached its normal ending point. It permits the program to immediately free the selector channel for an operation of higher priority. On the byte-multiplexer channel, HALT I/O provides a means of controlling real-time operations and permits the program to terminate data transmission on a communication line.

**START I/O**



**START I/O FAST RELEASE**



A write, read, read backward, control, or sense operation is initiated with the addressed I/O device and subchannel. Bits 8-14 of the instruction are ignored.

Either an SIO or SIOF function is performed, depending on the instruction, the channel, and the block-multiplexing control, bit 0 of control register 0. The instruction START I/O always causes the SIO function to be performed, as does START I/O FAST RELEASE when the block-multiplexing-control bit is zero. When the bit is one, START I/O FAST RELEASE may, depending on the channel, cause either the SIO or the SIOF function to be performed.

Bits 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies. The CAW, at location 72, contains the subchannel key and the address of the first CCW. This CCW specifies the operation to be performed, the storage area to be used, and the action to be taken when the operation is completed.

For the SIO function, the I/O operation is initiated if the addressed I/O device and subchannel are available, the channel is available or interruption-pending, and errors or exceptional situations have not been detected. The I/O operation is not initiated when the addressed part of the I/O system is in any other state or when the channel or device detects any error or exceptional situations during execution of the instruction.

For the SIOF function, the I/O operation is initiated if the subchannel is available, the channel is available or interruption-pending, and errors or exceptional situations have not been detected. The I/O operation is not initiated when the subchannel and channel are in any other state or when the channel or device detects any error or exceptional situation during execution of the instruction. The

device state or device-detected errors are not relevant during instruction execution but are indicated in a CSW stored during a subsequent interruption.

When the channel is available or interruption-pending, and the subchannel is available before the execution of the instruction, the following situations cause a CSW to be stored. How the CSW is stored depends on whether an SIO or SIOF function is performed. The SIO function causes the status portion of the CSW to be replaced by a new set of status bits. The status bits pertain to the device addressed by the instruction. The contents of the other fields of the CSW are not changed. When the SIOF function is performed, the first situation causes the same action as for the SIO function; also, the control-unit state may be tested, and so situation 5 may cause the same action as for the SIO function, or the situation may be indicated in a subsequent interruption during which the entire CSW will be stored. The remaining situations for the SIOF function will be indicated in a subsequent interruption, during which the entire CSW will be stored.

1. The channel detects a programming error in the contents of the CAW or detects an equipment error during execution of the instruction. The CSW identifies the error. The channel-end and busy bits are zeros, unless, for the SIO function, the error was detected after the device was selected, and the device was found to be busy, in which case the busy bit, as well as any bits indicating interruption conditions, are ones. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation is not initiated. No interruption conditions are generated at the I/O device or subchannel. The state of the PCI bit in the CSW is unpredictable.
2. The channel detects a programming error associated with the first CCW or, for the SIOF function, the channel detects an equipment error after completion of the instruction. The CSW identifies the error. The channel-end and busy bits are zeros, unless the error was detected after the device was selected, and the device was found to be busy, in which case the busy bit, as well as any bits indicating interruption conditions, are ones. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation is not initiated. No interruption conditions are generated at the I/O device or subchannel. The state of the PCI bit in the CSW is unpredictable.
3. An immediate operation was executed, and either (a) no command chaining is specified and no command retry occurs, or (b) chaining is suppressed because of unusual situations detected during the operation. In the CSW, the channel-end bit is one, the busy bit is zero, and other status may be indicated. The PCI bit is one if PCI was specified in the first CCW. The I/O operation is initiated, but no information has been transferred to or from the storage area designated by the CCW. No interruption conditions are generated at the subchannel, and the subchannel is available for a new I/O operation. If device end is not indicated, the device remains busy, and a subsequent device-end condition is generated.
4. The I/O device is interruption-pending, or the control unit is interruption-pending for the addressed device. The CSW unit-status field contains one in the busy-bit position, identifies the interruption condition, and may contain other bits provided by the device or control unit. The interruption condition is cleared. The I/O operation is not initiated. The channel-status field indicates any errors detected by the channel, and the PCI bit is one if PCI was specified in the first CCW.
5. The I/O device or the control unit is executing a previously initiated operation, or the control unit is interruption-pending for a device other than the one addressed. The CSW unit-status field contains one in the busy-bit position or, if the control unit is busy, the busy and status-modifier bits are ones. The I/O operation is not initiated. The channel-status field indicates any errors detected by the channel, and the PCI bit is one if specified in the first CCW.
6. The I/O device or control unit detected an equipment or programming error during the initiation, or the addressed device is not ready. The CSW identifies the error. The channel-end and busy bits are zeros, unless the device was busy, in which case the busy bit, as well as any bits causing interruption conditions, are ones. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation is not initiated. No interruption conditions are generated at the I/O device or subchannel. The PCI bit in the CSW is one if PCI was specified in the first CCW.

When the SIO or SIOF function cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

When the SIOF function causes condition code 0 to be set and subsequently a situation is encountered which would have caused a condition code 1 to be set had the function been SIO, a deferred-condition-code-1 I/O-interruption condition is generated.

When the SIOF function causes condition code 0 to be set and, subsequently, it is determined that the device is not operational, a deferred-condition-code-3 I/O-interruption condition is generated. In both of the above cases, in the resulting I/O interruption, a full CSW is stored, and the deferred condition code appears in the CSW.

On the byte-multiplexer channel, both the SIO and SIOF functions cause the addressed device to be

selected and the operation to be initiated only after the channel has serviced all outstanding requests for data transfer for previously initiated operations.

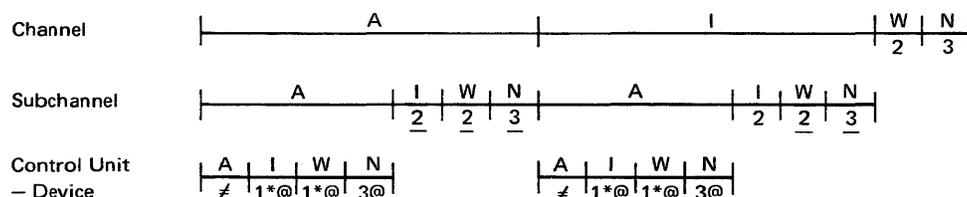
**Program Exceptions:**

Privileged Operation

**Resulting Condition Code:**

- 0 I/O operation initiated and channel proceeding with its execution
- 1 CSW stored
- 2 Channel or subchannel busy
- 3 Not operational

The condition code set by START I/O and START I/O FAST RELEASE for all possible states of the I/O system is shown in the figure "Condition Codes Set By START I/O and START I/O FAST RELEASE." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.



- A Available
- I Interruption pending
- W Working
- N Not operational
- \* CSW stored
- @ The SIOF function may cause condition code 0 to be set, in which case the other condition code shown will be specified as a deferred condition code.

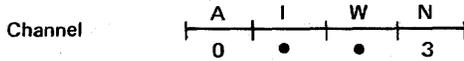
- ≠ ● When a nonimmediate I/O operation has been initiated, and the channel is proceeding with its execution, condition code 0 is set.
- When an immediate operation has been initiated, and no command chaining or command retry is taking place, or the device is not ready, or an error has been detected by the control unit or device, for the SIO function condition code 1 is set, and the CSW is stored. Under the same circumstances, for the SIOF function condition code 0 is set, and a deferred-condition-code 1 I/O-interruption condition is generated.

**Note:** Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

**Condition Codes Set by START I/O and START I/O FAST RELEASE**



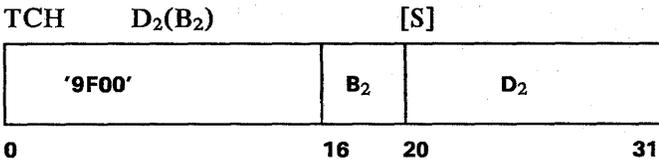
Input/Output System" for a detailed definition of the A, I, W, and N states.



- A Available
- I Interruption pending
- W Working
- N Not operational
- When the channel is unable to store the channel ID because of its working state or because it contains a pending interruption condition, a condition code 2 is set. If the working or interruption pending state does not preclude the storing of the channel ID, a condition code 0 is set.

Condition Codes Set by STORE CHANNEL ID

**TEST CHANNEL**



The condition code in the PSW is set to indicate the state of the addressed channel. The state of the channel is not affected, and no action is caused. Bits 8-15 of the instruction are ignored.

Bits 16-23 of the second-operand address identify the channel to which the instruction applies. Bit positions 24-31 of the address are ignored.

The instruction TEST CHANNEL inspects only the state of the addressed channel. It tests whether the channel is operating in the burst mode, is interruption-pending, or is not operational. When the channel is operating in the burst mode and contains an interruption condition, the condition code is set as for operation in the burst mode. When none of these situations exist, the available state is indicated. No device is selected, and, on the multiplexer channel, the subchannels are not interrogated.

**Program Exceptions:**

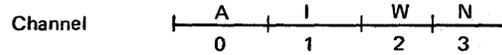
Privileged Operation

**Resulting Condition Code:**

- 0 Channel available
- 1 Interruption or logout condition in channel
- 2 Channel operating in burst mode
- 3 Channel not operational

The condition code set by TEST CHANNEL for all possible states of the addressed channel is shown in

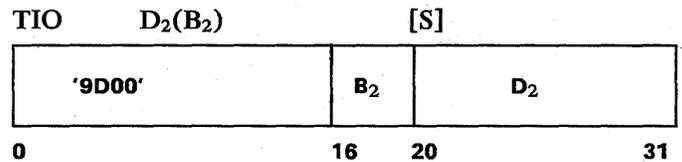
the figure "Condition Codes Set by TEST CHANNEL." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.



- A Available
- I Interruption pending
- W Working
- N Not operational

Condition Codes Set by TEST CHANNEL

**TEST I/O**



The state of the addressed channel, subchannel, and device is indicated by setting the condition code in the PSW and, in certain situations, by storing the CSW. Interruption conditions may be cleared. Bits 8-14 of the instruction are ignored.

Bits 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies.

The TIO function is performed by the instruction TEST I/O and, under certain circumstances, by CLEAR I/O.

When the channel is operating in burst mode and the addressed subchannel contains an interruption condition, the TIO function causes condition code 1 or 2 to be set, depending on the model and channel type. If condition code 1 is set, the CSW is stored at location 64 to identify the interruption condition, and the interruption condition is cleared.

When the situation described in the following paragraph occurs with the channel either available or interruption-pending or, on some channels, working, the TIO function causes the CSW to be stored. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

The subchannel contains an interruption condition due to a terminated operation at the addressed device. The CSW identifies the interruption condition, and the interruption condition is cleared. The subchannel key, CCW

address, and count fields contain the final values for the I/O operation, and the status field may include bits provided by the channel and the device. The interruption condition in the subchannel is not cleared, and the CSW is not stored if the channel is working and has not yet accepted the interruption condition from the device.

When any of the following situations occurs with the channel either available or interruption-pending, the TIO function causes the CSW to be stored. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

1. The subchannel is available, and the I/O device contains an interruption condition or the control unit contains control-unit end for the addressed device. The CSW unit-status field identifies the interruption condition and may contain other bits provided by the device or control unit. The interruption condition is cleared. The busy bit in the CSW is zero. The other fields of the CSW contain zeros unless an equipment error is detected.
2. The subchannel is available, and the I/O device or the control unit is executing a previously initiated operation or the control unit has an interruption condition associated with a device other than the one addressed. The CSW unit-status field contains one in the busy-bit position or, if the control unit is busy, the busy and status-modifier bits are ones. Other fields of the CSW contain zeros unless an equipment error is detected.
3. The subchannel is available, and the I/O device or channel detected an equipment error during execution of the instruction or the addressed device is not ready and does not have any interruption condition. The CSW identifies the error. If the device is not ready, unit check is indicated. No interruption conditions are generated at the I/O device or the subchannel.

When TEST I/O cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout is associated with this status.

When the TIO function is used to clear an interruption condition from the subchannel and the channel has not yet accepted the condition from the device, the function causes the device to be selected

and the interruption condition in the device to be cleared. During certain I/O operations, some types of devices cannot provide their current status in response to TEST I/O. Some magnetic-tape control units, for example, are in such a state when they have provided channel end and are executing the backspace-file operation. When TEST I/O is issued to a control unit in such a state, the unit-status field of the CSW has the busy and status-modifier bits set to ones, with zeros in the other CSW fields. The interruption condition in the device and in the subchannel is not cleared.

On some types of devices, the device never provides its current status in response to TEST I/O, and an interruption condition can be cleared only by permitting an I/O interruption. When TEST I/O is issued to such a device, the unit-status field has the status-modifier bit set to one, with zeros in the other CSW fields. The interruption condition in the device and in the subchannel, if any, is not cleared.

However, at the time the channel assigns the highest priority for interruptions to a condition associated with an operation at the subchannel, the channel accepts the status from the device and clears the corresponding condition at the device. When the TIO function is addressed to a device for which the channel has already accepted the interruption condition, the device is not selected, and the condition in the subchannel is cleared regardless of the type of device and its present state. The CSW contains unit status and other information associated with the interruption condition.

On the byte-multiplexer channel, the TIO function causes the addressed device to be selected only after the channel has serviced all outstanding requests for data transfer for previously initiated operations.

#### ***Program Exceptions:***

Privileged Operation

#### ***Resulting Condition Code:***

- 0 Available
- 1 CSW stored
- 2 Channel or subchannel busy
- 3 Not operational

The condition code set by the TIO function for all possible states of the I/O system is shown in the figure "Condition Codes Set by TEST I/O." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.



### Programming Notes

1. Disabling the CPU for I/O interruptions provides the program with a means of controlling the priority of I/O interruptions selectively by channels. The priority of devices attached on a channel cannot be controlled by the program. The instruction TEST I/O permits the program to clear interruption conditions selectively by I/O device.
2. When a CSW is stored by the TIO function, the interface-control-check and channel-control-check indications may be due to an interruption condition already existing in the channel or may be due to an interruption condition created by the TIO function. Similarly, the unit-check bit set to one with the channel-end, control-unit-end, or device-end bits set to zeros may be due to a situation created by the preceding operation, the I/O device being not ready, or an equipment error detected during the execution of TEST I/O. The instruction TEST I/O cannot be used to clear an interruption condition due to the PCI flag while the subchannel is working.

### Input/Output-Instruction-Exception Handling

Before the channel is signaled to execute an I/O instruction, the instruction is tested for validity by the CPU. Exceptional situations detected at this time cause a program interruption.

The following exception may cause a program interruption:

**Privileged Operation:** An I/O instruction is encountered when the CPU is in the problem state. The instruction is suppressed before the channel has been signaled to execute it. The CSW, the condition code in the PSW, and the state of the addressed subchannel and I/O device are not affected by the attempt to execute an I/O instruction while in the problem state.

### Execution of Input/Output Operations

The channel can execute six commands: write, read, read backward, control, sense, and transfer in channel. Each command except transfer in channel initiates a corresponding I/O operation. The term "I/O operation" refers to the activity initiated by a command in the I/O device and associated subchannel. The subchannel is involved with the

execution of the operation from the initiation of the command until the channel-end signal is received or, in the case of command chaining, until the device-end signal is received. The operation in the device lasts until device end is signaled.

### Blocking of Data

Data recorded by an I/O device may be 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 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.

### Channel-Address Word

The channel-address word (CAW) specifies the subchannel key and the address of the first CCW associated with START I/O or START I/O FAST RELEASE. The channel refers to the CAW only during the execution of START I/O or START I/O FAST RELEASE. The CAW is fetched from storage location 72. The pertinent information thereafter is stored in the subchannel, and the program is free to change the contents of the CAW. Fetching of the CAW by the channel does not affect the contents of the location.

The CAW has the following format:

Key	0000	CCW Address
0	4	8
		31

The fields in the CAW are allocated for the following purposes:

**Subchannel Key:** Bits 0-3 form the access key for all commands associated with START I/O and START I/O FAST RELEASE. This key is matched with a storage key during an I/O operation, as described in the section "Protection Check" later in this chapter.

**CCW Address:** Bits 8-31 designate the location of the first CCW in storage.

Bit positions 4-7 of the CAW must contain zeros. The three low-order bits of the CCW address must be zeros to specify the CCW on integral boundaries for doublewords. If any of these restrictions is violated, or if the CCW address specifies a storage location which is not provided or is protected against fetching or is in a disconnected page, START I/O and, in some cases, START I/O FAST RELEASE, cause the status portion of the CSW to be stored, with the protection-check or program-check bit set to one. In this event, the I/O operation is not initiated.

**Programming Note**

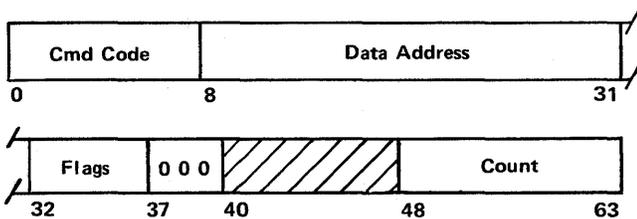
Bit positions 4-7 of the CAW, which presently must contain zeros, may in the future be assigned to the control of new functions. It is, therefore, recommended that these bit positions not be set to ones for the purpose of obtaining an intentional program-check indication.

**Channel-Command Word**

The channel-command word (CCW) specifies the command to be executed and, for commands initiating I/O operations, it designates the storage area associated with the operation and the action to be taken whenever transfer to or from the area is completed. The CCWs can be located anywhere in storage, and more than one can be associated with a START I/O or START I/O FAST RELEASE.

The first CCW is fetched during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When START I/O FAST RELEASE is executed independent of the device, the first CCW is fetched subsequent to the execution of START I/O FAST RELEASE. Each additional CCW in the sequence is obtained when the operation has progressed to the point where the additional CCW is needed. Fetching of the CCWs by the channel does not affect the contents of the location in storage.

The CCW has the following format:



The fields in the CCW are allocated for the following purposes:

**Command Code:** Bits 0-7 specify the operation to be performed.

**Data Address:** Bits 8-31 specify a location in storage. It is the first location referred to in the area designated by the CCW.

**Chain-Data (CD) Flag:** Bit 32, when one, specifies chaining of data. It causes the storage area designated by the next CCW to be used with the current operation.

**Chain-Command (CC) Flag:** Bit 33, when one, and when the CD flag is zero, specifies chaining of commands. It causes the operation specified by the command code in the next CCW to be initiated on normal completion of the current operation.

**Suppress-Length-Indication (SLI) Flag:** Bit 34 controls whether incorrect-length 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 one, command chaining takes place regardless of any incorrect-length situation.

**Skip (SKIP) Flag:** Bit 35, when one, specifies suppression of the transfer of information to storage during a read, read backward, or sense operation.

**Program-Controlled-Interruption (PCI) Flag:** Bit 36, when one, causes the channel to generate an interruption condition when the CCW takes control of the channel. When bit 36 is zero, normal operation takes place.

**Count:** Bits 48-63 specify the number of bytes in the storage area designated by the CCW.

Bit positions 37-39 of every CCW other than one specifying transfer in channel must contain zeros. Otherwise, a program-check condition is generated. When the first CCW designated by the CAW does not contain the required zeros, the I/O operation is not initiated, and the status portion of the CSW with the program-check indication is stored during execution of START I/O or START I/O FAST RELEASE being executed as START I/O. Detection of this condition during data chaining causes the I/O device to be signaled to conclude the operation.

When the absence of these zeros is detected during command chaining or subsequent to the execution of START I/O FAST RELEASE, the new operation is not initiated, and an interruption condition is generated.

The contents of bit positions 40-47 of the CCW are ignored.

**Programming Note**

Bit positions 37-39 of the CCW, which presently must contain zeros, may in the future be assigned to the control of new functions. It is recommended, therefore, that these bit positions not be set to ones for the purpose of obtaining an intentional program-check indication.

**Command Code**

The command code, bit positions 0-7 of the CCW, specifies to the channel and the I/O device the operation to be performed. A detailed description of each command appears under "Commands."

The two low-order bits or, when these bits are 00, the four low-order bits of the command code identify the operation to the channel. The channel distinguishes among the following four operations:

- Output forward (write, control)
- Input forward (read, sense)
- Input backward (read backward)
- Branching (transfer in channel)

The channel ignores the high-order bits of the command code.

Commands that initiate I/O operations (write, read, read backward, control, and sense) cause all eight bits of the command code to be transferred to the I/O device. In these command codes, 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 information 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 SL publication for the device.

The command-code assignment is listed in the following table. The symbol X indicates that the bit position is ignored; M identifies a modifier bit.

Code	Command
XXXX 0000	Invalid
MMMM MM01	Write
MMMM MM10	Read
MMMM 1100	Read Backward
MMMM MM11	Control
MMMM 0100	Sense
XXXX 1000	Transfer in Channel

Whenever the channel detects an invalid command code during the initiation of a command, a program check is generated. When the first CCW designated by the CAW contains an invalid command code, the status portion of the CSW with the program-check indication is stored during execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When the invalid code is detected during command chaining or subsequent to the execution of START I/O FAST RELEASE, the new operation is not initiated, and an interruption condition is generated. The command code is ignored during data chaining, unless it specifies transfer in channel.

**Designation of Storage Area**

The 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 appears in the data-address field of the CCW. The number of bytes contained in the storage area is specified in the count field.

In write, read, control, and sense operations, storage locations are used in ascending order of addresses. As information is transferred to or from storage, the address from the address field is incremented, and the count from the count field is decremented. 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 count reaches zero, the storage area defined by the CCW is exhausted.

Any storage location that is provided can be used in the transfer of data to or from an I/O device, if the location is in a page that is in the addressable or connected state and is not protected against the type of reference. Similarly, a CCW can be located in any part of storage, if the location is in a page that is in the addressable or connected state and is not protected against a fetch-type reference.

When the first CCW is designated by the CAW as being at a storage location that is not provided, the I/O operation is not initiated, and the status portion of the CSW with the program-check indication is stored during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When, subsequently, during the operation or chain of operations, the channel refers to a storage location that is not provided, an interruption condition indicating program check is generated, and the device is signaled to terminate the operation.

When the first CCW designated by the CAW is in a disconnected page or in a location that is protected against a fetch-type reference, the I/O operation is not initiated, and the status portion of the CSW with the protection-check indication is stored during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When, subsequently, during the I/O operation or chain of operations, the channel refers to a disconnected page or a protected location, an interruption condition indicating protection check is generated, and the device is signaled to terminate the operation.

During an output operation, the channel may fetch data from storage before the time the I/O device requests the data. Any number of bytes specified by the current CCW may thus be prefetched. When data chaining during an output operation, and for some block-multiplexer channels when data chaining during an input operation, the channel may prefetch the next CCW at any time during the execution of the current CCW.

Prefetching may cause the channel to refer to storage locations that are protected or not provided or in disconnected pages. Such errors detected during prefetching of data or CCWs do not affect the execution of the operation and do not cause error indications until the I/O operation actually attempts to use the data or until the CCW takes control. If the operation is concluded by the I/O device or by HALT I/O, HALT DEVICE, or CLEAR I/O before the invalid information is needed, no program check or protection check is generated.

The count field in the CCW can specify any number of bytes from one to 65,535. Except for a CCW specifying transfer in channel, which has no count field, the count field may not contain the value zero. Whenever the count field in the CCW initially contains a zero, a program check is generated. When

this occurs in the first CCW designated by the CAW, the operation is not initiated, and the status portion of the CSW with the program-check indication is stored during execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When a count of zero is detected during data chaining, the I/O device is signaled to terminate the operation. Detection of a count of zero during command chaining or subsequent to the execution of START I/O FAST RELEASE suppresses initiation of the new operation and generates an interruption condition.

### ***Chaining***

When the channel has performed the transfer of information specified by a CCW, it can continue the activity initiated by START I/O or START I/O FAST RELEASE by fetching a new CCW. Such fetching of a new CCW is called 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 specified in the original START I/O or START I/O FAST RELEASE.

Two types of chaining are provided: chaining of data and chaining of commands. 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 upon the exhaustion of the current CCW and upon receipt of ending status from the device, as shown in the figure "Channel-Chaining Action."

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 is used for the type of chaining specified in the CCW preceding the transfer in channel. The CD and CC flags are ignored in the transfer-in-channel command.

Flags in Current CCW			Action in Channel upon Exhaustion of Count or Receipt of Channel End			
CD	CC	SLI	Immediate Operation	Regular Operation		
				I	II	III
0	0	0	End, —	Stop, IL	End, —	End, IL
0	0	1	End, —	Stop, —	End, —	End, —
0	1	0	Chain command	Stop, IL	Chain command	End, IL
0	1	1	Chain command	Chain command	Chain command	Chain command
1	0	0	End, —	Chain data	*	End, IL
1	0	1	End, —	Chain data	*	End, IL
1	1	0	End, —	Chain data	*	End, IL
1	1	1	End, —	Chain data	*	End, IL

**Explanation :**

- |  |  |
|--|--|
| <p>I Count exhausted, end of block at device not reached.</p> <p>II Count exhausted and channel end from device.</p> <p>III Count not exhausted and channel end from device.</p> <p>End The operation is terminated. If the operation is immediate and has been specified by the first CCW associated with a START I/O, a condition code 1 is set, and the status portion of the CSW is stored as part of the execution of the START I/O. In all other cases an interruption condition is generated in the subchannel.</p> <p>Stop The device is signaled to terminate data transfer, but the subchannel remains in the working state until channel end is received; at this time an interruption condition is generated in the subchannel.</p> <p>IL Incorrect length is indicated with the interruption condition.</p> | <p>— Incorrect length is not indicated.</p> <p>Chain command The channel performs command chaining upon receipt of device end.</p> <p>Chain data The channel immediately fetches a new CCW for the same operation.</p> <p>* The situation where the residual count is zero but data chaining is indicated at the time the device provides channel end cannot validly occur. When data chaining is indicated, the channel fetches the new CCW after transferring the last byte of data designated by the current CCW but before the device provides the next request for data or status transfer. As a result, the channel recognizes the channel end from the device only after it has fetched the new CCW, which cannot contain a count of zero unless a programming error has been made.</p> |
|--|--|

**Channel-Chaining Action**

## Data Chaining

During data chaining, the new CCW fetched by the channel defines a new storage area for the original I/O operation. Execution of the operation at the I/O device is not affected. When all data designated by the current CCW has been transferred to 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 storage or to the device. When the last byte of the transfer has been placed in storage or accepted by the device, the new CCW takes over the control of the operation and replaces the pertinent information in the subchannel. If the device signals 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 CSW 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 channel end after the new CCW takes control but before transferring any data designated by the new CCW, program check or protection check is indicated in the CSW associated with the termination. The contents of the CSW pertain to the new CCW unless a program check or protection check is generated while fetching the new CCW or while fetching or executing an intervening transfer-in-channel command. A data address which causes a program check or protection check gives an error indication only after the I/O device has attempted to transfer data to or from the addressed storage location.

When data chaining during an output operation, the channel may fetch the new CCW from storage ahead of the time data chaining occurs. Similarly, some block-multiplexer channels may prefetch the new CCW when data chaining during input. Any programming errors in a prefetched CCW, however, do not affect the execution of the operation until all data designated by the current CCW has been transferred to the I/O device on output or to storage on input. If the device concludes the operation before all data designated by the current CCW has been transferred, or if data chaining is suppressed for

any other reason, the errors associated with the prefetched CCW are not indicated to the program.

Only one CCW describing a data area may be prefetched. If the prefetched CCW specifies transfer in channel, only one more CCW may be fetched before the exhaustion of the current CCW.

## Programming Note

Data chaining may be used to rearrange data as it is transferred between storage and an I/O device. Data chaining permits data to be transferred to or from noncontiguous areas of storage, and, when used in conjunction with the skipping function (see the section "Skipping" later in this chapter), data chaining enables the program to place in storage selected portions of a block of data.

When, during an input operation for a channel that does not prefetch CCWs on input, the program specifies data chaining to a location into which data has been placed under the control of the current CCW, the channel, 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, on input, a channel program data-chains to a CCW placed in storage by the CCW specifying data chaining, the block is said to be self-describing. A self-describing block contains one or more CCWs that specify storage locations and counts for subsequent data in the same 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 is signaled only at the completion of data transfer. The error normally does not prematurely terminate or otherwise affect the execution of the operation. Thus, there is no assurance that a CCW read as data is valid until the operation is completed. If the CCW is in error, the use of the CCW in the current operation may cause subsequent data to be placed in wrong storage locations with resultant destruction of the contents of those locations.

Self-describing blocks cannot be used with a channel that prefetches CCWs when data chaining on input.

### **Command Chaining**

During command chaining, the new CCW fetched by the channel specifies a new I/O operation. The channel fetches the new CCW and initiates the new operation upon receipt of the device-end signal for the current operation. When command chaining takes place, the completion of the current operation does not generate an interruption condition, 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 at the device.

Command chaining takes place and the new operation is initiated only if no unusual situations have been detected in the current operation. In particular, the channel initiates a new I/O operation by command chaining upon receipt of a status byte signaling one of the following status combinations: device end, device end and status modifier, device end and channel end, device end and channel end and status modifier. In the former two cases, channel end must have been signaled before device end, with all other status bits set to zeros. If status 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. Incorrect length 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 status modifier with device end. When command chaining is specified and no unusual situations have been detected, the combination of status modifier and device end causes the channel to fetch and chain to the CCW whose storage address is 16 higher than that of the current CCW.

When both command and data chaining are used, the first CCW associated with the operation specifies the operation to be executed, and the last CCW indicates whether another operation follows.

### **Programming Note**

Command chaining makes it possible for the program to initiate transfer of multiple blocks by means of a single START I/O or START I/O FAST RELEASE. It also permits a subchannel to be set up for the execution of auxiliary functions, 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 to modify the normal sequence of operations in response to signals provided by the I/O device.

### ***Skipping***

Skipping is the suppression of storage references during an I/O operation. It is defined only for read, read backward, and sense operations and is controlled by the skip flag, which can be specified individually for each CCW. When the skip flag is one, skipping occurs; when 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. The operation at the I/O device proceeds normally, and information is transferred to the channel. The channel keeps updating the count but does not place the information in storage. Chaining is not precluded by skipping. In the case of data chaining, normal operation is resumed if the skip flag in the new CCW is zero.

When the skip flag is set to one, the data address in the CCW is not checked.

### **Programming Note**

Skipping, when combined with data chaining, permits the program to place in storage selected portions of a block 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 execution of an I/O operation. The function is controlled by the PCI flag in the CCW. The flag can be on either in the first CCW specified by START I/O or START I/O FAST RELEASE or in a CCW fetched during chaining. Neither the PCI flag nor the associated interruption affects the execution of the current operation.

Whenever the PCI flag in the CCW is one, an interruption condition is generated in the channel.

When the first CCW associated with an operation contains the PCI flag, either initially or upon command chaining, the interruption may occur as early as immediately upon the initiation of the operation. The PCI flag in a CCW fetched on data chaining causes the interruption to occur after all data designated by the preceding CCW has been transferred. The time of the interruption, however, depends on the model and the current activity in the system and may be delayed even if I/O interruptions are allowed. No predictable relationship exists between the time the interruption due to the PCI flag occurs and the progress of data transfer to or from the area designated by the CCW, but the fields within the CSW pertain to the same instant of time.

If chaining occurs before the interruption due to the PCI flag has taken place, the PCI interruption condition is carried over to the new CCW. This carryover occurs both on data and command chaining and, in either case, the interruption condition is propagated through the transfer-in-channel command. The interruption conditions due to the PCI flags are not stacked; that is, if another CCW is fetched with a PCI flag before the interruption due to the PCI flag of the previous CCW has occurred, only one interruption takes place.

A CSW containing the PCI bit set to one may be stored by an interruption while the operation is still proceeding or by an interruption, TEST I/O, or CLEAR I/O upon the termination of the operation. It cannot be stored by TEST I/O while the subchannel is in the working state.

When the CSW is stored by an interruption before the operation or chain of operations has been concluded, the CCW address is 8 greater than the address of the current CCW, and the count is unpredictable. All unit-status bits in the CSW are zero. If the channel has detected any unusual situations, such as channel-data check, program check, or protection check by the time the interruption occurs, the corresponding channel-status bit is one, although the status in the subchannel is not reset and is indicated again upon the termination of the operation.

A unit-status bit set to one in the CSW indicates that the operation or chain of operations has been

concluded. The CSW in this case has its regular format with the PCI bit set to one.

However, when the interruption due to the PCI flag is delayed until the operation at the subchannel is concluded, two interruptions from the subchannel may still take place. The first interruption indicates and clears the interruption condition due to the PCI flag, and the second provides the CSW associated with the ending status. Whether one or two interruptions occur depends on the model and on whether the interruption condition due to the PCI flag has been assigned the highest priority for interruption at the time of conclusion. TEST I/O or CLEAR I/O addressed to the device associated with an interruption condition in the subchannel clears the interruption condition due to the PCI flag, as well as the one associated with the conclusion.

The setting of the PCI flag is inspected in every CCW except those specifying transfer in channel, where it is ignored. The PCI flag is also ignored during initial program loading.

#### **Programming Notes**

1. Since no unit-status bits are set to ones in the CSW associated with the conclusion of an operation of a selector channel by HALT I/O or HALT DEVICE, unit-status bits and the PCI bit set to ones are not necessary for the operation to be concluded. When status in a selector channel includes PCI at the time the operation is concluded by HALT I/O or HALT DEVICE, the CSW associated with the concluded operation is indistinguishable from the CSW provided by an interruption during execution of the operation.
2. Program-controlled interruption provides a means of alerting the program to the progress of chaining during an I/O operation. It permits programmed dynamic storage allocation.

#### **Commands**

The figure "Channel-Command Codes" lists the command codes for the six commands and indicates which flags are defined for each command. The flags are ignored for all commands for which they are not defined.

<u>Name</u>	<u>Code</u>	<u>Flags</u>				
Write	MMMM MM01	CD	CC	SLI		PCI
Read	MMMM MM10	CD	CC	SLI	SKIP	PCI
Read Backward	MMMM 1100	CD	CC	SLI	SKIP	PCI
Control	MMMM MM11	CD	CC	SLI		PCI
Sense	MMMM 0100	CD	CC	SLI	SKIP	PCI
Transfer In Channel	XXXX 1000					

**Explanation:**

CD Chain data  
 CC Chain command  
 SLI Suppress length indication  
 SKIP Skip  
 PCI Program-controlled interruption  
 IDA Indirect data addressing  
 M Modifier bit  
 X Ignored

**Channel-Command Codes**

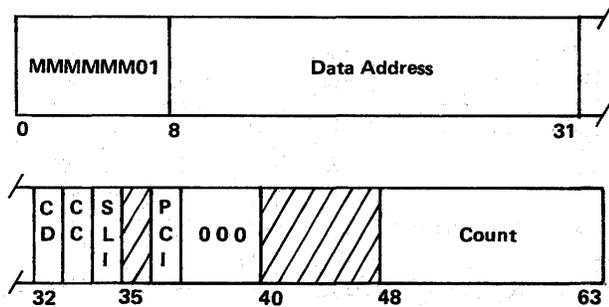
All flags have individual significance, except that the CC and SLI flags are ignored when the CD flag is set to one. The SLI flag is ignored on immediate operations, in which case the incorrect-length indication is suppressed, regardless of the setting of the flag. The PCI flag is ignored during initial program loading.

Each command is described below, and the format is illustrated.

### Programming Note

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 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 storage should not be destroyed until the program has verified that no malfunction affecting the transfer and recording of data was detected.

### Write



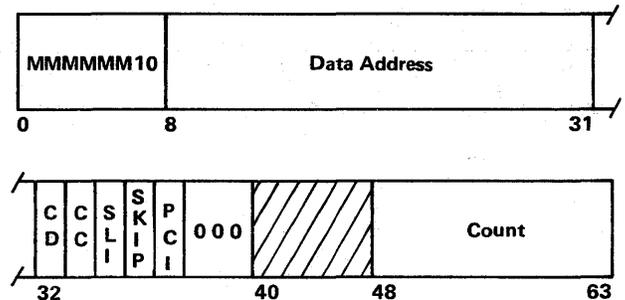
A write operation is initiated at the I/O device, and the subchannel is set up to transfer data from storage to the I/O device. Data in storage is fetched in an ascending order of addresses, starting with the address specified in the CCW.

A CCW used in a write operation is inspected for the CD, CC, SLI, and PCI flags. The setting of the skip flag is ignored. Bit positions 0-5 of the CCW contain modifier bits.

### Programming Note

When writing on devices for which block length is not defined, such as a magnetic-tape unit or an inquiry station, the amount of data written is controlled only by the count in the CCW. Every operation terminated under count control causes the incorrect-length indication, unless the indication is suppressed by the SLI flag.

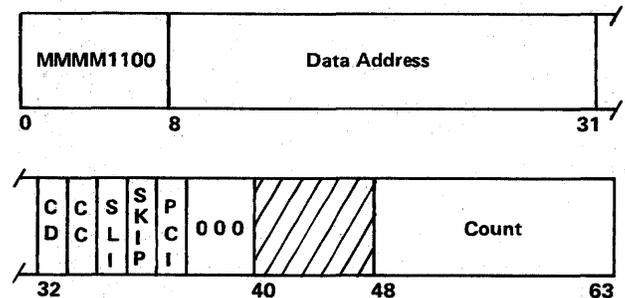
### Read



A read operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to storage. For devices such as magnetic-tape units, disk storage, and card equipment, the bytes of data within a block are provided in the same sequence as written by means of a write command. Data is placed in storage in an ascending order of addresses, starting with the address specified in the CCW.

A CCW used in a read operation is inspected for every flag—CD, CC, SLI, SKIP, and PCI. Bit positions 0-5 of the CCW contain modifier bits.

### Read Backward

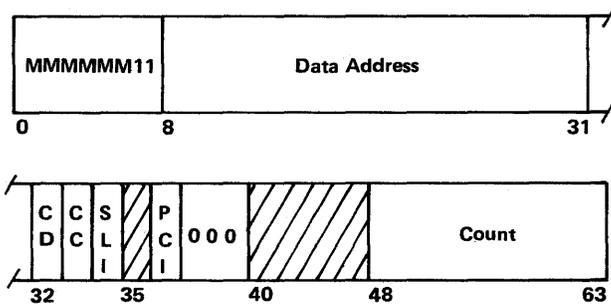


A read-backward operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to storage. On magnetic-tape units, read backward causes reading to be performed with the tape moving backwards. The bytes of data within

a block are sent to the channel in a sequence opposite to that on writing. The channel places the bytes in storage in a descending order of address, starting with the address specified in the CCW. The bits within a byte are in the same order as sent to the device on writing.

A CCW used in a read-backward operation is inspected for every flag—CD, CC, SLI, SKIP, and PCI. Bit positions 0-5 of the CCW contain modifier bits.

### Control



A control operation is initiated at the I/O device, and the subchannel is set up to transfer data from storage to the device. The device interprets the data as control information. The control information, if any, is fetched from storage in an ascending order of addresses, starting with the address specified in the CCW. A control command may be used to initiate at the I/O device an operation not involving transfer of data—such as backspacing or rewinding magnetic tape or positioning a disk-access mechanism.

For many control functions, the entire operation is specified by the modifier bits in the command code, and the function is performed as an immediate operation (see the section "Immediate Operations" later in this chapter). If the command code does not specify the entire control function, the data-address field of the CCW designates the location containing the required additional information. This control information may include a code further specifying the operation to be performed or an external address, such as the disk address for the seek function, and is transferred in response to requests by the device.

A control command code containing zeros for the six modifier bits is defined as a no-operation. The no-operation order causes the addressed device to respond with channel end and device end without causing any action at the device. The control command can be executed as an immediate operation, or the device can delay the status until after the

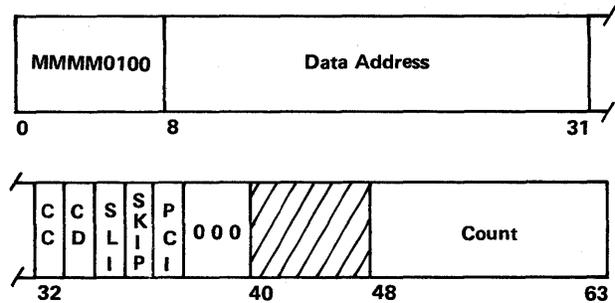
initial selection sequence is completed. Other operations that can be initiated by means of the control command depend on the type of I/O device. These operations and their codes are specified in the SL publication for the device.

A CCW used in a control operation is inspected for the CD, CC, SLI, and PCI flags. The setting of the skip flag is ignored. Bit positions 0-5 of the CCW contain modifier bits.

### Programming Note

Since a CCW (other than transfer in channel) with a count of zero is invalid, the program cannot use the CCW count field to specify that no data be transferred to the I/O device. Any operation terminated before data has been transferred causes the incorrect-length indication, provided the operation is not immediate and has not been rejected during the initiation sequence. The incorrect-length indication is suppressed when the SLI flag is on.

### Sense



A sense operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to storage. The data is placed in storage in an ascending order of addresses, starting with the address specified in the CCW.

Data transferred during a sense operation provides information concerning both unusual conditions detected in the last operation and the status of the device. The status information provided by the sense command is more detailed than that supplied by the unit-status byte in the CSW and may describe reasons for the unit-check indication. It may also indicate, for example, if the device is in the not-ready state, if the tape unit is in the file-protected state, or if magnetic tape is positioned beyond the end-of-tape mark.

For most devices, the first six bits of the sense data describe situations detected during the last operation.

These bits are common to all devices having this type of information and are designated as follows:

<u>Bit</u>	<u>Designation</u>
0	Command reject
1	Intervention required
2	Bus-out check
3	Equipment check
4	Data check
5	Overrun

The following is the meaning of the first six bits:

**Command Reject:** The device has detected a programming error. A command has been received which the device is not designed to execute, such as read backward issued to a direct-access storage device, or which the device cannot execute because of its present state, such as write issued to a file-protected tape unit. Command reject is also indicated when the program issues an invalid sequence of commands, such as write to a direct-access storage device without previous designation of the block.

**Intervention Required:** The last operation could not be executed because of a situation requiring some type of intervention at the device. This bit indicates situations such as the hopper in a card punch being empty or the printer being out of paper. It is also turned on when the addressed device is not ready, is in test mode, or is not provided on the control unit.

**Bus-Out Check:** The device or the control unit has received a data byte or a command code with an invalid parity from the channel. During writing, bus-out check indicates that incorrect data has been recorded at the device, but this does not cause the operation to be terminated prematurely. Parity errors on command codes and control information cause the operation to be immediately terminated and suppress checking for situations that would cause command reject and intervention required.

**Equipment Check:** During the last operation, the device or the control unit has detected equipment malfunctioning, such as an invalid card-hole count or a printer-buffer parity error.

**Data Check:** The device or the control unit has detected a data error other than those included in bus-out check. Data check identifies errors associated with the recording medium and includes errors such as reading an invalid card code or detecting invalid parity on data recorded on magnetic tape.

On an input operation, data check indicates that incorrect data may have been placed in storage. The control unit forces correct parity on data sent to the channel. On writing, data check indicates that incorrect data may have been recorded at the device. Unless the operation is of a type where the error precludes meaningful continuation, data errors on reading and writing do not cause the operation to be terminated prematurely.

**Overrun:** The channel has failed to respond on time to a request for service from the device. Overrun can occur when data is transferred to or from a nonbuffered control unit operating with a synchronous medium, and the total activity initiated by the program exceeds the capability of the channel. When the channel fails to accept a byte on an input operation, the following data transferred to storage may be used to fill the gap. On an output operation, overrun indicates that data recorded at the device may be invalid. The overrun bit is also set to one when the device receives the new command too late during command chaining.

All information significant to the use of the device normally is provided in the first two bytes. Any bit positions following those used for programming information contain diagnostic information, which may extend to as many bytes as needed. The amount and the meaning of the status information are peculiar to the type of I/O device and are specified in the SL publication for the device.

The basic sense command has zero modifier bits. This command initiates a sense operation on all devices and cannot cause the command-reject, intervention-required, data-check, or overrun bit to be set to one. If the control unit detects an equipment malfunction, or invalid parity of the sense command code, the equipment-check or bus-out-check bit is set to one, and unit check is indicated in the unit-status byte.

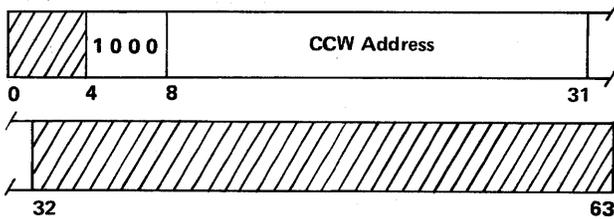
Devices that can provide special diagnostic sense information or can be instructed to perform other special functions by use of the sense command may define modifier bits for the control of these functions. The special sense operations may be initiated by a

unique combination of modifier bits, or a group of codes may specify the same function. Any remaining sense command codes may be considered invalid, thus causing the unit-check indication, or may cause the same action as the basic sense command, depending upon the type of device.

The sense information that pertains to the last I/O operation or other action at a device may be reset any time after the completion of a sense command addressed to that device. Any command addressed to the control unit of a device, other than the no-operation command and the command which results from a TEST I/O instruction, may be allowed to reset the sense information, provided that the busy bit is not included in the initial status. The sense information may also be changed as a result of asynchronous action, such as when attention or not-ready-to-ready device-end status is generated.

A CCW used in a sense operation is inspected for every flag—CD, CC, SLI, SKIP, and PCI. Bit positions 0-3 of the CCW contain modifier bits.

#### Transfer in Channel



The next CCW is fetched from the location in 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 at the channel, and the I/O device is not signaled. 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.

The first CCW designated by the CAW must not specify transfer in channel. When this restriction is violated, no I/O operation is initiated, and a program check is generated. The error causes the status portion of the CSW, with the program-check status bit set to one, to be stored during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When START I/O FAST RELEASE is executed independent of the device, the error causes an interruption condition to be generated.

To address a CCW on integral boundaries for doublewords, a CCW specifying transfer in channel must contain zeros in bit positions 29-31.

Furthermore, a CCW specifying a transfer in channel must not be fetched from a location designated by an immediately preceding transfer in channel. When either of these errors is detected, a program check is generated.

The contents of the second half of the CCW, bit positions 32-63, are ignored. Similarly, the contents of bit positions 0-3 of the CCW are ignored.

#### Command Retry

Some channels have the capability to perform command retry, a channel and control-unit 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 communication sequence with the channel. When immediate retry can be performed, it signals 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 the channel is not capable of performing command retry, or when any status bit other than device end accompanies the requested command retry initiation, the retry is suppressed, and an interruption condition is generated. The CSW will contain the channel-end, unit-check, and status-modifier status indications, along with any other appropriate status.

During command retry, the channel action is similar to that taken when command chaining. Thus, when command retry is performed, a START I/O initiating an immediate operation for which command chaining is not indicated in the CCW causes a condition code 0, rather than a condition code 1, to be set. The subsequent termination of the I/O operation causes an interruption condition to be generated. During command retry, the CCW may be refetched.

#### Programming Note

The following possible results of a command retry must be anticipated by the program:

1. A CCW with the PCI flag set to one may, if retried because of command retry, cause multiple PCI interruptions to occur.
2. A channel program consisting of a single, unchained CCW specifying an immediate command may cause a condition code 0 rather

than a condition code 1 to be set. This setting of the condition code occurs if the control unit signals command retry at the time initial status is signaled to the channel. An interruption condition is generated upon completion of the operation.

3. If a CCW used in an operation is changed before that operation has been successfully completed, the results are unpredictable.
4. A CSW stored after the initiation of a retry but before the presentation of device end, as when an interruption condition due to the PCI flag is taken, contains the address of the command to be retried plus 8.
5. If a HALT I/O, HALT DEVICE, or CLEAR I/O instruction is issued after the initiation of a retry but before the presentation of device end, the CSW contains the address of the command to be retried plus 8.
6. On a multiplexer channel, chained CCWs which might ordinarily have been executed in a burst may, upon the occurrence of command retry, cause multiplexing to occur, with the result that the channel becomes unexpectedly available.

## Conclusion of Input/Output Operations

When the operation or sequence of operations initiated by START I/O or START I/O FAST RELEASE is ended, the channel and the device generate status. Status can be brought to the attention of the program by means of an I/O interruption, by TEST I/O or CLEAR I/O, or, in certain cases, by START I/O or START I/O FAST RELEASE. This status, 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 channel-status word (CSW).

### *Types of Conclusion*

Normally an I/O operation at the subchannel lasts until the device signals channel end. Channel end can be signaled during the sequence initiating the operation, or later. When the channel detects equipment malfunctioning or an I/O system reset is performed, the channel disconnects the device without receiving channel end. The program can force a device to be disconnected prematurely by issuing CLEAR I/O, HALT I/O, or HALT DEVICE.

## Conclusion at Operation Initiation

After the addressed channel and subchannel have been verified to be in a state where START I/O or START I/O FAST RELEASE can be executed, certain tests are performed on the validity of the information specified by the program and on the availability of the addressed control unit and I/O device. This testing occurs during the execution of START I/O, either during or subsequent to the execution of START I/O FAST RELEASE, and during command chaining.

A data-transfer operation is initiated at the subchannel and device only when no programming or equipment errors are detected by the channel and when the device responds with zero status during the initiation sequence. When the channel detects or the device signals any unusual situations during the initiation of an operation, the command is said to be rejected.

Rejection of the command during the execution of START I/O or START I/O FAST RELEASE is indicated by the setting of the condition code in the PSW. Unless the device is not operational, the reasons for the rejection are detailed by the portion of the CSW stored by START I/O or START I/O FAST RELEASE. The device is not started, no interruption conditions are generated, and the subchannel is available subsequent to the initiation sequence. The device is immediately available for the initiation of another operation, provided the command was not rejected because the device was busy or not operational.

When an unusual situation causes a command to be rejected during initiation of an I/O operation by command chaining, an interruption condition is generated, and the subchannel is not available until the condition is cleared. The reasons for the rejection are indicated to the program by means of the corresponding status bits in the CSW. The not-operational state of the I/O device, which during the execution of START I/O and sometimes during the execution of START I/O FAST RELEASE causes condition code 3 to be set, instead causes the interface-control-check bit to be set to one. The new operation at the I/O device is not started.

When START I/O FAST RELEASE is executed by a channel independent of the addressed device, tests for most program-specified information, for control-unit and device availability, for control-unit and device status, and for most errors are performed subsequent to the execution of START I/O FAST RELEASE. Some situations which would have caused

a condition code 1 or 3 to be set had the instruction been START I/O instead cause an interruption condition to be generated. The CSW, when stored, indicates that the interruption condition is a deferred condition code 1 or 3.

### **Immediate Operations**

Some control commands cause the I/O device to signal channel end immediately upon receipt of the command code. An I/O operation causing channel end to be signaled during the initiation sequence is called an immediate operation.

When the first CCW designated by the CAW during a START I/O or START I/O FAST RELEASE executed as a START I/O initiates an immediate operation with command chaining not indicated and command retry not occurring, no interruption condition is generated. In this case, channel end is brought to the attention of the program by causing START I/O or START I/O FAST RELEASE to store the CSW status portion. The subchannel is immediately made available to the program. The I/O operation, however, is initiated, and, if channel end is not accompanied by device end, the device remains busy. Device end, when subsequently provided by the device, causes an interruption condition to be generated.

An immediate operation initiated by the first CCW designated by the CAW during a START I/O FAST RELEASE executed independent of the addressed device appears to the program as a nonimmediate command. That is, any status generated by the device for the immediate command or for a subsequent command if command chaining occurs, causes an interruption condition to be generated.

When command chaining is specified after an immediate operation and no unusual situations have been detected during the execution, or when command retry occurs for an immediate operation, neither START I/O nor START I/O FAST RELEASE causes the immediate storing of CSW status. The subsequent commands in the chain are handled normally, and channel end for the last operation generates an interruption condition even if the device provides the signal immediately upon receipt of the command code.

Whenever immediate completion of an I/O operation is signaled, no data has been transferred to or from the device.

Since a count of zero is not valid, any CCW specifying an immediate operation must contain a nonzero count. When an immediate operation is

executed, however, incorrect length is not indicated to the program, and command chaining is performed when so specified.

### **Programming Note**

Control operations for which the entire operation is specified in the command code may be executed as immediate operations. Whether the control function is executed as an immediate operation depends on the operation and type of device and is specified in the SL publication for the device.

### **Conclusion of Data Transfer**

When the device accepts a command, the subchannel is set up for data transfer. The subchannel is in the working state during this period. Unless the channel detects equipment malfunctioning or the operation is concluded by CLEAR I/O, or, on the selector channel, the operation is concluded by CLEAR I/O, HALT I/O, or HALT DEVICE, the working state lasts until the channel receives the channel-end signal from the device. When no command chaining is specified or when chaining is suppressed because of unusual situations, channel end causes the operation at the subchannel to be terminated and an interruption condition to be generated. The status bits in the associated CSW indicate channel end and any unusual situations. The device can signal channel end at any time after initiation of the operation, and the signal may occur before any data has been transferred.

For operations not involving data transfer, the device normally controls the timing of channel end. The duration of data-transfer operations may be variable and may be controlled by the device or the channel.

Excluding equipment errors, CLEAR I/O, HALT DEVICE, and HALT I/O, the channel signals the device to conclude data transfer whenever any of the following events occurs:

1. The storage areas specified for the operation are exhausted or filled.
2. A program check is detected.
3. A protection check is detected.
4. A chaining check is detected.

The first event occurs when the channel has stepped the count to zero in the last CCW associated with the operation. A count of zero indicates that the channel has transferred all information specified by the program. The other three events are due to errors and cause premature conclusion of data transfer. In every 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 from magnetic tape), it concludes the operation and generates channel end.

The device can control the duration of an operation and the timing of channel end. On certain operations for which blocks are defined (such as reading from magnetic tape), the device does not provide the channel-end signal until the end of the block is reached, regardless of whether or not the device has been previously signaled to conclude data transfer.

If the initial data address in the CCW refers to a storage location that is not provided or to a disconnected or protected page, 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 refers to a storage location that is not provided or to a disconnected or protected page, the operation is concluded before the recording medium has been advanced. However, since the operation has been initiated, the device provides channel end, and an interruption condition is generated. Whether a block at the device is advanced when no data is transferred depends on the type of device and is specified in the SL publication for the device.

When command chaining takes place, the subchannel is in the working state from the time the first operation is initiated until the device signals channel end for the last operation of the chain. On the selector channel, the device executing the operation stays connected to the channel and the whole channel is in the working state during the entire execution of the chain of operations. On the multiplexer channel, an operation in the burst mode causes the channel to be in the working state only while transferring a burst of data. If channel end and device end do not occur concurrently, the device disconnects from the channel after providing channel end, and the channel can in the meantime communicate with other devices.

Any unusual situations cause command chaining to be suppressed and an interruption condition to be generated. The unusual situations can be detected by either the channel or the device, and the device can provide the indications with channel end, control-unit end, or device end. When the channel is aware of the unusual situation by the time the channel-end signal for the operation is received, the chain is ended as if

the operation during which the situation occurred were the last operation of the chain. The device-end signal subsequently is processed as an interruption condition. When the device signals unit check or unit exception with control-unit end or device end, the subchannel terminates the working state upon receipt of the signal from the device. The channel-end indication in this case is not made available to the program.

#### **Termination by HALT I/O or HALT DEVICE**

The instructions HALT I/O and HALT DEVICE cause the current operation at the addressed channel or subchannel to be immediately terminated. 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 I/O or HALT DEVICE is not necessarily contingent on the receipt of a service request from the device.

When HALT I/O is issued to a channel operating in burst mode, the channel issues the halt signal to the device currently operating with the channel, regardless of the device address specified with the HALT I/O instruction. If the channel is involved in the data-transfer portion of an operation, data transfer is immediately terminated, and the device is disconnected from the channel. If HALT I/O is addressed to a selector channel executing a chain of operations and the device has already provided channel end for the current operation, the instruction causes the device to be disconnected and command chaining to be immediately suppressed.

When HALT DEVICE is issued to a channel operating in burst mode, the halt signal is issued to the device involved in the burst-mode operation only if that device is the one to which the HALT DEVICE is addressed. If the operation thus terminated is in the data-transfer portion of the operation, data transfer is immediately terminated, and the device is disconnected from the channel. If the terminated burst involves a selector channel executing a chain of operations and the device has already provided channel end for the current operation, HALT DEVICE causes the device to be disconnected and command chaining to be immediately suppressed. If, on a selector channel, the device involved in the burst is not the one to which the HALT DEVICE is addressed, no action is taken. If, on a multiplexer channel, the device involved in the burst is not the one to which the HALT DEVICE is addressed, HALT DEVICE causes any operation for the

addressed device to be terminated at the addressed subchannel by suppressing any further data transfer or command chaining for that device.

When HALT I/O or HALT DEVICE is issued to a channel not operating in burst mode, the addressed device is selected, and the halt signal is issued as the device responds. On a multiplexer channel, command chaining, if indicated in the subchannel, is immediately suppressed.

The termination of an operation by HALT I/O or HALT DEVICE on the selector channel results in up to four distinct interruption conditions. The first one is generated by the channel upon execution of the instruction and is not contingent on the receipt of status from the device. The channel-status bits reflect the unusual situations, if any, detected during the operation. If HALT I/O or HALT DEVICE is issued before all data specified for the operation has been transferred, incorrect length is indicated, subject to the control of the SLI flag in the current CCW. The execution of HALT I/O or HALT DEVICE itself is not reflected in CSW status, and all status bits in a CSW due to this interruption condition can be zero. The channel is available for the initiation of a new I/O operation as soon as the interruption condition is cleared.

The second interruption condition on the selector channel occurs when the control unit signals channel end. The selector channel handles this condition as any other interruption condition from the device after the device has been disconnected from the channel, and provides zeros in the subchannel-key, CCW-address, count, and channel-status fields of the associated CSW. Channel end is not made available to the program when HALT I/O or HALT DEVICE is issued to a channel executing a chain of operations and the device has already provided channel end for the current operation.

Finally, the third and fourth interruption conditions occur when control-unit end, if any, and device end are signaled. These signals are handled as for any other I/O operation.

The termination of an operation by HALT I/O or HALT DEVICE on a multiplexer channel causes the normal interruption conditions to be generated. If the instruction is issued when the subchannel is in the data-transfer portion of an operation, the subchannel remains in the working state until channel end is signaled by the device, at which time the subchannel is placed in the interruption-pending state. If HALT I/O or HALT DEVICE is issued after the device has signaled channel end and the subchannel is executing

a chain of operations, channel-end is not made available to the program, and the subchannel remains in the working state until the next status byte from the device is received. Receipt of a status byte subsequently places the subchannel in the interruption-pending state.

The CSW associated with the interruption condition in the subchannel contains the status byte provided by the device and the channel. If HALT I/O or HALT DEVICE is issued before all data areas associated with the current operation have been exhausted or filled, incorrect length is indicated, subject to the control of the SLI flag in the current CCW. The interruption condition is processed as for any other type of termination.

The termination of a burst operation by HALT I/O or HALT DEVICE on a block-multiplexer channel may, depending on the model and the type of subchannel, take place as for a selector channel or may allow the subchannel to remain in the working state until the device provides ending status.

#### **Programming Note**

The count field in the CSW associated with an operation terminated by HALT I/O or HALT DEVICE is unpredictable.

#### **Termination by CLEAR I/O**

The termination of an operation by CLEAR I/O causes the subchannel to be set to the available state and causes a CSW to be stored. The validity of the CSW fields is defined in the section "CLEAR I/O" earlier in this chapter.

When CLEAR I/O terminates an operation at a subchannel in the interruption-pending state, up to three subsequent interruption conditions related to the operation can occur. Since CLEAR I/O causes the subchannel to be made available, these interruption conditions will result in only the unit-status portion of the CSW being indicated.

The first interruption condition arises on a selector channel when channel end is signaled to the channel. This occurs only when the interruption-pending states of the channel and subchannel at the execution of CLEAR I/O were due to the previous execution of HALT I/O or HALT DEVICE.

The second and third interruption conditions arise when control-unit end, if any, and device end are signaled to the channel.

When CLEAR I/O terminates an operation at a subchannel in the working state, up to four subsequent interruption conditions related to the

operation can occur. For all of these conditions, only the status portion of the CSW is indicated.

The first interruption condition arises on certain channels when the terminated operation was in the midst of data transfer. Since the device is not signaled to terminate the operation during the execution of CLEAR I/O unless the channel is working with the addressed device when the instruction is received, the device may, subsequent to the CLEAR I/O, attempt to continue the data transfer. The channel responds by signaling the device to terminate data transfer. Depending on the channel, the need to signal the device to terminate data transfer may be ignored or may be considered an interface-control check which creates an interruption condition. Only channel status is indicated in the CSW.

The second interruption condition occurs when channel-end status is received from the device. The third and fourth conditions occur when control-unit end, if any, and device end are presented to the channel. In these three cases, only unit status is indicated in the CSW.

#### **Termination Due to Equipment Malfunction**

When channel-equipment malfunctioning is detected or invalid signals are received from a device, the recovery procedure and the subsequent states of the subchannels and devices on the channel depend on the type of error and on the model. Normally, the program is alerted to the termination by an I/O interruption, and the associated CSW indicates channel-control check or interface-control check. However, when the nature of the malfunction prevents an I/O interruption, a machine-check interruption occurs, and a CSW is not stored. A malfunction may cause the channel to perform the I/O selective reset or to generate the halt signal.

#### ***Input/Output Interruptions***

Input/output interruptions provide a means for the CPU to change its state in response to conditions that occur in I/O devices or channels. These conditions can be caused by the program or by an external event at the device.

#### **Interruption Conditions**

A request for an I/O interruption is called an I/O-interruption condition, or, in this chapter, simply an interruption condition. An interruption condition can be brought to the attention of the program only once and is cleared when it causes an interruption. Alternatively, an interruption condition can be cleared by TEST I/O or CLEAR I/O, and conditions generated by the I/O device following the termination of the operation at the subchannel can be cleared by START I/O or START I/O FAST RELEASE. The latter include interruption conditions caused by attention, device end, and control-unit end, and channel end when provided by a device after conclusion of the operation. The device attempts to initiate a request to the channel for an interruption whenever it detects any of the following:

- Channel end
- Control-unit end
- Device end
- Attention

The channel may also, at command chaining, create an interruption condition at the device, which can be due to the following:

- Unit check
- Unit exception
- Busy indication from device
- Program check
- Protection check

When an operation initiated by command chaining is terminated because of an unusual situation detected during the command initiation sequence, the interruption condition may remain pending within the channel, or the channel may create an interruption condition at the device. An interruption condition is created at the device in response to presentation of status by the device and causes the device subsequently to present the same status for interruption purposes. The interruption condition at the device may or may not be associated with unit status. If the unusual situation is detected by the device (unit check or unit exception) the unit-status field of the associated CSW identifies the condition. In the case of program and protection check, the identification of the error is preserved in the subchannel, and appears in the channel-status field of the associated CSW. If the associated interruption condition has been queued at the device, the device provides zero status for interruption purposes. When command chaining takes place, channel end and

device end do not cause an interruption, and are not made available.

An interruption condition caused by the device may be accompanied by channel and other unit status. Furthermore, more than one interruption condition associated with the same device can be cleared at the same time. As an example, when channel end is not cleared at the device by the time device end is generated, both may be indicated in the CSW and cleared at the device concurrently.

However, at the time the channel assigns highest priority for interruptions to an interruption condition associated with an operation at the subchannel, the channel accepts the status from the device and clears the condition at the device. The interruption condition and the associated status indication are subsequently preserved in the subchannel. Any subsequent status generated by the device is not included when the CSW is stored, even if the status is generated before the interruption condition is cleared.

When the channel detects any of the following, it generates an interruption condition without necessarily communicating with or having received the status byte from the device:

- PCI flag in a CCW
- Execution of HALT I/O or HALT DEVICE on a selector channel
- Channel-available interruption (CAI)
- A programming error associated with the CCW or first IDAW following the SIOF function

The interruption conditions from the channel, except for CAI, can be accompanied by other channel-status indications, but none of the device status bits is on when the channel initiates the interruption.

#### **Channel-Available Interruption**

The channel-available-interruption (CAI) condition is provided on block-multiplexer channels and causes the entire CSW to be replaced by a new set of bits. All fields of the CSW are set to zero. The I/O address stored contains a zero device address and a channel address identifying the interrupting channel.

The channel generates the CAI condition only if it previously had responded with a condition code 2 to an I/O instruction other than HALT I/O or HALT DEVICE and if the working state thus indicated no longer exists. When the working state which caused condition code 2 was due to a subchannel busy with a device other than the one addressed, the conclusion of the working state is not signaled by a CAI. Since any other interruption condition (except PCI)

accomplishes the same function as CAI, a CAI condition is reset upon the occurrence of any interruption (except PCI) on that channel. Some channels also reset a CAI condition when another interruption condition (except PCI) is cleared by a TEST I/O on the same channel. The occurrence of another channel-working state before the CAI causes the CAI condition to be suspended until the working state ends.

#### **Programming Note**

The CAI is designed to inform the program that a channel which previously indicated busy is no longer busy. The CAI condition pending in a channel does not cause the rejection of a subsequent START I/O or START I/O FAST RELEASE but does cause a condition code 1 to be returned to TEST CHANNEL. The CAI can therefore be used as a tool for keeping I/O requests in sequence by using it in conjunction with TEST CHANNEL. A channel which responded with condition code 2 because the channel was busy does not subsequently respond with a condition code 0 to a TEST CHANNEL without clearing an interruption condition in the interim.

#### **Priority of Interruptions**

Generation of interruption conditions is asynchronous to the activity in the CPU, and interruption conditions associated with more than one I/O device can exist at the same time. The priority among interruption conditions is controlled by two types of mechanisms—one establishes the priority among interruption conditions within a channel, and another establishes priority among interruption conditions from different channels. A channel requests an I/O interruption only after it has established priority among interruption conditions. The status associated with interruption conditions is preserved in the devices or channels until accepted by the CPU.

Assignment of priority among requests for interruption associated with devices on any one channel is a function of the type of channel, the type of interruption condition, and the position of the device on the I/O interface. A device's position on the interface is not related to its address. Interruption conditions from different devices do not necessarily occur in the sequence in which they are generated. However, multiple interruption conditions for a single device are presented in the sequence in which they are generated.

The priorities among requests for I/O interruptions from different channels depend on channel addresses.

The priorities of channels 1-15 are in the order of their addresses, with channel 1 having the highest priority. The priority of byte-multiplexer channel 0 is undefined. Its priority may be above, below, or between those priorities of channels 1-15.

**Interruption Action**

An I/O interruption can occur only when the CPU is enabled for I/O interruptions. The interruption occurs at the completion of a unit of operation. If a channel has established the priority among interruption conditions, 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. This interruption is associated with the highest priority condition for the channel. If interruptions are allowed from more than one channel concurrently, the interruption occurs from the channel having the highest priority among those requesting interruption.

If the priority among interruption conditions has not yet been established in the channel by the time the interruption is allowed, the interruption does not necessarily occur immediately after the completion of the instruction enabling the CPU. This delay can occur regardless of how long the interruption condition has existed in the device or the subchannel.

The interruption causes the current program-status word (PSW) to be stored as the old PSW at location 56 and causes the CSW associated with the interruption to be stored at location 64.

Subsequently, a new PSW is loaded from location 120, and processing resumes in the state indicated by this PSW. The channel and device causing the interruption are identified by the I/O address which, in the EC mode, is stored in locations 186-187 and, in the BC mode, is contained in bits 16-31 of the I/O old PSW. The CSW associated with the interruption identifies the interruption condition responsible for the interruption and provides further details about the progress of the operation and the status of the device.

**Programming Note**

When a number of I/O devices on a shared control unit are concurrently executing operations such as rewinding tape or positioning a disk-access mechanism, the initial device-end signals generated on completion of the operations are provided in the order of generation, unless command chaining is specified for the operation last initiated. In the latter

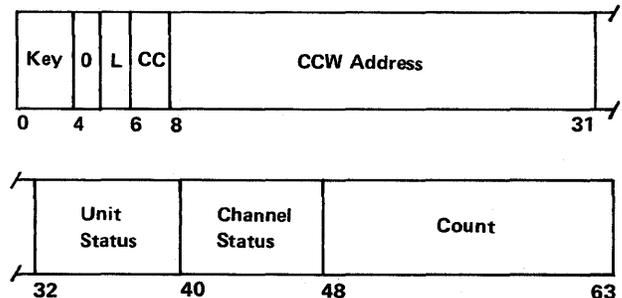
case, the control unit provides the device-end signal for the last initiated operation first, and the other signals are delayed until the subchannel is freed. Whenever interruptions due to the device-end signals are delayed because the CPU is disabled for I/O interruptions or the subchannel is busy, the original order of the signals is destroyed.

**Channel-Status Word**

The channel-status word (CSW) provides to the program the status of an I/O device or the indication of the reasons for which an I/O operation has been concluded. The CSW is formed, or parts of it are replaced, in the process of I/O interruptions and possibly during the execution of START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, HALT DEVICE, and STORE CHANNEL ID. The CSW is stored at location 64 and is available to the program at this location until the time the next I/O interruption occurs or until another I/O instruction causes its contents to be replaced, whichever occurs first.

The information placed in the CSW by an I/O interruption pertains to the device which is identified by the I/O address stored during the interruption. The information placed in the CSW by START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, or HALT DEVICE pertains to the device addressed by the instruction.

The CSW has the following format:



The fields in the CSW are allocated as follows:

**Subchannel Key:** Bits 0-3 form the access key used in the chain of operations at the subchannel.

**Logout Pending (L):** Bit 5, when one, indicates that an I/O instruction cannot be executed until a logout has been cleared. Bit 45, channel-control check, will always be one when bit 5 is one.

**Deferred Condition Code (CC):** Bits 6 and 7 indicate whether situations have been encountered subsequent to the setting of a condition code 0 for START I/O FAST RELEASE that would have caused a different condition-code setting for START I/O. The possible setting of these bits, and their meanings, are as follows:

Setting of		Meaning
Bit 6	Bit 7	
0	0	Normal I/O interruption
0	1	Deferred condition code is 1
1	0	(Reserved)
1	1	Deferred condition code is 3

**CCW Address:** Bits 8-31 form an address that is 8 higher than the address of the last CCW used.

**Status:** Bits 32-47 identify the status of the device and the channel that caused the storing of the CSW. Bits 32-39, the unit status, are obtained from the device or control unit and indicate situations detected by the device or control unit. Bits 40-47, the channel status, are provided by the channel and indicate situations associated with the subchannel. The 16 bits are designated as follows:

Bit	Designation
32	Attention
33	Status modifier
34	Control-unit end
35	Busy
36	Channel end
37	Device end
38	Unit check
39	Unit exception
40	Program-controlled interruption
41	Incorrect length
42	Program check
43	Protection check
44	Channel-data check
45	Channel-control check
46	Interface-control check
47	Chaining check

**Count:** Bits 48-63 form the residual count for the last CCW used.

### Unit Status

The following status indications are generated by the I/O device or control unit. The timing and causes of these status indications for each type of device are specified in the SL publication for the device.

When the I/O device is accessible from more than one channel, status due to channel-initiated operations is signaled to the channel that initiated the

associated I/O operation. The handling of status not associated with I/O operations, such as attention or device end due to transition from the not-ready to the ready state, depends on the type of device and situation and is specified in the SL publication for the device.

### Attention

Attention is signaled when the device detects an asynchronous situation that is significant to the program. Attention is interpreted by the program and is not associated with the initiation, execution, or conclusion of an I/O operation.

The device can signal attention to the channel when no operation is in progress at the I/O device, control unit, or subchannel. Attention can be signaled with device end upon completion of an operation, and it can be signaled to the channel during the initiation of a new I/O operation. Otherwise, the handling and presentation of attention to the channel depends on the type of device.

When the device signals attention during the initiation of an operation, the operation is not initiated. Attention causes command chaining to be suppressed.

### Status Modifier

Status modifier is generated by the device when the device cannot provide its current status in response to TEST I/O, when the control unit is busy, when the normal sequence of commands has to be modified, or when command retry is to be initiated.

When status modifier is signaled in response to TEST I/O and status modifier is the only status bit that is set to one, this indicates that the device cannot execute the instruction and has not provided its current status. The interruption condition, which may be pending at the device or subchannel, has not been cleared, and the CSW stored by TEST I/O contains zeros in the subchannel-key, CCW-address, and count fields.

When the status-modifier bit in the CSW is set to one together with the busy bit, it indicates that the busy status pertains to the control unit associated with the addressed I/O device. The control unit appears busy when it is executing a type of operation that precludes the acceptance and execution of any command or the instructions TEST I/O, HALT I/O, and HALT DEVICE or when it contains an interruption condition for a device other than the one addressed. The interruption condition may be due to control-unit end, due to channel end following the

execution of CLEAR I/O, or, on the selector channel, due to channel end following the execution of HALT I/O or HALT DEVICE. The busy state occurs for operations such as backspace file, in which case the control unit remains busy after providing channel end, for operations concluded by CLEAR I/O, and for operations concluded on the selector channel by HALT I/O or HALT DEVICE, and temporarily occurs on the 2702 Transmission Control after initiation of an operation on a device accommodated by the control unit. A control unit accessible from two or more channels appears busy when it is communicating with another channel.

Presence of status modifier and device end means that the normal sequence of commands must be modified. The handling of this status combination by the channel depends on the operation. If command chaining is specified in the current CCW and no unusual situations have been detected, presence of status modifier and device end causes the channel to fetch and chain to the CCW whose storage address is 16 higher than that of the current CCW. If the I/O device signals status modifier at a time when no command chaining is specified, or when any unusual situations have been detected, no action is taken in the channel, and the status-modifier bit is set to one in the CSW.

Status modifier is set to one in combination with unit check and channel end to initiate the command-retry procedure.

### **Control-Unit End**

Control-unit end indicates that the control unit has become available for use for another operation.

Control-unit end is provided only by control units shared by I/O devices or control units accessible by two or more channels, and only when one or both of the following have occurred:

1. The program had previously caused the control unit to be interrogated while the control unit was in the busy state. The control unit is considered to have been interrogated in the busy state when a command or the instructions TEST I/O, HALT I/O, or HALT DEVICE had been issued to a device on the control unit, and the control unit had responded with busy and status modifier in the unit-status byte. See the section "Status Modifier" earlier in this chapter.
2. The control unit detected an unusual situation during the portion of the operation after channel end had been signaled to the channel. The

indication of the unusual situation accompanies control-unit end.

If the control unit remains busy with the execution of an operation after signaling channel end but has not detected any unusual situations and has not been interrogated by the program, control-unit end is not generated. Similarly, control-unit end is not provided when the control unit has been interrogated and could perform the indicated function. The latter case is indicated by the absence of busy and status modifier in the response to the instruction causing the interrogation.

When the busy state of the control unit is temporary, control-unit end is included with busy and status modifier in response to the interrogation even though the control unit has not yet been freed. The busy condition is considered to be temporary if its duration is commensurate with the program time required to handle an I/O interruption. The 2702 Transmission Control is an example of a device in which the control unit may be busy temporarily and which includes control-unit end with busy and status modifier.

Control-unit end can be signaled with channel end, with device end, or between the two. When control-unit end is signaled by means of an I/O interruption in the absence of any other status, the interruption may be identified by any address assigned to the control unit. A control-unit end causes the control unit to appear busy for initiation of new operations.

### **Busy**

Busy indicates that the I/O device or control unit cannot execute the command or instruction because (1) it is executing a previously initiated operation, (2) it contains an interruption condition, (3) it is shared by channels or I/O devices and the shared facility is not available, or (4) a self-initiated function is being performed. The status associated with the interruption condition for the addressed device, if any, accompanies the busy status. If busy applies to the control unit, busy is accompanied by status modifier.

The figure "Indications of Busy in CSW" lists the situations for devices connected to only one channel when the busy bit is set to one in the CSW and when busy is accompanied by status modifier. For devices shared by more than one channel, operations related to one channel may cause the control unit or device to appear busy to the other channels.

**CSW Status Stored By**

<b>Condition</b>	<b>SIO or SIOF #</b>	<b>TIO</b>	<b>CLRIO+</b>	<b>HIO or HDV</b>	<b>I/O Interruption #</b>
<b>Subchannel available</b>					
DE or attention in device	B, cl	-, cl	*	*	-, cl
Device working, CU available	B	B	*	*	B
<b>CU end or channel end in CU:</b>					
for the addressed device	B, cl	-, cl	-	*	-, cl
for another device	B, SM	B, SM	-	*	-, cl
CU working	B, SM	B, SM	-	*	B, SM
<b>Interruption condition in subchannel for the addressed device because of:</b>					
chaining terminated by busy	*	B, cl	-, cl	*	B, cl
other type of termination	*	-, cl	-, cl	*	-, cl
<b>Subchannel working</b>					
CU available	*	*	-	-	*
CU working	*	*	-	B, SM	*

**Explanation:**

**B** Busy bit in CSW is one.

**cl** Interruption condition cleared; status is placed in CSW.

**CU** Control unit.

**DE** Device end.

**SM** Status-modifier bit appears in CSW.

**\*** CSW not stored, or I/O interruption cannot occur.

**-** Busy bit is zero.

**#** When a channel executes START I/O FAST RELEASE as START I/O, the CSW status stored for the two instructions is identical. When START I/O FAST RELEASE is executed independently of the device, the same status is stored by an I/O interruption with the CSW also indicating deferred condition code 1.

**#** Except when the I/O interruption is caused by a deferred condition code 1 for START I/O FAST RELEASE.

**+** The entries in this column apply only when the CLRIO function is executed. When CLEAR I/O causes the TIO function to be executed, the entries in the TIO column apply.

**Indications of Busy in CSW**

### **Channel End**

Channel end is caused by the completion of the portion of an I/O operation involving transfer of data or control information between the I/O device and the channel. The condition indicates that the subchannel has become available for use for another operation.

Each I/O operation causes channel end to be signaled, and there is only one channel end for an operation. Channel end is not signaled when programming errors or equipment malfunctions are detected during initiation of the operation. When command chaining takes place, only the channel end of the last operation of the chain is made available to the program. Channel end is not made available to the program when a chain of commands is prematurely concluded because of an unusual situation indicated with control-unit end or device end or during the initiation of a chained command.

The instant within an I/O operation when channel end is signaled depends on the operation and the type of device. For operations such as writing on magnetic tape, channel end occurs when the block has been written. On devices that verify the writing, channel end may or may not be delayed until verification is performed, depending on the device. When magnetic tape is being read, channel end occurs when the gap on tape reaches the read-write head. On devices equipped with buffers, channel end occurs upon completion of data transfer between the channel and the buffer. During control operations, channel end is generated when the control information has been transferred to the devices, although for short operations channel end may be delayed until completion of the operation. Operations that do not cause any data to be transferred can provide channel end during the initiation sequence.

Channel end in the control unit causes the control unit to appear busy for the initiation of new operations.

Channel end is presented in combination with status modifier and unit check to initiate the command-retry procedure.

### **Device End**

Device end is caused by the completion of an I/O operation at the device. On some devices, it is also caused by manually changing the device from the not-ready to the ready state. Device end normally indicates that the I/O device has become available for use in another operation.

Each I/O operation causes device end, and there is only one device end to an operation. Device end is not generated when any programming or equipment malfunction is detected during initiation of the operation. When command chaining takes place, only the device end of the last operation of the chain is made available to the program unless an unusual situation is detected during the initiation of a chained command, in which case the chain is concluded without device end.

Device end associated with an I/O operation is generated either simultaneously with channel end or later. For data-transfer operations on devices such as magnetic-tape units, the device concludes the operation at the time channel end is generated, and both device end and channel end occur together. On buffered devices, device end occurs upon completion of the mechanical operation. For control operations, device end is generated at the completion of the operation at the device. The operation may be completed at the time channel end is generated or later.

When command chaining is specified, receipt of the device-end signal, in the absence of any unusual situations, causes the channel to initiate a new I/O operation.

### **Unit Check**

Unit check indicates that the I/O device or control unit has detected an unusual situation that is detailed by the information available to a sense command. Unit check may indicate that a programming or equipment error has been detected, that the not-ready state of the device has affected the execution of the command or instruction, or that an exceptional situation other than the one identified by unit exception has occurred. The unit-check bit provides a summary indication of the sense data.

An error causes the unit-check indication only when it occurs during the execution of a command or TEST I/O, or during some activity associated with an I/O operation. Unless the error pertains to the activity initiated by a command and is of immediate significance to the program, the error does not cause the program to be alerted after device end has been

cleared; a malfunction may, however, cause the device to become not ready.

Unit check is indicated when the existence of the not-ready state precludes a satisfactory execution of the command, or when the command, by its nature, tests the state of the device. When no interruption condition is pending for the addressed device at the control unit, the control unit signals unit check when TEST I/O or the no-operation control command is issued to a not-ready device. In the case of no-operation, the command is rejected, and channel end and device end do not accompany unit check.

Unless the command is designed to cause unit check, such as rewind and unload on magnetic tape, unit check is not indicated if the command is properly executed even though the device has become not ready during or as a result of the operation. Similarly, unit check is not indicated if the command can be executed with the device not ready. Selection of a device that is not ready does not cause a unit check when the sense command is issued or when an interruption condition is pending for the addressed device at the control unit.

If the device detects during the initiation sequence that the command cannot be executed, unit check is signaled to the channel without channel end, control-unit end, or device end. Such unit status indicates that no action has been taken at the device in response to the command. If the situation precluding proper execution of the operation occurs after execution has been started, unit check is accompanied by channel end, control-unit end, or device end, depending on when the situation was detected. Any errors associated with an operation, but detected after device end has been cleared, are indicated by signaling unit check with attention.

Errors, such as invalid command code or invalid command-code parity, do not cause unit check when the device is working or contains an interruption condition at the time of selection. Under these circumstances, the device responds by providing busy status and indicating the interruption condition, if any. The command-code invalidity is not indicated.

Concluding an operation with the unit-check indication causes command chaining to be suppressed.

Unit check is presented in combination with channel end and status modifier to initiate the command-retry procedure.

### **Programming Notes**

1. If a device becomes not ready upon completion of a command, the ending interruption condition can be cleared by TEST I/O without generation of unit check due to the not-ready state, but any subsequent TEST I/O issued to the device causes a unit-check indication.
2. In order that sense indications set in conjunction with unit check are preserved by the device until requested by a sense command, some devices inhibit certain functions until a command other than test I/O or no-operation is received. Furthermore, any command other than sense, test I/O, or no-operation causes the device to reset any sense information. To avoid degradation of the device and its control unit and to avoid inadvertent resetting of the sense information, a sense command should be issued immediately to any device signaling unit check.

### **Unit Exception**

Unit exception is caused when the I/O device detects a situation that usually does not occur. Unit exception includes situations such as recognition of a tape mark and does not necessarily indicate an error. It has only one meaning for any particular command and type of device.

Unit exception can be generated only when the device is executing an I/O operation, or when the device is involved with some activity associated with an I/O operation and the situation is of immediate significance to the program. If the device detects during the initiation sequence that the operation cannot be executed, unit exception is presented to the channel and appears without channel end, control-unit end, or device end. Such unit status indicates that no action has been taken at the device in response to the command. If the situation precluding normal execution of the operation occurs after the execution has been started, unit exception is accompanied by channel end, control-unit end, or device end, depending on when the situation was detected. Any unusual situation associated with an operation, but detected after device end has been cleared, is indicated by signaling unit exception with attention.

A command does not cause unit exception when the device responds with busy status to the command during the initial selection.

Concluding an operation with the unit-exception indication causes command chaining to be suppressed.

## **Channel Status**

The following status bits are generated by the channel. Except for the status bits resulting from equipment malfunction, they can occur only while the subchannel is involved with the execution of an I/O operation.

### **Program-Controlled Interruption**

A program-controlled interruption occurs when the channel fetches a CCW with the program-controlled-interruption (PCI) flag set to one. The I/O interruption due to the PCI flag takes place as soon as possible after the CCW takes control of the operation but may be delayed an unpredictable amount of time because I/O interruptions are disallowed or because of other activity in the system.

The interruption condition due to the PCI flag 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 storage after the assigned storage areas were filled. The extra bytes have not been placed in storage. The count in the CSW is zero.

**Long Block on Output:** During a write or control operation, the device requested one or more bytes from the channel after the assigned storage areas were exhausted. The count in the CSW is zero.

**Short Block on Input:** The number of bytes transferred during a read, read-backward, or sense operation is insufficient to fill the storage areas assigned to the operation. The count in the CSW is not zero.

**Short Block on Output:** The device terminated a write or control operation before all information contained in the assigned storage areas was transferred to the device. The count in the CSW is not zero.

Incorrect length is not indicated when the current CCW has the SLI flag set to one and the CD flag set to zero. The indication does not occur for immediate

operations and for operations rejected during the initiation sequence.

When incorrect length occurs, command chaining is suppressed, unless the SLI flag in the CCW is one or unless the operation is immediate. See the figure "Channel-Chaining Action" in this chapter for the effect of the CD, CC, and SLI flags on the indication of incorrect length.

### **Programming Note**

The setting of incorrect length is unpredictable in the CSW stored during CLEAR I/O.

### **Program Check**

Program check occurs when programming errors are detected by the channel. Program check can be due to the following causes:

**Invalid CCW-Address Specification:** The CAW or the transfer-in-channel command does not designate the CCW on integral boundaries for doublewords. The three rightmost bits of the CCW address are not zeros.

**CCW Location Not Provided:** The channel has attempted to fetch a CCW from a storage location that is not provided. This may occur because the program has specified in the CAW or in the transfer-in-channel command a page address (bits 8-20) equal to or greater than the page-capacity count (PCC), or because on chaining the channel has attempted to fetch a CCW from a page with a page address equal to PCC.

**Invalid Command Code:** The command code in the first CCW designated by the CAW or in a CCW fetched on command chaining has four low-order zeros. The command code is not tested for validity during data chaining.

**Invalid Count:** A CCW other than a CCW specifying transfer in channel contains the value zero in bit positions 48-63.

**Data Location Not Provided:** The channel has attempted to transfer data to or from a storage location that is not provided. This may occur because the program has specified in the CCW a page address (bits 8-20) equal to or greater than the page-capacity count (PCC) or because the channel attempts during data transfer to access a page with a page address equal to PCC.

**Invalid CAW Format:** The CAW does not contain zeros in bit positions 4-7.

**Invalid CCW Format:** A CCW other than a CCW specifying transfer in channel does not contain zeros in bit positions 37-39.

**Invalid Sequence:** The first CCW designated by the CAW specifies transfer in channel, or the channel has fetched two successive CCWs both of which specify transfer in channel.

Detection of program check during the initiation of an operation causes execution of the operation to be suppressed. When program check is detected after the device has been started, the device is signaled to conclude the operation the next time it requests or offers a byte of data. Program check causes command chaining to be suppressed.

### **Protection Check**

Protection check occurs when the channel attempts a storage access that is prohibited by key-controlled storage protection. Protection applies to the fetching of CCWs and output data, and to the storing of input data. Storage accesses associated with each I/O operation are performed using the subchannel key provided in the CAW associated with that operation. For details, see the section "Key-Controlled Protection" in Chapter 3, "Storage."

Protection check also occurs when it is detected that the channel has attempted to access a CCW or data from a page that is in the disconnected state. For details, see the section "Page States" in Chapter 3, "Storage."

When protection check occurs during the fetching of a CCW that specifies the initiation of an I/O operation, the operation is not initiated. When protection check is detected after the device has been started, the device is signaled to conclude the operation the next time it requests or offers a byte of data. Protection check causes command chaining to be suppressed.

### **Channel-Data Check**

Channel-data check indicates that a machine error has been detected in the information transferred to or from storage during an I/O operation, or that a parity error has been detected on the data on bus-in during an input operation. This information includes the data read or written, as well as the information transferred as data during a sense or control operation. The error may have been detected in the

channel, in storage, or on the path between the two. Channel-data check may be indicated for data with an invalid checking-block code in storage when the data is referred to by the channel but the data does not participate in the operation.

Whenever a parity error on I/O input data is indicated by means of channel-data check, the channel forces correct parity on all data received from the I/O device, and all data placed in storage has valid checking-block code. When, on an input operation, the channel attempts to store less than a complete checking block, and invalid checking-block code is detected on the checking block in storage, the contents of the location remain unchanged with invalid checking-block code. On an output operation, whenever a channel-data check is indicated, all bytes that came from a checking block with invalid checking-block code have been transmitted with parity errors.

Channel-data check causes command chaining to be suppressed but does not affect the execution of the current operation. Data transfer proceeds to normal completion, if possible, and an interruption condition is generated when the device presents channel end. A logout may be performed, depending on the channel. Accordingly, the detection of the error may affect the state of the channel and the device.

### **Channel-Control Check**

Channel-control check is caused by machine malfunction affecting channel controls. It may be caused by invalid checking-block code on CCW and data addresses and invalid checking-block code on the contents of the CCW. Channel-control check may also include those channel-detected errors associated with data transfer that are not indicated as channel-data check, as well as those I/O interface errors detected by the channel that are not indicated as interface-control check. Errors responsible for channel-control check may cause the contents of the CSW to be invalid and conflicting. The CSW as generated by the channel has valid checking-block code.

Detection of channel-control check causes the current operation, if any, to be immediately concluded.

Channel-control check is set whenever CSW bit 5, logout pending, is set to one.

In some situations, machine malfunctions affecting channel control may instead be reported as an external-damage or system-damage machine-check condition.

### **Interface-Control Check**

Interface-control check indicates that an invalid signal has been received by the channel when communicating with a control unit or device. It is detected by the channel and usually indicates malfunctioning of an I/O device. It can be due to the following:

1. The address or status byte received from a device has invalid parity.
2. A device responded with an address other than the address specified by the channel during initiation of an operation.
3. During command chaining the device appeared not operational.
4. A signal from a device occurred at an invalid time or had invalid duration.
5. A device signaled I/O error alert.

Detection of interface-control check causes the current operation, if any, to be immediately concluded.

### **Chaining Check**

Chaining check is caused by channel overrun during data chaining on input operations. Chaining check occurs when the I/O data rate is too high to be handled by the channel and by storage under current conditions. Chaining check cannot occur on output operations.

Chaining check causes the I/O device to be signaled to conclude the operation. It causes command chaining to be suppressed.

### ***Contents Of Channel-Status Word***

The contents of the CSW depend on the reason the CSW was stored and on the programming method by which the information is obtained. The status portion always identifies the reason the CSW was stored. The subchannel-key, CCW-address, and count fields may contain information pertaining to the last operation or may be set to zero, or the original contents of these fields at location 64 may be left unchanged.

### **Information Provided by Channel-Status Word**

Interruption conditions resulting from the execution or conclusion of an operation at the subchannel cause the whole CSW to be replaced. Such a CSW can be stored only by an I/O interruption or by TEST I/O or CLEAR I/O. Except for situations associated with command chaining and equipment malfunctioning, the storing can be caused by PCI or

channel end and by the execution of HALT I/O or HALT DEVICE on the selector channel. The contents of the CSW are related to the current values of the corresponding quantities, although the count is unpredictable after program check, protection check, and chaining check, and after an interruption due to the PCI flag.

A CSW stored upon the execution of a chain of operations pertains to the last operation which the channel executed or attempted to initiate. Information concerning the preceding operations is not preserved and is not made available to the program.

When an unusual situation causes command chaining to be suppressed, the premature conclusion of the chain is not explicitly indicated in the CSW. A CSW associated with a conclusion due to a situation occurring at channel-end time contains channel end and identifies the unusual situation. When the device signals the unusual situation with control-unit end or device end, the channel-end indication is not made available to the program, and the channel provides the current subchannel key, CCW address, and count, as well as the unusual indication, with control-unit end or device end in the CSW. The CCW-address and count fields pertain to the operation that was executed.

When the execution of a chain of commands is concluded by an unusual situation detected during initiation of a new operation, the CCW-address and count fields pertain to the rejected command. Except for situations resulting from equipment malfunctioning, conclusion at initiation time can occur because of attention, unit check, unit exception, or program check, and causes both the channel-end and device-end bits in the CSW to be set to zeros.

A CSW associated with status signaled after the operation at the subchannel has been concluded contains zeros in the subchannel-key, CCW-address, and count fields, provided the status is not cleared during START I/O or START I/O FAST RELEASE and provided logout pending is not indicated. This status includes attention, control-unit end, and device end (and channel end when it occurs after the conclusion of an operation on the selector channel by HALT I/O or HALT DEVICE).

When the above status indications, other than logout pending, are cleared during START I/O or START I/O FAST RELEASE, only the status portion of the CSW is stored, and the original contents of the subchannel-key, CCW-address,

deferred-condition-code, logout-pending, and count fields in location 64 are preserved. Similarly, only the status bits of the CSW are changed when the command is rejected or the operation at the subchannel is concluded during the execution of START I/O or START I/O FAST RELEASE or whenever HALT I/O or HALT DEVICE causes CSW status to be stored.

Errors detected during execution of the I/O operation do not affect the validity of the CSW unless channel-control check or interface-control check are indicated. Channel-control check indicates that equipment errors have been detected which can cause any part of the CSW, as well as the I/O address, to be invalid. Interface-control check indicates that the address identifying the device or the status bits received from the device may be invalid. The channel forces correct parity on invalid CSW fields. The validity of these fields can be ascertained by inspecting the limited channel logout.

When any I/O instruction cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones.

### **Subchannel Key**

A CSW stored to reflect the progress of an operation at the subchannel contains the subchannel key used in that operation. The contents of this field are not affected by programming errors detected by the channel or by the situations causing termination of the operation.

### **CCW Address**

When the CSW is formed to reflect the progress of the I/O operation at the subchannel, the CCW address is normally 8 higher than the address of the last CCW used in the operation.

The figure "Contents of the CCW-Address Field in the CSW" lists the contents of the CCW-address field for all situations that can cause the CSW to be stored. They are listed in order of priority; that is, if two situations occur, the CSW appears as indicated for the situation higher on the list. When a CSW has been stored and the situation exists that a command-retry request has been recognized but the CCW has not been re-executed, the "last-used CCW + 8" is the CCW that is to be retried. When a program check is caused by two CCWs in sequence both of which specify transfer in channel, the second CCW is the one considered invalid. In the figure, the three cases of disconnected location and the two cases of invalid key are all protection checks.

## Situations

Channel-control check  
Status stored by START I/O or START I/O FAST RELEASE  
Status stored by HALT I/O or HALT DEVICE  
Program check because CCW location in TIC not provided  
Program check (all others)  
Disconnected CCW location in TIC  
Disconnected CCW location generated  
Disconnected data location  
Invalid key on CCW fetch  
Invalid key on data access  
Chaining check  
Termination under count control  
Termination by I/O device  
Termination by HALT I/O  
Termination by CLEAR I/O  
Suppression of command chaining due to unit check  
or unit exception with device end or control-unit end  
Termination on command chaining by busy, unit check,  
or unit exception  
Deferred condition code 1 for START I/O FAST RELEASE  
PCI flag in CCW  
Interface-control check  
Channel end after HALT I/O on selector channel  
Channel end after CLEAR I/O  
Control-unit end  
Device end  
Attention  
Busy  
Status modifier

## Contents of Field

Unpredictable  
Unchanged  
Unchanged  
Address of TIC + 8  
Address of invalid CCW + 8  
Address of TIC + 8  
First invalid CCW address + 8  
Address of invalid CCW + 8  
Address of protected CCW + 8  
Address of current CCW + 8  
Address of last-used CCW + 8  
Address of last CCW used in the completed operation + 8  
  
Address of CCW specifying the new operation + 8  
  
Address of CCW specifying the new operation + 8  
Address of last-used CCW + 8  
Unpredictable  
Zero  
Zero  
Zero  
Zero  
Zero  
Zero  
Zero  
Zero

## Contents of the CCW-Address Field in the CSW

### Count

The residual count, in conjunction with the original count specified in the last CCW used, indicates the number of bytes transferred to or from the area designated by the CCW. When an input operation is concluded, the difference between the original count in the CCW and the residual count in the CSW is equal to the number of bytes transferred to storage;

on an output operation, the difference is equal to the number of bytes transferred to the I/O device.

The figure "Contents of the Count Field in the CSW" lists the contents of the count field for all situations that can cause the CSW to be stored. They are listed in the order of priority; that is, if two situations occur, the CSW appears as for the situation higher on the list.

**Situations****Contents of Field**

<b>Situations</b>	<b>Contents of Field</b>
Channel-control check	Unpredictable
Status stored by START I/O or START I/O FAST RELEASE	Unchanged
Status stored by HALT I/O or HALT DEVICE	Unchanged
Program check	Unpredictable
Protection check	Unpredictable
Chaining check	Unpredictable
Termination under count control	Correct
Termination by I/O device	Correct
Termination by HALT I/O or HALT DEVICE	Unpredictable
Termination by CLEAR I/O	Unpredictable
Suppression of command chaining due to unit check or unit exception with device end or control-unit end	Correct. Residual count of last CCW used in the completed operation.
Termination on command chaining by busy, unit check, or unit exception	Correct. Original count of CCW specifying the new operation.
Deferred condition code 1 or 3 for START I/O FAST RELEASE	Correct. Original count of CCW specifying the new operation.
PCI flag in CCW	Unpredictable
Interface-control check	Unpredictable
Channel end after HALT I/O on selector channel	Zero
Channel end after CLEAR I/O	Zero
Control-unit end	Zero
Device end	Zero
Attention	Zero
Busy	Zero
Status modifier	Zero

**Contents of the Count Field in the CSW****Status**

The status bits identify the situations that have been detected during the I/O operation, that have caused a command to be rejected, or that have been generated by external events.

When the channel detects several errors, all corresponding status bits in the CSW may be set to ones or only one may be set, depending on the error and model. Errors associated with equipment malfunctioning have precedence, and whenever malfunctioning causes an operation to be terminated, channel-control check, interface-control check, or channel-data check is indicated, depending on the error. When an operation is concluded by program check, protection check, or chaining check, the channel identifies the situation responsible for the conclusion and may or may not indicate incorrect length. When a data error has been detected and the operation is concluded prematurely because of a program check, protection check, or chaining check,

both data check and the programming error are identified.

If the CCW fetched on command chaining has the PCI flag set to one but a programming error in the contents of the CCW precludes the initiation of the operation, whether the PCI bit is one in the CSW associated with the interruption condition is unpredictable. Similarly, if a programming error in the contents of the CCW causes the command to be rejected during execution of START I/O or START I/O FAST RELEASE, the CSW stored by the instruction may or may not have the PCI bit set to one. Furthermore, when the channel detects a programming error in the CAW or in the first CCW, the PCI bit is unpredictable in a CSW stored by START I/O or START I/O FAST RELEASE when the PCI flag is zero in the first CCW associated with the instruction.

However, if the CCW fetched on command chaining has the PCI flag set to one but an unusual situation detected by the device precludes the

initiation of the operation, the PCI bit is one in the CSW associated with the interruption condition. Likewise, if device status causes the command to be rejected during execution of START I/O or START I/O FAST RELEASE, the CSW stored by the instruction contains the PCI bit set to one.

Situations detected by the channel are not related to those identified by the I/O device.

The figure "Contents of the CSW Status Fields" summarizes the handling of status bits. The figure lists the states and activities that can cause status indications to be created and the methods by which these indications can be placed in the CSW.

Status	When I/O Is Idle	When Subchannel Is Working	Upon Termination of Operation at			During Command Chaining	By SIO or SIOF	By TIO	By CLRIO+	By HIO or HDV	By I/O Interruption
			Sub-channel	Control Unit	I/O Device						
Attention	C*				C*	S	S	S			S
Status modifier					C	CS	CS	S	CS		S
Control-unit end				C*		CS	CS	S	CS		S
Busy						C	CS	CS	S	CS	S
Channel end			C*	C*H		C*#	CS#	S	S		S
Device end	C*				C*	C#	CS#	S	S		S
Unit check	C		C	C	C	C*	CS	CS	S		CS
Unit exception			C	C	C	C*	CS	S	S		S
Program-controlled interruption		C*	C*			C	CS	S	S		S
Incorrect length		C	C					S	S		S
Program check		C	C			C*	CS	S	S		S
Protection check		C	C			C*	CS	S	S		S
Channel-data check		C	C					S	S		S
Channel-control check	C*	C*	C*	C*	C*	C*	CS	CS	CS	CS	CS
Interface-control check	C*	C*	C*	C*	C*	C*	CS	CS	CS	CS	CS
Chaining check		C	C					S	S		S
Deferred cond code 1							C*#	S	S		S
Deferred cond code 3							C*#	S	S		S

**Explanation:**

**C** The channel or device can create or present status at the indicated time. A CSW or its status portion is not necessarily stored at this time.

Status such as channel end or device end is created at the indicated time. Other status bits may have been created previously but are made accessible to the program only at the indicated time. Examples of such status bits are program check and channel-data check, which are detected while data is transferred but are made available to the program only with channel end, unless the PCI flag or an equipment malfunction has caused an interruption condition to be generated earlier.

**S** The status indication is stored in the CSW at the indicated time.

An *S* appearing alone indicates that the status has been created previously. The letter *C* appearing with the *S* indicates that the status did not necessarily exist previously in the form that causes the program to be alerted, and may have been created by the I/O instruction or I/O interruption. For example, an equipment malfunction may be detected during an I/O interruption, causing channel-control or interface-control check to be indicated; or a device such as the 2702 may signal control-unit busy in response to interrogation by an I/O instruction, causing status modifier, busy, and control-unit end to be indicated in the CSW.

**\*** The status generates an interruption condition.

Channel end and device end do not result in interruption conditions when command chaining is specified and no unusual situations have been detected.

**#** This indication is created at the indicated time only by an immediate operation.

**#** Applies only to SIOF.

**H** When an operation on the selector channel has been concluded by HALT DEVICE or HALT I/O, or an operation has been concluded by CLEAR I/O, channel end indicates the conclusion of the data-handling portion of the operation at the control unit.

**+** The entries in this column apply only when the CLRIO function is executed. When CLEAR I/O causes the TIO function to be executed, the entries in the TIO column apply.

**Contents of the CSW Status Fields**

## Channel Logout

When a channel stores a CSW that indicates channel-control check in the absence of logout pending, or interface-control check, or, on some channels, channel-data check, a limited channel logout accompanies the storing of the CSW. Such a logout is useful for error recovery.

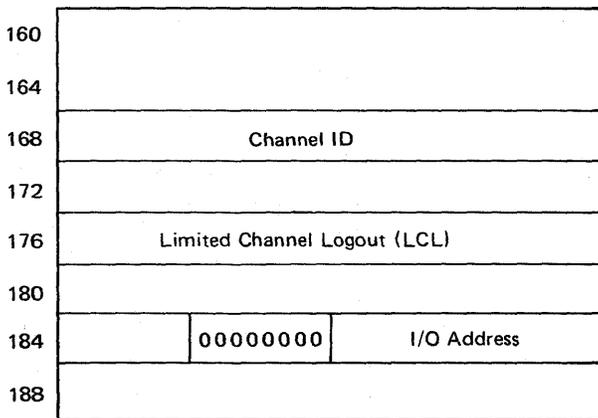
The limited channel logout contains model-independent information and is stored at locations 176-179. When it is stored, bit 0 of the logout is always stored as a zero.

## I/O Communications Area

Storage locations 160-191 comprise a permanently assigned area of storage used for I/O, designated the I/O communications area (IOCA). (See the figure "I/O Communications Area.")

Locations 160-167, 172-175, 180-184, and 188-191 are reserved for future I/O use.

**Channel ID (Locations 168-171):** Locations 168-171, when stored during the execution of a STORE CHANNEL ID instruction, contain information which describes the addressed channel.



I/O Communications Area

**Limited Channel Logout (Locations 176-179):** The limited-channel-logout field (locations 176-179) contains model-independent information related to equipment errors detected by the channel. This information is used to provide detailed machine status when errors have affected I/O operations. The field may be stored only when the CSW or a portion of the CSW is stored.

The bits of the field are defined as follows:

0 This bit is always stored as a zero when a limited channel logout (LCL) is stored. If the program ensures that this bit is set to one and any channel-control check, interface-control check, or channel-data check occurs, a test of this bit can determine if the LCL was stored by the channel. The LCL cannot be stored by a channel unless one of these three channel-status bits is set to one.

1-3 **Identity of the storage-control unit (SCU)** identifies the SCU through which storage references were directed when an error was detected. This identity is not necessarily the identity of the storage unit involved with the transfer. When only one physical path exists between channel and storage, the storage-control unit has the identity of the CPU. If more than one path exists, the storage-control unit has its own identity.

When bit 3 is zero, bits 1 and 2 are undefined. In this case, the SCU identity is implied to be the same as the CPU identity. When bit 3 is one, the binary value of bits 1 and 2 identifies a physical SCU. Each SCU in the system has a unique identity.

4-7 **Detect field** identifies the type of unit that detected the error. At least one bit is present in this field, and multiple bits may be set when more than one unit detects the error.

- Bit 4 – CPU
- Bit 5 – Channel
- Bit 6 – Main-storage control
- Bit 7 – Main storage

8-12 **Source field** indicates the most likely source of the error. The determination is made by the channel on the basis of the type of error check, the location of the checking station, the information flow path, and the success or failure of transmission through previous check stations.

Normally, only one bit will be present in this field. However, when interunit communication cannot be resolved to a single unit, such as when the interface between units is at fault, multiple bits (normally two) may be set to ones in this field. When a reasonable determination cannot be made, all bits in this field are set to zeros.

If the detect and source fields indicate different units, the interface between them can also be considered suspect.

- Bit 8 – CPU
- Bit 9 – Channel
- Bit 10 – Main-storage control
- Bit 11 – Main storage
- Bit 12 – Control unit
- 13-18 *Reserved.* Stored zero.
- 19-23 *Field-validity flags.* These bits indicate the validity of the information stored in the designated fields. When the validity bit is set to one, the field is stored and usable. When the validity bit is set to zero, the field is not usable.
  - The fields designated are:
    - Bit 19 – Sequence code
    - Bit 20 – Unit status
    - Bit 21 – CCW address and subchannel key in CSW
    - Bit 22 – Channel address
    - Bit 23 – Device address
- 24-25 *Type of termination* that has occurred is indicated by these two bits.
  - This encoded field has meaning only when a channel-control check or an interface-control check is indicated in the CSW. When neither of these two checks is indicated, no termination has been forced by the channel.
  - 00 Interface disconnect
  - 01 Stop, stack, or normal termination
  - 10 Selective reset
  - 11 Reserved
- 26-27 *Reserved.* Stored zero.
- 28 *I/O error alert.* This bit, when set to one, indicates that the limited channel logout resulted from the signaling of I/O error alert by the indicated unit. The I/O-error-alert signal indicates that the control unit has detected a malfunction which prevents it from communicating properly with the channel. The channel, in response, performs a malfunction reset and causes interface-control check to be set.
- 29-31 *Sequence code* identifies the I/O sequence in progress at the time of error. It is meaningless if stored during the execution of HALT I/O or HALT DEVICE.
  - For all cases, the CCW address in the CSW, if validly stored and nonzero, is the address of the current CCW plus 8.
  - The sequence code assignments are:
    - 000 A channel-detected error occurred during the execution of a TEST I/O or CLEAR I/O instruction.
    - 001 Command-out with a nonzero command byte on bus-out has been sent by the channel, but device status has not yet been analyzed by the channel. This code is set with a command-out response to address-in during initial selection.
    - 010 The command has been accepted by the device, but no data has been transferred. This code is set by a service-out or command-out response to status-in during an initial selection sequence, if the status is either channel end alone, or channel end and device end, or channel end, device end, and status modifier, or all zeros.
    - 011 At least one byte of data has been transferred between the channel and the device. This code is set with a service-out response to service-in and, when appropriate, may be used when the channel is in an idle or polling state.
    - 100 The command in the current CCW has either not yet been sent to the device or else was sent but not accepted by the device. This code is set when one of the following situations occurs:
      1. When the CCW address is updated during command chaining or a START I/O.
      2. When service-out or command-out is raised in response to status-in during an initial selection sequence with the status on bus-in including attention, control-unit end, unit check, unit exception, busy, status modifier (without channel end and device end), or device end (without channel end).
      3. When a short, control-unit-busy sequence is signaled.
      4. When command retry is signaled.
      5. When the channel issues a test-I/C command rather than the command in the current CCW.
    - 101 The command has been accepted, but data transfer is unpredictable. This code applies from the time a device

comes on the interface until the time it is determined that a new sequence code applies. It may thus be used when a channel goes into 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 cannot distinguish between code 010 or 011.

110 *Reserved.*

111 *Reserved.*

***Reserved (Location 185):*** Zero is stored at location 185 whenever an I/O address is stored at locations 186-187.

***I/O Address (Locations 186-187):*** A two-byte field is provided for storing the I/O address on each I/O interruption in the EC mode.

## Chapter 13. Operator Facilities

### Contents

Basic Operator Facilities	13-1	Power Controls	13-4
Address-Compare Controls	13-1	Rate Control	13-4
Alter-and-Display Controls	13-2	Restart Key	13-4
Check Control	13-2	Save Indicator	13-4
Check-Stop Indicator	13-2	Start Key	13-4
IML Controls	13-2	Stop Key	13-4
Interrupt Key	13-2	Storage-Size Control	13-4
Interval-Timer Control	13-3	System Indicator	13-5
Load Indicator	13-3	System-Reset-Clear Key	13-5
Load-Clear Key	13-3	System-Reset-Normal Key	13-5
Load-Normal Key	13-3	Test Indicator	13-5
Load-Unit-Address Controls	13-3	TOD-Clock Control	13-5
Machine-Save Key	13-3	Wait Indicator	13-5
Manual Indicator	13-3		
Mode Indicator	13-3		

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 time-of-day clock, initial program loading, resets, and other manual controls for operator intervention in normal machine operation.

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 control settings are contained in the appropriate System Library (SL) 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 a particular model 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 settings to specify the action, if any, to be taken when the address match occurs. The two settings are normal and stop. When this control is set to stop, the test indicator is turned on.

The normal setting disables the address-compare operation.

The stop setting causes the CPU to enter the stopped state on an address match. 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 settings, as well as others:

The any setting causes the address comparison to be performed on all storage references.

The data-store setting causes address comparison to be performed when storage is addressed to store data.

The I/O setting causes address comparison to be performed when storage is addressed by a channel to transfer data or to fetch a channel-command word. Whether references to the channel-address word or the channel-status word cause a match to be indicated depends on the model.

The IC setting 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. It depends on the model whether a match is indicated when fetching the target instruction of EXECUTE.

### ***Alter-and-Display Controls***

The operator facilities provide controls and procedures to permit the operator to alter and display the contents of addressable locations in main storage, the storage keys, the page bits, the general, floating-point, and control registers, and the PSW. Information in storage can only be altered or displayed if the storage pages containing the information are in the connected or addressable state.

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.

### ***Check Control***

The check control has at least two settings, stop and normal. If the control is set to stop, the CPU enters the check-stop state when either

1. A machine-check condition is detected and not corrected
2. A channel check occurs which would cause information to be stored in a channel-logout area at locations 176-179

Whether information is actually stored in assigned storage locations as a result of the machine check or

channel check, the indications given for the cause of the stoppage, and the manner of resuming CPU operation depend on the model.

If the check control is set to normal, the action resulting from the detection of a machine check or channel check is the same as described in Chapter 11, "Machine-Check Handling," or in Chapter 12, "Input/Output Operations," respectively.

The test indicator is on while the check control is set to stop.

### **Programming Note**

Except that recovery from a machine check or a channel check with logout is not possible, the check control permits a System/360 program, which uses assigned storage locations above 128 as ordinary storage, to be run in the BC 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 perform initial microprogram loading (IML). The IML operation selects the ECPS:VSE mode or the System/370 mode of operation.

When the IML operation is completed, the state of the affected CPU, channels, storage, and operator facilities is the same as if a power-on reset had been performed, except that the value and state of the time-of-day clock are not reset.

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 the section "Interrupt Key" in Chapter 6, "Interruptions.")

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.

### ***Interval-Timer Control***

The interval-timer control disables or enables operation of the interval timer. Disabling the interval timer does not affect any other facility.

When the control is set to disable the interval timer, updating of assigned storage locations 80-83 ceases. The contents of locations 80-83 remain at the last value to which they were updated, unless changed by a subsequent store operation. Any already pending interval-timer-interruption condition is kept pending without regard to the state of the external mask, PSW bit 7, and the interval-timer mask, bit 24 of control register 0.

When the control is set to enable the interval timer, updating of locations 80-83 is resumed using the current contents. If an interval-timer-interruption request existed and was kept pending when the interval-timer control was last set to disable, that condition remains pending until the CPU is enabled for the interruption.

The test indicator may or may not be turned on when the interval-timer control is set to disable.

#### **Programming Note**

Disabling the interval timer allows execution of a program which uses locations 80-83 as ordinary storage. A program which does not use the interval timer will function correctly with the interval timer disabled, even when the interval timer fails.

### ***Load Indicator***

The load indicator is on during initial program loading, indicating that the CPU is in the load state. The indicator goes on when the load-clear or load-normal key is activated and the corresponding operation is started. It goes off after the new PSW is loaded successfully.

### ***Load-Clear Key***

Activating the load-clear key causes a clear-reset operation to be performed and initial program loading to be started using the I/O device specified by the load-unit-address controls. For details, see the sections "Resets" and "Initial Program Loading" in Chapter 4, "Control."

The load-clear 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 an initial-program-reset operation to be performed and initial program loading to be started using the I/O device specified by the load-unit-address controls. For details, see the sections "Resets" and "Initial Program Loading" in Chapter 4, "Control."

The load-normal 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 select three hexadecimal digits, which provide the 12 rightmost I/O address bits used for initial program loading.

### ***Machine-Save Key***

Activating the machine-save key initiates a machine-save operation. (See the section "Machine Save" in Chapter 4, "Control.") The save indicator is turned on when the operation is completed successfully.

The machine-save key is effective only when the CPU is in the stopped state.

#### **Operation Note**

The machine-save 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 machine-save key
3. Activation of the load-normal key to enter a standalone dump program

The system-reset-normal key must be activated in step 1 when the stop key is not effective because a continuous string of interruptions occurs or the CPU is in the check-stop state.

### ***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.

### ***Mode Indicator***

The mode indicator shows the architectural mode of operation selected by the last IML operation.

## ***Power Controls***

The power controls are used to turn the power on and off.

The CPU, storage, channels, 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. A power-on reset is then initiated (see the section "Resets" in Chapter 4, "Control"). It depends on the model whether the architectural mode of operation can be selected when the power is turned on, or whether the IML controls have to be used to change the mode after the power is on.

## ***Rate Control***

The setting of the rate control determines the effect of the start function and the manner in which instructions are executed.

The rate control has at least two settings. The normal setting is process. When the rate control is set to process and the start function is performed, the CPU starts operating at normal speed. When the rate control is set to instruction step, one instruction or, for interruptible instructions, one unit of operation is executed each time that the start function is performed. For details, see the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control."

The test indicator is on while the rate control is not set to process.

If the setting of the rate control is changed while the CPU is in the operating or load state, the results are unpredictable.

## ***Restart Key***

Activating the restart key initiates a restart interruption. (See the section "Restart Interruption" in Chapter 6, "Interruptions.")

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 the CPU is in the load state.

## ***Save Indicator***

The save indicator is turned on when a machine-save operation has been successfully completed. It is turned off when the load-clear, load-normal, restart, start, system-reset-clear, or system-reset-normal key is activated. It may also be turned off when other controls are activated. The indicator is off after a power-on reset. If an error is encountered during the machine-save operation, the indicator remains off.

## ***Start Key***

Activating the start key causes the CPU to perform the start function. (See the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control.")

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 the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control.")

The stop key is effective only when the CPU is in the operating state.

## ***Operation Note***

Activating the stop key has no effect when a continuous string of interruptions occurs or when the CPU is in the check-stop state.

## ***Storage-Size Control***

The storage-size control is provided when a model permits more than one size of virtual storage. The control determines the storage size and, hence, the value of the page-capacity count. The number of storage-size settings of the control depends on the model. (See the section "Storage Size" in Chapter 3, "Storage.")

A new setting of the storage-size control becomes effective only as part of the IML operation performed when turning the power on or when activating the IML controls.

### **System Indicator**

The system indicator is on when the customer or customer-engineer usage meter of the central processing complex is running.

In general, the system indicator is on when the CPU is not in the wait state and not in the stopped or check-stop state, or when the I/O system is working, or both.

### **System-Reset-Clear Key**

Activating the system-reset-clear key causes a clear-reset operation to be performed. For details, see the section "Resets" in Chapter 4, "Control."

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 program-reset operation to be performed. For details, see the section "Resets" in Chapter 4, "Control."

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 or the check control to stop or setting the rate control to instruction step turns on the test indicator. Setting the interval-timer control to disable may or may not turn 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.

### **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 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 manual.

### **TOD-Clock Control**

When the TOD-clock control is not activated, that is, the control is set to secure, the value of the time-of-day (TOD) clock is protected against unauthorized or inadvertent change by not permitting the instruction SET CLOCK to change the value.

When the TOD-clock control is activated, that is, the control is set to enable set, alteration of the clock value by means of SET CLOCK is permitted. This setting is temporary, and the control automatically returns to secure.

### **Wait Indicator**

The wait indicator is on when the CPU is in the wait state.

### **Operation Note**

The manual indicator, system indicator, and wait indicator may be used by the operator to determine the status of the system. The following figure shows the possible conditions when power is on and the CPU is not in the load or check-stop state.

<u>Manual Indicator</u>	<u>System Indicator<sup>1</sup></u>	<u>Wait Indicator</u>	<u>CPU State</u>	<u>State of I/O System<sup>2</sup></u>
off	off	off	*	*
off	off	on	Operating, Wait	Not Working
off	on	off	Operating	Undetermined
off	on	on	Operating, Wait	Working
on	off	off	Stopped	Not Working
on	off	on	Stopped, Wait	Not Working
on	on	off	Stopped	Working
on	on	on	Stopped, Wait	Working

#### **Explanation:**

\* Abnormal condition.

<sup>1</sup> When the system indicator is turned on, it remains on for a minimum of approximately 1 second.

<sup>2</sup> The operation of the console I/O device is included here as an I/O operation.

### **System-Status Indications**

# Appendix A. Number Representation and Instruction-Use Examples

## Contents

Number Representation	A-2	MOVE NUMERICS (MVN)	A-18
Binary Integers	A-2	MOVE WITH OFFSET (MVO)	A-19
Signed Binary Integers	A-2	MOVE ZONES (MVZ)	A-19
Unsigned Binary Integers	A-3	MULTIPLY (M, MR)	A-20
Decimal Integers	A-3	MULTIPLY HALFWORD (MH)	A-20
Floating-Point Numbers	A-4	OR (O, OR, OI, OC)	A-21
Conversion Example	A-5	Or (OI)	A-21
Instruction-Use Examples	A-5	PACK (PACK)	A-21
Machine Format	A-5	SHIFT LEFT DOUBLE (SLDA)	A-21
Assembler-Language Format	A-5	SHIFT LEFT SINGLE (SLA)	A-22
General Instructions	A-6	STORE CHARACTERS UNDER MASK (STCM)	A-22
ADD HALFWORD (AH)	A-6	STORE MULTIPLE (STM)	A-23
AND (N, NR, NI, NC)	A-6	TEST UNDER MASK (TM)	A-23
AND (NI)	A-6	TRANSLATE (TR)	A-23
BRANCH AND LINK (BAL, BALR)	A-7	TRANSLATE AND TEST (TRT)	A-24
BRANCH ON CONDITION (BC, BCR)	A-7	UNPACK (UNPK)	A-25
BRANCH ON COUNT (BCT, BCTR)	A-8	Decimal Instructions	A-25
BRANCH ON INDEX HIGH (BXH)	A-8	ADD DECIMAL (AP)	A-26
BRANCH ON INDEX LOW OR EQUAL (BXLE)	A-9	COMPARE DECIMAL (CP)	A-26
COMPARE HALFWORD (CH)	A-9	DIVIDE DECIMAL (DP)	A-26
COMPARE LOGICAL (CL, CLR, CLI, CLC)	A-9	EDIT (ED)	A-27
Compare Logical (CLR)	A-9	EDIT AND MARK (EDMK)	A-28
Compare Logical (CLI)	A-9	MULTIPLY DECIMAL (MP)	A-28
Compare Logical (CLC)	A-10	SHIFT AND ROUND DECIMAL (SRP)	A-29
COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)	A-10	Decimal Left Shift	A-29
COMPARE LOGICAL LONG (CLCL)	A-11	Decimal Right Shift	A-29
CONVERT TO BINARY (CVB)	A-12	Decimal Right Shift and Round	A-29
CONVERT TO DECIMAL (CVD)	A-12	Multiplying by a Variable Power of 10	A-30
DIVIDE (D, DR)	A-13	ZERO AND ADD (ZAP)	A-30
EXCLUSIVE OR (X, XR, XI, XC)	A-13	Floating-Point Instructions	A-30
Exclusive Or (XI)	A-13	ADD NORMALIZED (AE, AER, AD, ADR)	A-30
Exclusive Or (XC)	A-13	ADD UNNORMALIZED (AU, AUR, AW, AWR)	A-31
EXECUTE (EX)	A-14	COMPARE (CE, CER, CD, CDR)	A-31
INSERT CHARACTERS UNDER MASK (ICM)	A-15	Multiprogramming and Multiprocessing Examples	A-32
LOAD (L, LR)	A-16	Example of a Program Failure Using OR Immediate	A-32
LOAD ADDRESS (LA)	A-16	COMPARE AND SWAP (CS, CDS)	A-32
LOAD HALFWORD (LH)	A-16	Setting a Single Bit	A-33
MOVE (MVI)	A-17	Updating Counters	A-33
MOVE (MVC)	A-17		
MOVE LONG (MVCL)	A-18		

# Number Representation

## Binary Integers

### Signed Binary Integers

Signed binary integers are most commonly represented as halfwords (16 bits) or fullwords (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 fullwords) are used to designate the magnitude of the number. Binary integers are also referred to as fixed-point numbers, because the radix 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:

S		
	0	000 0000 0001 1010 (S is the sign bit)
Invert	1	111 1111 1110 0101
Add	1	1
	-26	1 111 1111 1110 0110 (Two's complement form)

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 high-order position and carry out.

3.  $+35 = 0010\ 0011$   
 $-57 = 1100\ 0111$   
 $-22 = 1110\ 1010$

Sign change only—no carry into high-order position and no carry out.

4.  $-57 = 1100\ 0111$   
 $-35 = 1101\ 1101$   
 $-92 = 1010\ 0100$

No overflow—carry into high-order position and carry out.

5.  $+57 = 0011\ 1001$   
 $+92 = 0101\ 1100$   
 $+149 = *1001\ 0101$

\*Overflow—carry into high-order position, no carry out.

6.  $-57 = 1100\ 0111$   
 $-92 = 1010\ 0100$   
 $-149 = *0110\ 1011$

\*Overflow—no carry into high-order position but carry out.

The presence or absence of an overflow condition may be recognized from the carries:

- There is no overflow:
  - a. If there is no carry into the high-order bit position and no carry out (examples 1 and 3).
  - b. If there is a carry into the high-order position and also a carry out (examples 2 and 4).
- There is an overflow:
  - a. If there is a carry into the high-order position but no carry out (example 5).
  - b. If there is no carry into the high-order 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).

<i>Number</i>	<i>Decimal</i>	<i>S</i>	<i>Integer</i>
$2^{15}-1$	= 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

The following are 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.

Number	Decimal	S	Integer
$2^{31}-1$	2 147 483 647	= 0	111 1111 1111 1111 1111 1111 1111 1111
$2^{16}$	65 536	= 0	000 0000 0000 0001 0000 0000 0000 0000
$2^0$	1	= 0	000 0000 0000 0000 0000 0000 0000 0001
0	0	= 0	000 0000 0000 0000 0000 0000 0000 0000
$-2^0$	-1	= 1	111 1111 1111 1111 1111 1111 1111 1111
$-2^1$	-2	= 1	111 1111 1111 1111 1111 1111 1111 1110
$-2^{16}$	-65 536	= 1	111 1111 1111 1111 0000 0000 0000 0000
$-2^{31}+1$	-2 147 483 647	= 1	000 0000 0000 0000 0000 0000 0000 0001
$-2^{31}$	-2 147 483 648	= 1	000 0000 0000 0000 0000 0000 0000 0000

### 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.

- $$\begin{array}{r} 57 = 0011\ 1001 \\ 35 = 0010\ 0011 \\ \hline 92 = 0101\ 1100 \end{array}$$
- $$\begin{array}{r} 57 = 0011\ 1001 \\ 221 = 1101\ 1101 \\ \hline 278 = *0001\ 0110 \end{array}$$

\*Carry out of high-order position
- $$\begin{array}{r} 35 = 0010\ 0011 \\ 199 = 1100\ 0111 \\ \hline 234 = 1110\ 1010 \end{array}$$
- $$\begin{array}{r} 199 = 1100\ 0111 \\ 221 = 1101\ 1101 \\ \hline 420 = *1010\ 0100 \end{array}$$

\*Carry out of high-order position
- $$\begin{array}{r} 57 = 0011\ 1001 \\ 92 = 0101\ 1100 \\ \hline 149 = 1001\ 0101 \end{array}$$
- $$\begin{array}{r} 199 = 1100\ 0111 \\ 164 = 1010\ 0100 \\ \hline 363 = *0110\ 1011 \end{array}$$

\*Carry out of high-order position

A carry out of the high-order bit position may or may not imply an overflow, depending on the application.

The following are several 32-bit unsigned binary integers arranged in descending order:

Number	Decimal	Integer
$2^{32}-1$	4 294 967 296	= 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

### Decimal Integers

Decimal integers are represented as one or more decimal digits and a sign digit. Each digit is 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 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 but are not generated for results.

Decimal integers may have different lengths, from one to sixteen bytes. There are two decimal formats: packed and zoned. In the packed format, each byte contains two decimal digits, except for the rightmost byte which contains the sign in its right digit. The number of decimal digits in the packed format can vary from one to 31. Because decimal integers must consist of whole bytes and there must be a sign digit on the right, the number of decimal digits 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, decimal integers in the zoned format can have anywhere 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 low-order digit and handled as a separate character. For positive (unsigned) numbers, however, the sign code of the low-order digit can simply be replaced by the zone code, which 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:

Value	Packed Format	Zoned Format
+123	12 3C	F1 F2 C3 or F1 F2 F3
-4321	04 32 1D	F4 F3 F2 D1
+000050	00 00 05 0C	F0 F0 F0 F0 F5 C0 or F0 F0 F0 F0 F5 F0
-7	7D	D7
00000	00 00 0C	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)

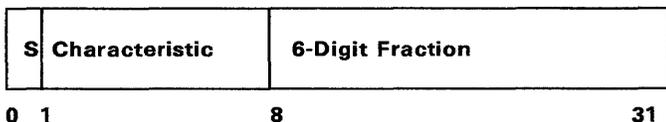
because the product sign follows the algebraic rule of signs even when the value is zero. A negative zero, however, is entirely equivalent to a positive zero; they compare equal in a decimal comparison.

### Floating-Point Numbers

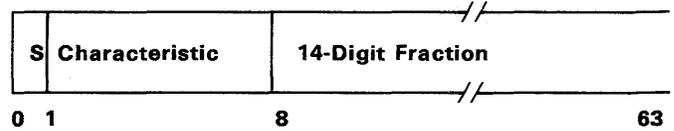
A floating-point number is expressed as a fraction multiplied by a separate power of 16. The term floating point indicates that the radix-point placement, or scaling, is automatically maintained by the machine.

The part of a floating-point 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).

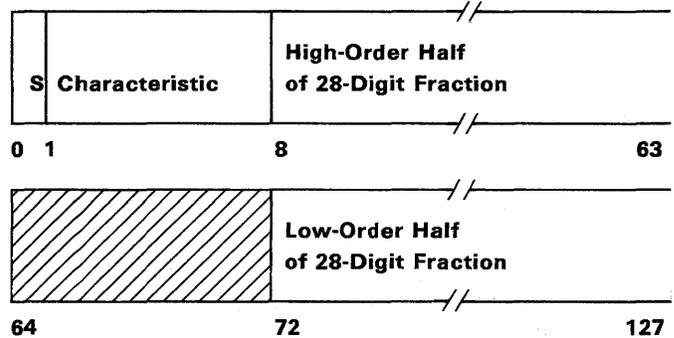
#### Short Floating-Point Number



#### Long Floating-Point Number



#### Extended Floating-Point Number



A floating-point 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, must be less than one and greater than or equal to  $1/16$ , so that the range covered by the magnitude M of a floating-point number is:

$$16^{-65} \leq M < 16^{63}$$

In decimal terms:

$$16^{-65} \text{ is approximately equal to } 54 \times 10^{-79}$$

$$16^{63} \text{ is approximately equal to } 7.2 \times 10^{75}$$

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), a floating-point operation will provide the greatest precision if the fraction is normalized. A fraction is normalized when the high-order digit (bit positions 8, 9, 10, and 11) is nonzero. It is unnormalized if the high-order digit contains all zeros.

If normalization of the operand is desired, the floating-point 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 high-order digit position. The characteristic is reduced by one for each digit shifted.

The following are sample normalized short floating-point numbers. The last two numbers represent the smallest and the largest positive normalized numbers.

Number	Powers of 16	S	Char	Fraction
1.0	= +1/16x16 <sup>1</sup>	= 0	100 0001	0001 0000 0000 0000 0000 0000
0.5	= +8/16x16 <sup>0</sup>	= 0	100 0000	1000 0000 0000 0000 0000 0000
1/64	= +4/16x16 <sup>-1</sup>	= 0	011 1111	0100 0000 0000 0000 0000 0000
0.0	= +0 x16 <sup>-64</sup>	= 0	000 0000	0000 0000 0000 0000 0000 0000
-15.0	= -15/16x16 <sup>1</sup>	= 1	100 0001	1111 0000 0000 0000 0000 0000
5.4x10 <sup>-79</sup>	≅ +1/16x16 <sup>-64</sup>	= 0	000 0000	0001 0000 0000 0000 0000 0000
7.2x10 <sup>75</sup>	≅ (1-16 <sup>-6</sup> )x16 <sup>63</sup>	= 0	111 1111	1111 1111 1111 1111 1111 1111

### Conversion Example

Convert the decimal number 149.25 to a short floating-point number. (In another appendix are tables for the conversion of hexadecimal and decimal integers and fractions.)

1. The number is decomposed into a decimal integer and a decimal fraction.

$$149.25 = 149 \text{ plus } 0.25$$

2. The decimal integer is converted to its hexadecimal representation.

$$149_{10} = 95_{16}$$

3. The decimal fraction is converted to its hexadecimal representation.

$$0.25_{10} = 0.4_{16}$$

4. Combine the integral and fractional parts and express as a fraction times a power of 16 (exponent).

$$95.4_{16} = 0.954_{16} \times 16^2$$

5. The characteristic is developed from the exponent and converted to binary.

$$\begin{aligned} \text{base} + \text{exponent} &= \text{characteristic} \\ 64 + 2 &= 66 = 1000010 \end{aligned}$$

6. The fraction is converted to binary and grouped hexadecimally.

$$.954_{16} = .1001\ 0101\ 0100$$

7. The characteristic and the fraction are stored in the short format. The sign position contains the sign of the fraction.

$$\begin{aligned} &S \text{ CHAR} \text{ FRACTION} \\ &0\ 1000010\ 1001\ 0101\ 0100\ 0000\ 0000\ 0000 \end{aligned}$$

### 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 in this manual for the particular instruction of interest to him.

The instruction-use examples are written principally for assembler-language programmers, to be used in conjunction with the appropriate assembler-language manuals.

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 numerical operands are written in hexadecimal notation unless otherwise specified. Hexadecimal operands are shown converted into binary, decimal, or both if such conversion helps to clarify the example for the reader. Storage addresses are also given in hexadecimal.

#### 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', or 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, S1 represents the combination of B1 and D1.) In the machine-language format, the base and displacement address components are shown as asterisks (\*\*).

## General Instructions

(See Chapter 7.)

### ADD HALFWORD (AH)

The ADD HALFWORD instruction algebraically adds the halfword contents of a storage location to the contents of a register. The halfword 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<sub>10</sub>.

Storage locations 2000-2001 contain FF FE = -2<sub>10</sub>.

Register 12 contains 00 00 18 00.

Register 13 contains 00 00 01 50.

The format of the required instruction is:

#### Assembler Format

Op Code	R <sub>1</sub>	D <sub>2</sub>	X <sub>2</sub>	B <sub>2</sub>
AH	5,X'6B0'(13,12)			

#### Machine Format

Op Code	R <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
4A	5	D	C	6B0

After the instruction is executed, register 5 contains 00 00 00 17 = 23<sub>10</sub>.

### AND (N, NR, NI, NC)

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 0101<sub>2</sub>

Second-operand byte: 0101 1100<sub>2</sub>

Result byte: 0001 0100<sub>2</sub>

### AND (NI)

A frequent use of the AND instruction is to set a particular bit to zero. For example, assume that storage location 4891 contains 0100 0011<sub>2</sub>. 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):

#### Assembler Format

Op Code	D <sub>1</sub> B <sub>1</sub>	I <sub>2</sub>
NI	1(8),X'FE'	

#### Machine Format

Op Code	I <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>
94	FE	8	001

When this instruction is executed, the byte in storage is ANDed with the immediate byte (the I<sub>2</sub> field of the instructions):

```
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.

### BRANCH AND LINK (BAL, BALR)

The BRANCH AND LINK instructions are commonly used to branch to a subroutine with the option of later returning to the main instruction sequence. For example, assume that you wish to branch to a subroutine at storage address 1160. Also assume:

The contents of register 2 are not significant.  
 Register 5 contains 00 00 11 50.  
 Address 00 00 C6 contains the BAL instruction, so that 00 00 CA is the address of the next sequential instruction.

The format of the BAL instruction is:

#### Assembler Format

```
Op Code  R1 D2 X2B2
-----
  BAL    2,X'10'(0,5)
```

#### Machine Format

Op Code	R <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
45	2	0	5	010

After the instruction is executed:

Register 2 (bits 8-31) contains 00 00 CA.  
 PSW bits 40-63 contain 00 11 60.

The programmer can return to the main instruction sequence at any time with a BRANCH ON CONDITION (BCR) instruction that specifies register 2 and a mask of 15<sub>10</sub>, provided that register 2 has not meanwhile been disturbed.

The BALR instruction with the R2 field equal to zero may be used to load a register for use as a base register. For example, in the assembler language, the sequence of statements:

```
BALR  15,0
USING *,15
```

tells the assembler program that register 15 is to be used as the base register in assembling this program and that when the program is executed, the address of the next sequential instruction following the BALR will be placed in the register. (The USING statement is an "assembler instruction" and is thus not a part of the object program.)

BALR 6,0 preserves the condition code in bits 2 and 3 of register 6 for future inspection.

### BRANCH ON CONDITION (BC, BCR)

The BRANCH ON CONDITION instructions test the condition code to see whether a branch should or should not be taken. The branch is taken only if the condition code is as specified by a mask.

Mask Value	Condition Code
8	0
4	1
2	2
1	3

For example, assume that an ADD (A, AR) operation has been performed and that you wish to branch to address 6050 if the sum is zero or less (condition code 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:

#### Assembler Format

```
Op Code  M1 D2 X2B2
-----
  BC     12,X'50'(11,10)
```

#### Machine Format

Op Code	M <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
47	C	B	A	050

A mask of 15 indicates a branch on any condition (an unconditional branch). A mask of zero indicates that no branch is to occur (a no-operation).

### BRANCH ON COUNT (BCT, BCTR)

The BRANCH ON COUNT instructions are 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:

#### Assembler Format

```

Op Code  R1 D2 X2B2
-----
  BCT    6,X'26'(0,10)

```

#### Machine Format

Op Code	R <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
46	6	0	A	026

The effect of the coding is to execute three times the loop defined by locations LUPE through BACK.

### BRANCH ON INDEX HIGH (BXH)

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 comparand. For example, assume that:

Register 4 contains 00 00 00 8A = 138<sub>10</sub> = the index.  
 Register 6 contains 00 00 00 02 = 2<sub>10</sub> = the increment.  
 Register 7 contains 00 00 00 AA = 170<sub>10</sub> = the comparand.  
 Register 10 contains 00 00 71 30 = the branch address.

The format of the instruction is:

#### Assembler Format

```

Op Code  R1R3D2 B2
-----
  BXH    4,6,0(10)

```

#### Machine Format

Op Code	R <sub>1</sub>	R <sub>3</sub>	B <sub>2</sub>	D <sub>2</sub>
86	4	6	A	000

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 = 140<sub>10</sub>  
 Registers 6 and 7 are unchanged.

Since the new value in register 4 is not greater than the value in register 7, the branch to address 7130 is not taken.

When the register used to contain the increment is odd, that register also becomes the comparand register. The following assembler-language routine illustrates how this feature 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 0 contains the search argument.  
 Register 1 contains the width of the table in bytes (00 00 00 04).  
 Register 2 contains the length of the table in bytes (00 00 00 18).  
 Register 3 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 0 is successively compared with the arguments in the table, starting with argument 6 and working backwards to argument 1. If an equality is found, the corresponding function replaces the argument in register 0. If an equality is not found,

FF<sub>16</sub> replaces the argument in register 0.

The first instruction (LNR) causes the value in register 1 to be made negative. After execution of this instruction, register 1 contains FFFFFFFC = -4<sub>10</sub>. Considering the case when no equality is found, the BXH instruction will be executed seven times. Each time the BXH is executed, a value of -4 is added to register 2, thus reducing the value in register 2 by 4. The new value in register 2 is compared with the -4 value in register 1. Thus the branch is taken each time until the value in register 2 is -4.

SEARCH	LNR	1,1
NOTEQUAL	BXH	2,1,LOOP
NOTFOUND	LA	0,X'FF'
	BCR	15,14
LOOP	CH	0,0(2,3)
	BC	7,NOTEQUAL
	LH	0,2(2,3)
	BCR	15,14

### BRANCH ON INDEX LOW OR EQUAL (BXLE)

This instruction is similar to BRANCH ON INDEX HIGH except that the branch is successful when the sum is low or equal compared to the comparand.

### 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<sub>10</sub>.
- Register 13 contains 00 01 60 50.
- Storage locations 16080-16081 contain 8000 = -32,768<sub>10</sub>.

When the instruction

#### Assembler Format

Op Code	R <sub>1</sub>	D <sub>2</sub>	X <sub>2</sub> B <sub>2</sub>
CH	4	X'30'	(0,13)

#### Machine Format

Op Code	R <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
49	4	0	D	030

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, CLR, CLI, CLC)

The COMPARE LOGICAL instructions differ from the signed-binary comparison instructions (C, CH, CR) in that all quantities are handled as unsigned binary integers or as unstructured data.

#### Compare Logical (CLR)

Assume that:

- Register 4 contains 00 00 00 01 = 1.
- Register 7 contains FF FF FF FF = 2<sup>32</sup>-1.

Execution of the instruction

#### Assembler Format

Op Code	R <sub>1</sub> R <sub>2</sub>
CLR	4,7

#### Machine Format

Op Code	R <sub>1</sub>	R <sub>2</sub>
15	4	7

sets condition code 1. Condition code 1 indicates that the first operand is lower than the second.

If, instead, a signed-binary comparison instruction 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 (CLI)

The 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

*Assembler Format*

Op Code D<sub>1</sub>B<sub>1</sub> I<sub>2</sub>  
-----  
CLI 3(10),X'AF'

*Machine Format*

Op Code	I <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>
95	AF	A	003

sets condition code 1, indicating that the first operand (the quantity in main storage) is lower than the second (immediate) operand.

**Compare Logical (CLC)**

The CLC instruction can be used to perform the bit-for-bit 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

*Assembler Format*

Op Code D<sub>1</sub>L B<sub>1</sub> D<sub>2</sub>B<sub>2</sub>  
-----  
CLC 6(12,9),0(7)

*Machine Format*

Op Code	L	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
D5	0B	9	006	7	000

sets condition code 1, indicating that the contents of field 1 are lower in value than the contents of field 2.

Because CLC compares on a bit-for-bit basis, the instruction can be used to collate fields composed of characters from the EBCDIC code. For example, in EBCDIC, the above two data fields are:

Field 1 JOHNSON,A.B.  
Field 2 JOHNSON,A.C.

Condition code 1 tells us that A.B.JOHNSON precedes A.C.JOHNSON, thus placing the names in the correct alphabetic order.

**COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)**

The CLM instruction provides a means of comparing bytes selected from a general register to a contiguous field of bytes in main storage. The M<sub>3</sub> 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:

Three bytes starting at storage location 10200 contain F0 BC 7B.  
Register 12 contains 10000.  
Register 6 contains F0 BC 5C 7B.

Execution of the instruction

*Assembler Format*

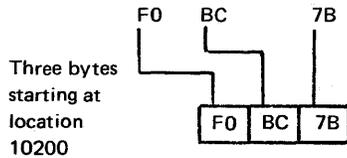
Op Code R<sub>1</sub> M<sub>3</sub> D<sub>2</sub> B<sub>2</sub>  
-----  
CLM 6,B'1101',X'200'(12)

*Machine Format*

Op Code	R <sub>1</sub>	M <sub>3</sub>	B <sub>2</sub>	D <sub>2</sub>
BD	6	D	C	200

causes the following comparison:

Register 6: F0 BC 5C 7B  
Mask: 1 1 0 1

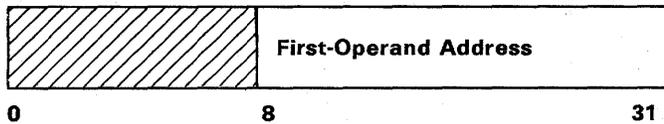


Because the selected bytes are equal, condition code 0 is set.

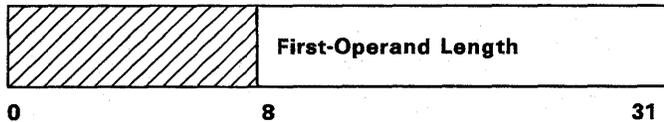
### COMPARE LOGICAL LONG (CLCL)

The CLCL instruction is used to compare two operands in main storage, bit for bit. 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 location of the byte currently being compared in each operand. The odd register of each pair contains the length of the operand it covers, and the high-order 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.

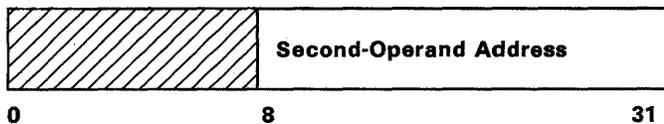
R<sub>1</sub> (even)



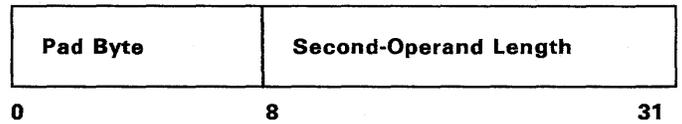
R<sub>1</sub>+1 (odd)



R<sub>2</sub> (even)



R<sub>2</sub>+1 (odd)



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:

<i>Operand 1</i>	<i>Padding Byte</i>
Address: 20800 (hex)	Address: 20003 (hex)
Length: 100 (dec)	Length: 1
	Value: 40 (hex)

*Operand 2*  
Address: 20A00 (hex)  
Length: 132 (dec)

Register 12 contains 00 02 00 00

The setup instructions are:

LA 4,X'800'(12)	Point register 4 to start of first operand
LA 5,100	Set register 5 to length of first operand
LA 8,X'A00'(12)	Point 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 location 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:

### Assembler Format

Op Code R<sub>1</sub> R<sub>2</sub>

-----  
CLCL 4,8

### Machine Format

Op Code R<sub>1</sub> R<sub>2</sub>

0F	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 are successfully compared, the operand lengths in registers 5 and 9 will have been decremented to X'28' and X'48', respectively, and the operand locations in registers 4 and 8 will have been incremented to X'2083C' and X'20A3C'. The padding byte X'40' remains in register 9. When the CLCL instruction is reissued with these register contents, the comparison resumes at the point of interruption.

If the instruction is interrupted after 110 bytes are successfully compared, the residual operand lengths in registers 5 and 9 are 0 and X'16', respectively, and the current operand locations in registers 4 and 8 are X'2086E' and X'20A6E'.

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 locate the bytes that caused the mismatch.

### CONVERT TO BINARY (CVB)

The CONVERT TO BINARY instruction converts an eight-byte, packed-decimal number into a signed binary integer and loads the result into a general 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 packed format: 00 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:

### Assembler Format

Op Code R<sub>1</sub>D<sub>2</sub>X<sub>2</sub>B<sub>2</sub>

-----  
CVB 7,8(0,13)

### Machine Format

Op Code R<sub>1</sub> X<sub>2</sub> B<sub>2</sub> D<sub>2</sub>

4F	7	0	D	008
----	---	---	---	-----

After the instruction is executed, register 7 contains 00 00 63 FA.

### CONVERT TO DECIMAL (CVD)

The CONVERT TO DECIMAL instruction performs functions exactly opposite to those of the CONVERT TO BINARY instruction. CVD converts a signed binary integer in a register to packed decimal 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:

### Assembler Format

Op Code R<sub>1</sub>D<sub>2</sub>X<sub>2</sub>B<sub>2</sub>

-----  
CVD 1,8(0,13)

### Machine Format

Op Code R<sub>1</sub> X<sub>2</sub> B<sub>2</sub> D<sub>2</sub>

4E	1	0	D	008
----	---	---	---	-----

After the instruction is executed, storage locations 7608-760F contain 00 00 00 00 00 03 85 5C (+3855).

The plus sign generated is the preferred plus sign, 1100<sub>2</sub>.

## 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 dividend is assumed to be 64 bits long, it is important that the proper sign be first affixed. For example, assume that:

Storage locations 3550-3553 contain 00 00 08 DE = 2270<sub>10</sub> = the dividend.

Storage locations 3554-3557 contain 00 00 00 32 = 50<sub>10</sub> = 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 <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
5D	6	0	8	004

After all the foregoing instructions are executed:  
Register 6 contains 00 00 00 14 = 20<sub>10</sub> = the remainder.  
Register 7 contains 00 00 00 2D = 45<sub>10</sub> = 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 sign bits (zeros in this example), and the DIVIDE instruction might not have given the expected results.

## EXCLUSIVE OR (X, XR, XI, XC)

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 0101 <sub>2</sub>
Second-operand byte:	0101 1100 <sub>2</sub>
Result byte:	0110 1001 <sub>2</sub>

## Exclusive OR (XI)

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<sub>2</sub>. 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):

### Assembler Format

Op Code	D <sub>1</sub> B <sub>1</sub> I <sub>2</sub>
XI	2(9),X'81'

### Machine Format

Op Code	I <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>
97	81	9	002

When the instruction is executed, the byte in storage is EXCLUSIVE ORed with the immediate byte (the I<sub>2</sub> field of the instruction):

Location 8082:	0110 1001 <sub>2</sub>
Immediate byte:	1000 0001 <sub>2</sub>
Result:	1110 1000 <sub>2</sub>

The resulting byte is stored back in location 8082. Condition code 1 is set to indicate a nonzero result.

## Exclusive OR (XC)

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 3-byte fields in storage:

Field 1

359	35B
00	17 90

Field 2

360	362
00	14 01

Execution of the instruction (assume that register 7 contains 00 00 03 58):

*Assembler Format*

Op Code    D<sub>1</sub>LB<sub>1</sub> D<sub>2</sub>B<sub>2</sub>  
-----  
XC        1(3,7),8(7)

*Machine Format*

Op Code	L	B <sub>1</sub>	D <sub>1</sub>	B <sub>1</sub>	D <sub>2</sub>
D7	02	7	001	7	008

Field 1 is EXCLUSIVE ORed with field 2 as follows:

Field 1:    0000 0000 0001 0111 1001 0000<sub>2</sub>  
          = 00 17 90  
Field 2:    0000 0000 0001 0100 0000 0001<sub>2</sub>  
          = 00 14 01  
-----  
Result:    0000 0000 0000 0011 1001 0001<sub>2</sub>  
          = 00 03 91

The result replaces the former contents of field 1.  
Now, execution of the instruction

*Assembler Format*

Op Code    D<sub>1</sub>LB<sub>1</sub> D<sub>2</sub>B<sub>2</sub>  
-----  
XC        8(3,7),1(7)

*Machine Format*

Op Code	L	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
D7	02	7	008	7	001

produces the following result:

Field 1:    0000 0000 0000 0011 1001 0001<sub>2</sub>  
          = 00 03 91  
Field 2:    0000 0000 0001 0100 0000 0001<sub>2</sub>  
          = 00 14 01  
-----  
Result:    0000 0000 0001 0111 1001 0000<sub>2</sub>  
          = 00 17 90

The result of this operation replaces the former contents of field 2. Field 2 now contains the original value of field 1.

Lastly, execution of the instruction

*Assembler Format*

Op Code    D<sub>1</sub>LB<sub>1</sub> D<sub>2</sub>B<sub>2</sub>  
-----  
XC        1(3,7),8(7)

*Machine Format*

Op Code	L	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
D7	02	7	001	7	008

produces the following result:

Field 1:    0000 0000 0000 0011 1001 0001<sub>2</sub>  
          = 00 03 91  
Field 2:    0000 0000 0001 0111 1001 0000<sub>2</sub>  
          = 00 17 90  
-----  
Result:    0000 0000 0001 0100 0000 0001<sub>2</sub>  
          = 00 14 01

The result of this operation replaces the former contents of field 1. Field 1 now contains the original value of field 2.

**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.

**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 R<sub>1</sub> field of the EXECUTE instruction is zero, bits 8-15 of the target instruction are ORed with bits 24-31 of the R<sub>1</sub> 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:

*Assembler Format*

Op Code D<sub>1</sub>L B<sub>1</sub> D<sub>2</sub>B<sub>2</sub>  
 -----  
 MVC 3(1,12),0(13)

*Machine Format*

Op Code	L	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
D2	00	C	003	D	000

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:

*Assembler Format*

Op Code R<sub>1</sub>D<sub>2</sub>X<sub>2</sub>B<sub>2</sub>  
 -----  
 EX 1,0(0,10)

*Machine Format*

Op Code	R <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
44	1	0	A	000

where register 10 contains 00 00 38 20 and register 1 contains 00 0F FO 03.

When the instruction at 5000 is executed, the rightmost byte of register 1 is ORed with the second byte of the target instruction:

Register byte: 0000 0000<sub>2</sub> = 00  
 Instruction byte: 0000 0011<sub>2</sub> = 03  
 Result: 0000 0011<sub>2</sub> = 03

causing the instruction at 3820 to be executed as if it originally were:

*Assembler Format*

Op Code D<sub>1</sub>L B<sub>1</sub> D<sub>2</sub>B<sub>2</sub>  
 -----  
 MVC 3(4,12),0(13)

*Machine Format*

Op Code	L	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
D2	03	C	003	D	000

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 40-63 contain 00 50 04).

**INSERT CHARACTERS UNDER MASK (ICM)**

The 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 high-order byte of the register unchanged, assume:

*Assembler Format*

Op Code R<sub>1</sub> M<sub>3</sub> S<sub>2</sub>  
 -----  
 ICM 5,B'0111',FIELD A

*Machine Format*

Op Code	R <sub>1</sub>	M <sub>3</sub>	S <sub>2</sub>
BF	5	7	****

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:

*Assembler Format*

Op Code R<sub>1</sub> M<sub>3</sub> S<sub>2</sub>  
 -----  
 ICM 6,B'1001',FIELD B

*Machine Format*

Op Code	R <sub>1</sub>	M <sub>3</sub>	S <sub>2</sub>
BF	6	9	****



The instruction required to load the register is:

*Assembler Format*

Op Code R<sub>1</sub>D<sub>2</sub> X<sub>2</sub>B<sub>2</sub>

LH 6,0(0,14)

*Machine Format*

Op Code R<sub>1</sub> X<sub>2</sub> B<sub>2</sub> D<sub>2</sub>

48	6	0	E	000
----	---	---	---	-----

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 (MVI)**

The MOVE (immediate) instruction places one byte of information from the instruction stream into storage. For example, the instruction

*Assembler Format*

Op Code D<sub>1</sub> B<sub>1</sub> I<sub>2</sub>

MVI 0(1),C'\$'

*Machine Format*

Op Code I<sub>2</sub> B<sub>1</sub> D<sub>1</sub>

92	5B	1	000
----	----	---	-----

may be used, in conjunction with the instruction EDIT AND MARK, to insert a dollar symbol at the storage address contained in general register 1 (see also the example for EDIT AND MARK).

**MOVE (MVC)**

The 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:

Field 1

2048

2052

C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB
----	----	----	----	----	----	----	----	----	----	----

Field 2

3840

3848

F1	F2	F3	F4	F5	F6	F7	F8	F9
----	----	----	----	----	----	----	----	----

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:

*Assembler Format*

Op Code D<sub>1</sub>L B<sub>1</sub> D<sub>2</sub>B<sub>2</sub>

MVC 0(8,1),0(2)

*Machine Format*

Op Code L B<sub>1</sub> D<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

D2	07	1	000	2	000
----	----	---	-----	---	-----

After the instruction is executed, field 1 becomes:

Field 1

2048

2052

F1	F2	F3	F4	F5	F6	F7	F8	C9	CA	CB
----	----	----	----	----	----	----	----	----	----	----

Field 2 is unchanged.

MVC can also be used to propagate a byte through a field by starting the first-operand field one byte location to the right of the second-operand field. For example, suppose that an area in storage starting with address 358 contains the following data:

358

360

00	F1	F2	F3	F4	F5	F6	F7	F8
----	----	----	----	----	----	----	----	----

With the following MVC instruction, the zeros in location 358 can be propagated throughout the entire field (assume that register 11 contains 00 00 03 58):

### Assembler Format

Op Code D<sub>1</sub>L B<sub>1</sub> D<sub>2</sub> B<sub>2</sub>

MVC 1(8,11),0(11)

### Machine Format

Op Code L B<sub>1</sub> D<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

D2	07	B	001	B	000
----	----	---	-----	---	-----

Because the MVC handles one byte at a time, the above instruction essentially takes the byte at address 358 and stores it at 359 (359 now contains 00), takes the byte at 359 and stores it at 35A, and so on, until the entire field is filled with zeros. Note that an MVI instruction could have been used originally to place the byte of zeros in location 358.

### Notes:

1. Although the field occupying locations 358-360 contains nine bytes, the length coded in the assembler format is equal to the number of moves (one less than the field length).
2. The order of operands is important even though only one field is involved.

### MOVE LONG (MVCL)

The 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

### Assembler Format

Op Code R<sub>1</sub>R<sub>2</sub>

MVCL 8,2

### Machine Format

Op Code R<sub>1</sub> R<sub>2</sub>

0E	8	2
----	---	---

moves 2,048<sub>10</sub> bytes from locations A0000-A07FF to location 60000-607FF. Condition code 0 is set to indicate that the operand lengths are equal.

If register 3 had contained F0 00 04 00, only the 1,024<sub>10</sub> 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. Condition code 2 is set to indicate that the first operand is 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. The contents of register 2 are not significant. Executing the instruction MVCL 8,2 then causes locations 60000-607FF to be filled with zeros. Condition code 2 is set.

### MOVE NUMERICS (MVN)

To illustrate the operation of the MOVE NUMERICS instruction, assume that the following two fields are in storage:

Field A

7090 7093

C6	C7	C8	C9
----	----	----	----

Field B

7041 7046

F0	F1	F2	F3	F4	F5
----	----	----	----	----	----

Also assume:

Register 14 contains 00 00 70 90.

Register 15 contains 00 00 70 40.

After the instruction

*Assembler Format*

Op Code D<sub>1</sub>L B<sub>1</sub> D<sub>2</sub> B<sub>2</sub>

MVN 1(4,15),0(14)

*Machine Format*

Op Code L B<sub>1</sub> D<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

D1	03	F	001	E	000
----	----	---	-----	---	-----

is executed, field B becomes:

7041 7046

F6	F7	F8	F9	F4	F5
----	----	----	----	----	----

The numeric portions (the rightmost four bits) of the bytes at locations 7090-7093 have been stored in the numeric portions of the bytes at locations 7041-7044. The contents of locations 7090-7093 and 7045-7046 are unchanged.

### MOVE WITH OFFSET (MVO)

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 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

*Assembler Format*

Op Code D<sub>1</sub>L<sub>1</sub>B<sub>1</sub> D<sub>2</sub> L<sub>2</sub>B<sub>2</sub>

MVO 0(4,12),0(3,15)

*Machine Format*

Op Code L<sub>1</sub> L<sub>2</sub> B<sub>1</sub> D<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

F1	3	2	C	000	F	000
----	---	---	---	-----	---	-----

is executed, the storage locations 5600-5603 contain 01 23 45 6C. Note that the second operand is extended with one high-order zero to fill out the first-operand field.

### MOVE ZONES (MVZ)

The MOVE ZONES instruction can, similarly to MVC and MVN, operate on overlapping or nonoverlapping fields. (See the examples for MVC and MVN.) When operating on nonoverlapping fields, MVZ works similarly to the MVN instruction, except that MVZ moves the zone portion (the leftmost four bits) of each byte. To illustrate the use of MVZ 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

*Assembler Format*

Op Code D<sub>1</sub>L B<sub>1</sub> D<sub>2</sub> B<sub>2</sub>

MVZ 1(5,15),0(15)

*Machine Format*

Op Code L B<sub>1</sub> D<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

D3	04	F	001	F	000
----	----	---	-----	---	-----

propagates the zone from the byte at address 800 through the data 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 word at address 3750. Initially:

The contents of register 4 are not significant.

Register 5 contains 00 00 00 9A = 154<sub>10</sub> = the multiplicand.

Register 11 contains 00 00 30 00.

Register 12 contains 00 00 06 00.

Storage locations 3750-3753 contain 00 00 00 83 = 131<sub>10</sub> = the multiplier.

The instruction required for performing the multiplication is:

#### *Assembler Format*

Op Code	R <sub>1</sub>	D <sub>2</sub>	X <sub>2</sub>	B <sub>2</sub>
M	4	X'150'	(11,12)	

#### *Machine Format*

Op Code	R <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
5C	4	B	C	150

After the instruction is executed:

Registers 4 and 5 contain the product:

Register 4 contains 00 00 00 00.

Register 5 contains 00 00 4E CE = 20,174<sub>10</sub>.

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 00 00 10 = 16<sub>10</sub>. The instruction

#### *Assembler Format*

Op Code	R <sub>1</sub> R <sub>2</sub>
MR	6,7

#### *Machine Format*

Op Code	R <sub>1</sub>	R <sub>2</sub>
1C	6	7

multiplies the number in register 7 by itself.

The product, 00 00 00 00 00 00 01 00 = 256<sub>10</sub>, appears in registers 6 and 7.

### **MULTIPLY HALFWORD (MH)**

The MULTIPLY HALFWORD instruction is used to multiply the contents of a register by a halfword in storage. For example, assume that:

Register 11 contains 00 00 00 15 = 21<sub>10</sub> = the multiplicand.

Register 14 contains 00 00 01 00.

Register 15 contains 00 00 20 00.

Storage locations 2102-2103 contain FF D9 = -39 = the multiplier.

The instruction

#### *Assembler Format*

Op Code	R <sub>1</sub> D <sub>2</sub>	X <sub>2</sub>	B <sub>2</sub>
MH	11,2	(14,15)	

#### *Machine Format*

Op Code	R <sub>1</sub>	X <sub>2</sub>	B <sub>2</sub>	D <sub>2</sub>
4C	B	E	F	002

multiplies the two numbers. The product, FF FF FC CD = -819<sub>10</sub>, replaces the original contents of register 11.

Only the low-order 32 bits of a product are stored in a register; any high-order bits are lost. No program interruption occurs on overflow.

### OR (O, OR, OI, OC)

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 0101<sub>2</sub>  
 Second-operand byte: 0101 1100<sub>2</sub>  
 Result byte: 0111 1101<sub>2</sub>

### OR (OI)

A frequent use of the OR instruction is to set a particular bit to one. For example, assume that storage location 4891 contains 0100 0010<sub>2</sub>. 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):

#### Assembler Format

Op Code D<sub>1</sub>B<sub>1</sub> I<sub>2</sub>  
 -----  
 OI 1(8),X'01'

#### Machine Format

Op Code	I <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>
96	01	8	001

When this instruction is executed, the byte in storage is ORed with the immediate byte (the I<sub>2</sub> field of the instruction):

Location 4891: 0100 0010<sub>2</sub>  
 Immediate byte: 0000 0001<sub>2</sub>  
 Result: 0100 0011<sub>2</sub>

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 packed-decimal number and left in the same location:

### Zoned Number

1000 1003

F1	F2	F3	C4
----	----	----	----

Also assume that register 12 contains 00 00 10 00. After the instruction

#### Assembler Format

Op Code D<sub>1</sub>L<sub>1</sub>B<sub>1</sub> D<sub>2</sub>L<sub>2</sub>B<sub>2</sub>  
 -----  
 PACK 0(4,12),0(4,12)

#### Machine Format

Op Code	L <sub>1</sub>	L <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
F2	3	3	C	000	C	000

is executed, the result in locations 1000-1003 is in the packed-decimal format:

#### Packed Number

1000 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 with high-order zeros.

### SHIFT LEFT DOUBLE (SLDA)

The SHIFT LEFT DOUBLE instruction is similar to SHIFT LEFT SINGLE except that SLDA shifts the 63 bits (not including the sign) of an even-odd register pair. The R<sub>1</sub> field of this instruction must be even. For example, if the contents of registers 2 and 3 are:

00 7F 0A 72 FE DC BA 98 =  
 0000 0000 0111 1111 0000 1010 0111 0010  
 1111 1110 1101 1100 1011 1010 1001 1000<sub>2</sub>

the instruction

*Assembler Format*

Op Code R<sub>1</sub>D<sub>2</sub>B<sub>2</sub>

SLDA 2,31(0)

*Machine Format*

Op Code R<sub>1</sub> R<sub>3</sub> B<sub>2</sub> D<sub>2</sub>

8F	2		0	01F
----	---	--	---	-----

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 =  
 0111 1111 0110 1110 0101 1101 0100 1100  
 0000 0000 0000 0000 0000 0000 0000 0000<sub>2</sub>

In this case, a significant bit is shifted out of bit position 1 of register 2. Condition code 3 is set to indicate this overflow and, if the fixed-point-overflow mask bit in the PSW is one, a fixed-point overflow interruption occurs.

**SHIFT LEFT SINGLE (SLA)**

Because SHIFT LEFT SINGLE leaves the sign bit unchanged, this instruction performs an algebraic shift. For example, if the contents of register 2 are:

00 7F 0A 72 = 0000 0000 0111 1111 0000 1010 0111 0010<sub>2</sub>

the instruction

*Assembler Format*

Op Code R<sub>1</sub>D<sub>2</sub>B<sub>2</sub>

SLA 2,8(0)

*Machine Format*

Op Code R<sub>1</sub> R<sub>3</sub> B<sub>2</sub> D<sub>2</sub>

8B	2		0	008
----	---	--	---	-----

results in register 2 being shifted left eight bit positions so that its new contents are:

7F 0A 72 00 = 0111 1111 0000 1010 0111 0010 0000 0000<sub>2</sub>

Condition code 2 is set to indicate that the result is nonzero and positive.

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 is one, a fixed-point overflow interruption would have occurred.

**STORE CHARACTERS UNDER MASK (STCM)**

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:

*Register Format*

Op Code R<sub>1</sub> M<sub>3</sub> S<sub>2</sub>

STCM 8,B'0111',FIELD3

*Machine Format*

Op Code R<sub>1</sub> M<sub>3</sub> S<sub>2</sub>

BE	8	7	****
----	---	---	------

Register 8: 12 34 56 78  
 FIELD3 (before): not significant  
 FIELD3 (after): 34 56 78

As another example:

*Register Format*

Op Code R<sub>1</sub> M<sub>3</sub> S<sub>2</sub>

STCM 9,B'0101',FIELD2

*Machine Format*

Op Code R<sub>1</sub> M<sub>3</sub> S<sub>2</sub>

BE	9	5	****
----	---	---	------

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 words 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:

### Assembler Format

Op Code R<sub>1</sub> R<sub>3</sub> D<sub>2</sub> B<sub>2</sub>

STM 14,1,X'50'(6)

### Machine Format

Op Code R<sub>1</sub> R<sub>3</sub> B<sub>2</sub> D<sub>2</sub>

90	E	1	6	050
----	---	---	---	-----

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.

Execution of the instruction

### Assembler Format

Op Code D<sub>1</sub>B<sub>1</sub> I<sub>2</sub>

TM 9(7),B'11000011'

### Machine Format

Op Code I<sub>2</sub> B<sub>1</sub> D<sub>1</sub>

91	C3	7	009
----	----	---	-----

produces the following result:

FB = 1111 1011<sub>2</sub>  
Mask = 1100 0011<sub>2</sub>

Result = 11 11<sub>2</sub>

Condition code 3 is set: all selected bits are ones.  
If location 9999 had contained B9, the result would have been:

B9 = 1011 1001<sub>2</sub>  
Mask = 1100 0011<sub>2</sub>

Result = 10 01<sub>2</sub>

Condition code 1 is set: the selected bits are both zeros and ones.

If location 9999 had contained 3C, the result would have been:

3C = 0011 1100<sub>2</sub>  
Mask = 1100 0011<sub>2</sub>

Result = 00 00<sub>2</sub>

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 coded character consists of eight bits or fewer. In the following example, EBCDIC is translated to ASCII. The first step is to create a 256-byte table in storage locations 1000-10FF. This table contains the characters of the target code in the sequence of the binary representation of the source 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:

Translate Table for 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

*Assembler Format*

Op Code D<sub>1</sub> L B<sub>1</sub> D<sub>2</sub> B<sub>2</sub>

TR 0(4,12),0(15)

*Machine Format*

Op Code L B<sub>1</sub> D<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

DC	03	C	000	F	000
----	----	---	-----	---	-----

is executed, the binary value of each source byte is added to the starting address of the table, and the resulting address is used to fetch a target byte:

Table starting address: 1000

First source byte: F1

Address of target 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.

The translate-and-test table that follows 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 FF<sub>16</sub>, a table of fewer than 256 bytes can be used.

**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 04<sub>16</sub> to 3C<sub>16</sub> in increments of 4. Thus, the blank (code 40) has the entry 04<sub>16</sub>, the period (code 4B) has the entry 08<sub>16</sub>, and so on.

All other table positions have the entry 40<sub>16</sub> to indicate an invalid character.

The table entries are chosen so that they may be used to select one of a list of 16 fullwords 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 21<sub>10</sub> EBCDIC characters:

Locations CA80-CA94: UNPKbPROUT(9),WORD(5)

Also assume:

Register 1 contains 00 00 2F FF.  
 Register 2 contains 00 00 30 00.  
 Register 15 contains 00 00 20 00.

As the instruction

*Assembler Format*

Op Code	D <sub>1</sub> L	B <sub>1</sub>	D <sub>2</sub> B <sub>2</sub>
TRT	1(21,1),0(15)		

*Machine Format*

Op Code	L	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
DD	14	1	001	F	000

is executed, the value of the first argument byte, 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 argument byte (U)	E4
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 argument 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
Argument byte (blank)	40
Address of table entry	2040

Because location 2040 contains a nonzero value, the following actions occur:

1. The address of the argument byte, 00CA84, is placed in the rightmost 24 bits of register 1.
2. The table entry, 04, is placed in the rightmost eight bits of register 2, which now contains 00 00 30 04.
3. 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 repeated over and over 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 argument byte translated. It is then a simple matter to subtract this address from the address of the last argument 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

*Assembler Format*

Op Code	D <sub>1</sub> L <sub>1</sub> B <sub>1</sub>	D <sub>2</sub> L <sub>2</sub> B <sub>2</sub>
UNPK	0(5,12),1(2,13)	

*Machine Format*

Op Code	L <sub>1</sub>	L <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
F3	4	1	C	000	D	001

is executed, the storage locations 1000-1004 contain F0 F0 F1 F2 D3.

**Decimal Instructions**

(See Chapter 8.)

### 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

#### Assembler Format

Op Code	D <sub>1</sub> L <sub>1</sub> B <sub>1</sub>	D <sub>2</sub> L <sub>2</sub> B <sub>2</sub>
AP	0(3,12),0(4,13)	

#### Machine Format

Op Code	L <sub>1</sub>	L <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
FA	2	3	C	000	D	000

is executed, the storage locations 2000-2002 contain 73 88 5C; condition code 2 is set to indicate that the sum is positive. 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

#### Assembler Format

Op Code	D <sub>1</sub> L <sub>1</sub> B <sub>1</sub>	D <sub>2</sub> L <sub>2</sub> B <sub>2</sub>
CP	X'100'(4,12),X'200'(3,13)	

#### Machine Format

Op Code	L <sub>1</sub>	L <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
F9	3	2	C	100	D	200

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

#### Assembler Format

Op Code	D <sub>1</sub> L <sub>1</sub> B <sub>1</sub>	D <sub>2</sub> L <sub>2</sub> B <sub>2</sub>
DP	0(5,12),0(2,13)	

#### Machine Format

Op Code	L <sub>1</sub>	L <sub>2</sub>	B <sub>1</sub>	D <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
FD	4	1	C	000	D	000

is executed, the dividend is entirely replaced by the signed quotient and remainder, as follows:

Locations 2000-2004

2000 2004

38	46	0D	01	8C
----	----	----	----	----

Quotient | Remainder  
|



Pattern

1000						100C						
40	40	40	40	40	40	F0	4B	F2	F6	40	C3	D9
b	b	b	b	b	b	0	.	2	6	b	C	R

This pattern field prints as:

0.26 CR

The significance starter forces the significance indicator to the on state and hence causes the decimal point to be preserved. Because the minus-sign code has no effect on the significance indicator, the characters CR are printed to show a negative (credit) amount.

Condition code 1 is set (number less than zero).

### EDIT AND MARK (EDMK)

The EDIT AND MARK instruction may be used, in addition to the functions of EDIT, to insert a currency symbol, such as a dollar sign, at the appropriate position in the edited result. Assume the same source in storage locations 1200-1203, the same pattern in locations 1000-100C, and the same contents of general register 12 as for the EDIT instruction above. The previous contents of general register 1 are immaterial; a LOAD ADDRESS instruction is used to set up the first digit position that is forced to print if no significant digits occur to the left.

The instructions

LA	1,6(0,12)	Load address of forced significant digit into GR1.
EDMK	0(13,12),X'200'(12)	Leave address of first significant digit in GR1.
BCTR	1,0	Subtract 1 from address in GR1.
MVI	0(1),C'\$'	Store dollar sign and address in GR1.

produce the following results for the two examples under EDIT:

Pattern

1000						100C						
40	5B	F2	6B	F5	F7	F4	4B	F2	F6	40	40	40
b	\$	2	,	5	7	4	.	2	6	b	b	b

This pattern field prints as:

\$2,574.26

Condition code 2 is set to indicate that the number edited was greater than zero.

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).

Multiplicand

1202		1204	
38	46	0D	

Multiplier

500		501	
32	1D		

Because the multiplier and multiplicand have a total of eight significant digits, at least five bytes must be reserved for the signed result. 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:

Multiplicand (new)

1300				1304			
00	00	38	46	0D			

Now, after the instruction  
*Assembler Format*

Op Code D<sub>1</sub> L<sub>1</sub>B<sub>1</sub> D<sub>2</sub>L<sub>2</sub>B<sub>2</sub>

MP X'100'(5,4),0(2,6)

*Machine Format*

Op Code L<sub>1</sub> L<sub>2</sub> B<sub>1</sub> D<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

FC	4	1	4	100	6	000
----	---	---	---	-----	---	-----

is executed, storage locations 1300-1304 contain the product: 01 23 45 66 0C.

### SHIFT AND ROUND DECIMAL (SRP)

The 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:

*Assembler Format*

Op Code S<sub>1</sub> L<sub>1</sub> S<sub>2</sub>I<sub>3</sub>

SRP FIELD1(6),3,0

*Machine Format*

Op Code L<sub>1</sub> I<sub>3</sub> S<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

F0	5	0	****	0	003
----	---	---	------	---	-----

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 factor, I<sub>3</sub>, is not used in 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:

*Assembler Format*

Op Code S<sub>1</sub> L<sub>1</sub> S<sub>2</sub> I<sub>3</sub>

SRP FIELD2(5),64 - 1,0

*Machine Format*

Op Code L<sub>1</sub> I<sub>3</sub> S<sub>1</sub> B<sub>2</sub> D<sub>2</sub>

F0	4	0	****	0	03F
----	---	---	------	---	-----

0011 1111  
 6-bit two's complement for -1

FIELD2 (before): 01 23 45 67 8C  
 FIELD2 (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.

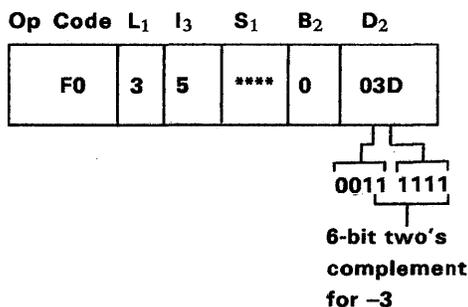
#### Decimal Right Shift and Round

In this example, the contents of storage location FIELD3 are shifted three places to the right and rounded, effectively dividing by 1000 and rounding to the nearest whole number. FIELD3 is four bytes in length.

### Assembler Format

Op Code S<sub>1</sub> L<sub>1</sub> S<sub>2</sub> I<sub>3</sub>  
 -----  
 SRP FIELD3(4),64 - 3,5

### Machine Format



FIELD3 (before): 12 39 60 0D  
 FIELD3 (after): 00 01 24 0D

The shift amount (three places) is specified in the D<sub>2</sub> field. The I<sub>3</sub> field specifies the rounding factor of 5. The rounding factor 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 designated 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 variable, it may be specified by the B<sub>2</sub> field instead of the displacement D<sub>2</sub> of the SRP instruction. The general register designated by B<sub>2</sub> 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 B<sub>2</sub> field (variable part in a general register) and the D<sub>2</sub> field (fixed part in the displacement).

The SRP instruction uses only the rightmost six bits of the effective address D<sub>2</sub>(B<sub>2</sub>) and interprets them as a six-bit signed binary integer to control the left or right shift as in the previous two 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.

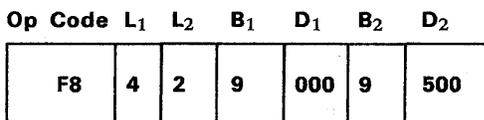
Storage locations 4500-4502 contain 38 46 0D.

After the instruction

### Assembler Format

Op Code D<sub>1</sub>L<sub>1</sub>B<sub>1</sub> D<sub>2</sub> L<sub>2</sub>B<sub>2</sub>  
 -----  
 ZAP 0(5,9),X'500'(3,9)

### Machine Format



is executed, the storage locations 4000-4004 contain 00 00 38 46 0D; condition code 1 is set to indicate a negative result.

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.

## Floating-Point Instructions

(See Chapter 9.)

In this section, the abbreviations FPR0, FPR2, FPR4, and FPR6 stand for floating-point registers 0, 2, 4, and 6 respectively.

### ADD NORMALIZED (AE, AER, AD, ADR)

The ADD NORMALIZED instructions perform the addition of two floating-point numbers and place the normalized result in a floating-point register. Neither of the two numbers to be added must necessarily be normalized before addition occurs. For example, assume that:



### Assembler Format

Op Code	R <sub>1</sub> R <sub>2</sub>
CDR	4,6

### Machine Format

Op Code	R <sub>1</sub>	R <sub>2</sub>
29	4	6

The number with the smaller characteristic, which is the one in register FPR6, is right-shifted 15 hexadecimal digit positions so that the two characteristics agree. The shifted contents of FPR6 are 43 00 00 00 00 00 00 00, with a guard digit of zero. Therefore, when the two numbers are compared, condition code 0 is set, indicating an equality.

As the above example implies, when floating-point numbers are compared, more than two numbers may compare equal if one of the numbers is unnormalized. For example, the unnormalized floating-point number 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 low-order 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 floating-point numbers are compared, the relationship between numbers that compare equal is unique: each digit in one number must be identical to the corresponding digit in the other number.

## Multiprogramming and Multiprocessing Examples

Although the 4300 Processors make no provision for multiple-CPU systems, the references to multiprocessing may be helpful when writing problem-state programs that are to run correctly on multiprocessing configurations of System/370 as well.

When two or more programs sharing common storage locations are running 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 system 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 nearly 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
	X'01'	

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 date is now invalid.

The COMPARE AND SWAP instruction has been provided to overcome this and similar problems.

### COMPARE AND SWAP (CS, CDS)

The COMPARE AND SWAP and COMPARE DOUBLE AND SWAP 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 running enabled for interruption (multiprogramming) or by programs that are running on a multiprocessing configuration. That is, the routine allows a program to modify the contents of a storage location while

running 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 CS 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 CS, no other CPU can access the specified location.

### Setting a Single Bit

The following instruction sequence shows how the CS instruction can be used to update a single bit in storage. Assume that FLAGS is the first byte of a word in storage called "WORD."

LA	6,X'80'	Put bit to be ORed into register 6
SLL	6,24	Shift left 24 places to align the byte to be ORed with the location of FLAGS within WORD
L	5,WORD	Get original flag bit values
RETRY	LR 4,5	Put flags to be modified into register 4
	OR 4,6	Turn on bit in new copy of flags
	CS 5,4,WORD	Store new flags unless original flags were changed
	BC 4,RETRY	If new flags not stored, try again

The format of the CS instruction is:

#### Assembler Format

Op Code	R <sub>1</sub> R <sub>3</sub>	S <sub>2</sub>
-----		
CS	5,4,	WORD

#### Machine Format

Op Code	R <sub>1</sub>	R <sub>3</sub>	S <sub>2</sub>
---------	----------------	----------------	----------------

BA	5	4	****
----	---	---	------

The CS instruction compares the first operand (register 5 containing the original flag values) to the second operand (WORD) while storage access to the specified location is not permitted to any CPU other than the one executing the CS instruction.

If the comparison is successful, indicating that FLAGS still has the same value that it originally had, the modified copy in register 4 is stored into FLAGS. If FLAGS has changed since it was loaded, the compare will not be successful, and the current value of FLAGS is loaded into register 5.

The CS instruction sets condition code 0 to indicate a successful compare and swap, and condition code 1 to indicate an unsuccessful compare and swap.

The program executing the sample instructions tests the condition code following the CS instruction and reexecutes the flag-modifying instructions if the CS instruction indicated an unsuccessful comparison. When the CS instruction is successful, the program continues execution outside the loop and FLAGS contains valid data.

The branch to RETRY will be taken only if some other program modifies the update location. 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 one 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 CS 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 4. The original counter is moved into general register 5 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 CS 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 CS instruction, the CS instruction would have loaded the new counter value into general register 4 and set the condition code to 1, indicating an unsuccessful update. The program then must update the new counter value in general register 4 and retry the CS instruction,

retesting the condition code, and retrying until a successful update is completed.

The following instruction sequence performs the above procedure:

LA	6,1	Put increment (1) in GR6
L	4,CNTR	Put original counter value in GR4
LOOP LR	5,4	Set up copy in GR5 to modify
AR	5,6	Increment copy
CS	4,5,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.

<u>CPU A</u>			<u>CPU B</u>		<u>Comments</u>
<u>GR4</u>	<u>GR5</u>	<u>CNTR</u>	<u>GR4</u>	<u>GR5</u>	
		16			
16	16	16			CPU A loads GR4 and GR5 from CNTR
16	16	16	16	16	CPU B loads GR4 and GR5 from CNTR
		16	16	17	CPU B adds one to GR5
16	17	16			CPU A adds one to GR5
16	17	17			CPU A executes CS; successful match, store
		17	17	17	CPU B executes CS; no match, GR4 changed to CNTR value
			17	18	CPU B loads GR5 from GR4, adds one to GR5
		18	17	18	CPU B executes CS; successful match, store

## Appendix B. Lists of Instructions

The following three figures list instructions arranged by name, mnemonic, and operation code. Some processors may offer instructions that do not appear in the figures, such as those provided for emulation or as part of special or custom features.

The operation code 00, with a two-byte instruction format, and the set of sixteen 16-bit operation codes

B2E0 to B2EF, with a four-byte instruction format, are allocated for use by the program when an indication of an invalid operation is required. It is improbable that these operation codes will ever be assigned to an instruction implemented in the CPU.

### *Explanation of Symbols in "Characteristics" and "Op Code" Columns*

A	Access exceptions
AD	Addressing exception
AP	Addressing and protection exceptions
B	PER branch event
C	Condition code is set
D	Data exception
DF	Decimal-overflow exception
DK	Decimal-divide exception
DM	DIAGNOSE may generate various program exceptions and may change the condition code
EO	Exponent-overflow exception
EX	Execute exception
FK	Floating-point-divide exception
IF	Fixed-point-overflow exception
II	Interruptible instruction
IK	Fixed-point-divide exception
L	New condition code loaded
LS	Significance exception
M	Privileged-operation exception
MO	Monitor event
PS	Page-state exception
PT	Page-transition exception
R	PER general-register-alteration event
RR	RR instruction format
RS	RS instruction format
RX	RX instruction format
S	S instruction format
SI	SI instruction format
SO	Special-operation exception
SP	Specification exception
SS	SS instruction format
ST	PER storage-alteration event
U	Exponent-underflow exception
*	Bits 8-14 of the operation code are ignored
‡	Bits 8-15 of the operation code are ignored
\$	Causes serialization
\$ <sup>1</sup>	Causes serialization when the R <sub>1</sub> and R <sub>2</sub> fields are 15 and zero, respectively

\* New Instructions  
(8)  
(Dropped instructions: 8)

Name	Mnemonic	Characteristics						Op Code
ADD	AR	RR	C			IF	R	1A
ADD	A	RX	C	A		IF	R	5A
ADD DECIMAL	AP	SS	C	A	D	DF	ST	FA
ADD HALFWORD	AH	RX	C	A		IF	R	4A
ADD LOGICAL	ALR	RR	C				R	1E
ADD LOGICAL	AL	RX	C	A			R	5E
ADD NORMALIZED (extended)	AXR	RR	C		SP	U EO	LS	36
ADD NORMALIZED (long)	ADR	RR	C		SP	U EO	LS	2A
ADD NORMALIZED (long)	AD	RX	C	A	SP	U EO	LS	6A
ADD NORMALIZED (short)	AER	RR	C		SP	U EO	LS	3A
ADD NORMALIZED (short)	AE	RX	C	A	SP	U EO	LS	7A
ADD UNNORMALIZED (long)	AWR	RR	C		SP	EO	LS	2E
ADD UNNORMALIZED (long)	AW	RX	C	A	SP	EO	LS	6E
ADD UNNORMALIZED (short)	AUR	RR	C		SP	EO	LS	3E
ADD UNNORMALIZED (short)	AU	RX	C	A	SP	EO	LS	7E
AND	NR	RR	C				R	14
AND	N	RX	C	A			R	54
AND (character)	NC	SS	C	A			ST	D4
AND (immediate)	NI	SI	C	A			ST	94
BRANCH AND LINK	BALR	RR					B R	05
BRANCH AND LINK	BAL	RX					B R	45
BRANCH ON CONDITION	BCR	RR				\$ <sup>1</sup>	B	07
BRANCH ON CONDITION	BC	RX					B	47
BRANCH ON COUNT	BCTR	RR					B R	06
BRANCH ON COUNT	BCT	RX					B R	46
BRANCH ON INDEX HIGH	BXH	RS					B R	86
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS					B R	87
CLEAR I/O	CLRIO	S	C	M		\$		9D01*
CLEAR PAGE	CLRP	S		M	AP	PS	ST	B215
COMPARE	CR	RR	C					19
COMPARE	C	RX	C	A				59
COMPARE (long)	CDR	RR	C		SP			29
COMPARE (long)	CD	RX	C	A	SP			69
COMPARE (short)	CER	RR	C		SP			39
COMPARE (short)	CE	RX	C	A	SP			79
COMPARE AND SWAP	CS	RS	C	A	SP	\$	R ST	BA
COMPARE DECIMAL	CP	SS	C	A	D			F9
COMPARE DOUBLE AND SWAP	CDS	RS	C	A	SP	\$	R ST	BB
COMPARE HALFWORD	CH	RX	C	A				49
COMPARE LOGICAL	CLR	RR	C					15
COMPARE LOGICAL	CL	RX	C	A				55
COMPARE LOGICAL (character)	CLC	SS	C	A				D5
COMPARE LOGICAL (immediate)	CLI	SI	C	A				95
COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS	C	A				BD
COMPARE LOGICAL LONG	CLCL	RR	C	A	SP	II	R	0F
CONNECT PAGE	CTP	RS	C	M	AD SP		PT R	B0
CONVERT TO BINARY	CVB	RX		A	D	IK	R	4F
CONVERT TO DECIMAL	CVD	RX		A			ST	4E
DECONFIGURE PAGE	DEP	S		M	AD SP		PT	B21B
DIAGNOSE				M	DM			83
* DISCONNECT PAGE	DCTP	S	C	M	AD SP		PT	B21C
DIVIDE	DR	RR			SP	IK	R	1D
DIVIDE	D	RX		A	SP	IK	R	5D
DIVIDE (long)	DDR	RR			SP U	EO FK		2D
DIVIDE (long)	DD	RX		A	SP U	EO FK		6D

Instructions Arranged by Name (Part 1 of 4)

\* new

Name	Mnemonic	Characteristics										Op Code	
DIVIDE (short)	DER	RR				SP	U	EO	FK			3D	
DIVIDE (short)	DE	RX				A	SP	U	EO	FK		7D	
DIVIDE DECIMAL	DP	SS				A	SP	D		DK	ST	FD	
EDIT	ED	SS	C			A		D			ST	DE	
EDIT AND MARK	EDMK	SS	C			A		D			R	ST	DF
EXCLUSIVE OR	XR	RR	C								R		17
EXCLUSIVE OR	X	RX	C			A					R		57
EXCLUSIVE OR (character)	XC	SS	C			A						ST	D7
EXCLUSIVE OR (immediate)	XI	SI	C			A						ST	97
EXECUTE	EX	RX				A	SP				EX		44
HALT DEVICE	HDV	S	C	M							\$		9E01*
HALT I/O	HIO	S	C	M							\$		9E00*
HALVE (long)	HDR	RR					SP	U					24
HALVE (short)	HER	RR					SP	U					34
INSERT CHARACTER	IC	RX				A					R		43
INSERT CHARACTERS UNDER MASK	ICM	RS	C			A					R		BF
INSERT PAGE BITS	IPB	RS		M	AD						R		B4
INSERT PSW KEY	IPK	S		M							R		B20B
INSERT STORAGE KEY	ISK	RR		M	AD	SP					R		09
LOAD	LR	RR									R		18
LOAD	L	RX				A					R		58
LOAD (long)	LDR	RR					SP						28
LOAD (long)	LD	RX				A	SP						68
LOAD (short)	LER	RR					SP						38
LOAD (short)	LE	RX				A	SP						78
LOAD ADDRESS	LA	RX									R		41
LOAD AND TEST	LTR	RR	C								R		12
LOAD AND TEST (long)	LTDR	RR	C				SP						22
LOAD AND TEST (short)	LTER	RR	C				SP						32
LOAD COMPLEMENT	LCR	RR	C						IF		R		13
LOAD COMPLEMENT (long)	LCDR	RR	C				SP						23
LOAD COMPLEMENT (short)	LCER	RR	C				SP						33
LOAD CONTROL	LCTL	RS		M	A	SP							B7
LOAD FRAME INDEX	LF1	RS	C	M							R		B8
LOAD HALFWORD	LH	RX				A					R		48
LOAD MULTIPLE	LM	RS				A					R		98
LOAD NEGATIVE	LNR	RR	C								R		11
LOAD NEGATIVE (long)	LNDR	RR	C				SP						21
LOAD NEGATIVE (short)	LNER	RR	C				SP						31
LOAD POSITIVE	LPR	RR	C						IF		R		10
LOAD POSITIVE (long)	LPDR	RR	C				SP						20
LOAD POSITIVE (short)	LPER	RR	C				SP						30
LOAD PSW	LPSW	S	L	M	A	SP					\$		82
LOAD ROUNDED (extended to long)	LRDR	RR					SP		EO				25
LOAD ROUNDED (long to short)	LRER	RR					SP		EO				35
MAKE ADDRESSABLE	MAD	S	C	M	AD						PT		B21D
MAKE UNADDRESSABLE	MUN	S	C	M	AD	SP					PT		B21E
MONITOR CALL	MC	SI				SP					MO		AF
MOVE (character)	MVC	SS				A						ST	D2
MOVE (immediate)	MVI	SI				A						ST	92
MOVE INVERSE	MVCIN	SS				A						ST	E8
MOVE LONG	MVCL	RR	C			A	SP			II	R	ST	0E
MOVE NUMERICS	MVN	SS				A						ST	D1
MOVE WITH OFFSET	MVO	SS				A						ST	F1
MOVE ZONES	MVZ	SS				A						ST	D3

*Load Real Address*

Instructions Arranged by Name (Part 2 of 4)

Name	Mnemonic	Characteristics							Op Code	
MULTIPLY	MR	RR			SP			R	1C	
MULTIPLY	M	RX		A	SP			R	5C	
MULTIPLY (extended)	MXR	RR			SP	U	EO		26	
MULTIPLY (long)	MDR	RR			SP	U	EO		2C	
MULTIPLY (long)	MD	RX		A	SP	U	EO		6C	
MULTIPLY (long to extended)	MXDR	RR			SP	U	EO		27	
MULTIPLY (long to extended)	MXD	RX		A	SP	U	EO		67	
MULTIPLY (short to long)	MER	RR			SP	U	EO		3C	
MULTIPLY (short to long)	ME	RX		A	SP	U	EO		7C	
MULTIPLY DECIMAL	MP	SS		A	SP	D			ST	FC
MULTIPLY HALFWORD	MH	RX		A				R		4C
OR	OR	RR	C					R		16
OR	O	RX	C	A				R		56
OR (character)	OC	SS	C	A					ST	D6
OR (immediate)	OI	SI	C	A					ST	96
PACK	PACK	SS			A				ST	F2
RESET REFERENCE BIT	RRB	S	C	M	AD					B213
* RETRIEVE STATUS AND PAGE	RSP	SS	C	M	A				ST	D8
SET CLOCK	SCK	S	C	M	A	SP				B204
SET CLOCK COMPARATOR	SCKC	S		M	A	SP				B206
SET CPU TIMER	SPT	S		M	A	SP				B208
* SET PAGE BITS	SPB	RS	C	M	AD					B5
SET PROGRAM MASK	SPM	RR	L							04
SET PSW KEY FROM ADDRESS	SPKA	S		M						B20A
SET STORAGE KEY	SSK	RR		M	AD	SP				08
SET SYSTEM MASK	SSM	S		M	A	SP		SO		80
SHIFT AND ROUND DECIMAL	SRP	SS	C		A	D	DF		ST	F0
SHIFT LEFT DOUBLE	SLDA	RS	C			SP	IF		R	8F
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS				SP			R	8D
SHIFT LEFT SINGLE	SLA	RS	C				IF		R	8B
SHIFT LEFT SINGLE LOGICAL	SLL	RS							R	89
SHIFT RIGHT DOUBLE	SRDA	RS	C			SP			R	8E
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS				SP			R	8C
SHIFT RIGHT SINGLE	SRA	RS	C						R	8A
SHIFT RIGHT SINGLE LOGICAL	SRL	RS							R	88
START I/O	SIO	S	C	M				\$		9C00*
START I/O FAST RELEASE	SIOF	S	C	M				\$		9C01*
STORE	ST	RX			A				ST	50
STORE (long)	STD	RX			A	SP			ST	60
STORE (short)	STE	RX			A	SP			ST	70
STORE CAPACITY COUNTS	STCAP	S		M	A				ST	B21F
STORE CHANNEL ID	STIDC	S	C	M				\$		B203
STORE CHARACTER	STC	RX			A				ST	42
STORE CHARACTERS UNDER MASK	STCM	RS			A				ST	BE
STORE CLOCK	STCK	S	C		A			\$	ST	B205
STORE CLOCK COMPARATOR	STCKC	S		M	A	SP			ST	B207
STORE CONTROL	STCTL	RS		M	A	SP			ST	B6
STORE CPU ID	STIDP	S		M	A	SP			ST	B202
STORE CPU TIMER	STPT	S		M	A	SP			ST	B209
STORE HALFWORD	STH	RX			A				ST	40
STORE MULTIPLE	STM	RS			A				ST	90
STORE THEN AND SYSTEM MASK	STNSM	SI		M	A				ST	AC
STORE THEN OR SYSTEM MASK	STOSM	SI		M	A	SP			ST	AD
SUBTRACT	SR	RR	C				IF		R	1B
SUBTRACT	S	RX	C		A		IF		R	5B

*PURGE TLB, READ DIRECT*

*Set Prefix*

*Signal Present*

*Store CPU address*

*Store Prefix*

Instructions Arranged by Name (Part 3 of 4)

Name	Mnemonic	Characteristics						Op Code
SUBTRACT DECIMAL	SP	SS	C	A	D	DF	ST	FB
SUBTRACT HALFWORD	SH	RX	C	A		IF	R	4B
SUBTRACT LOGICAL	SLR	RR	C				R	1F
SUBTRACT LOGICAL	SL	RX	C	A			R	5F
SUBTRACT NORMALIZED (extended)	SXR	RR	C		SP	U EO	LS	37
SUBTRACT NORMALIZED (long)	SDR	RR	C		SP	U EO	LS	2B
SUBTRACT NORMALIZED (long)	SD	RX	C	A	SP	U EO	LS	6B
SUBTRACT NORMALIZED (short)	SER	RR	C		SP	U EO	LS	3B
SUBTRACT NORMALIZED (short)	SE	RX	C	A	SP	U EO	LS	7B
SUBTRACT UNNORMALIZED (long)	SWR	RR	C		SP	EO	LS	2F
SUBTRACT UNNORMALIZED (long)	SW	RX	C	A	SP	EO	LS	6F
SUBTRACT UNNORMALIZED (short)	SUR	RR	C		SP	EO	LS	3F
SUBTRACT UNNORMALIZED (short)	SU	RX	C	A	SP	EO	LS	7F
SUPERVISOR CALL	SVC	RR	L				\$	0A
TEST AND SET	TS	S	C	A			\$	ST 93
TEST CHANNEL	TCH	S	C	M			\$	9F00#
TEST I/O	TIO	S	C	M			\$	9D00*
TEST UNDER MASK	TM	SI	C	A				91
TRANSLATE	TR	SS		A				DC
TRANSLATE AND TEST	TRT	SS	C	A			R	DD
<del>UNPACK</del> <i>write Direct</i>	UNPK	SS		A				ST F3
ZERO AND ADD	ZAP	SS	C	A	D	DF		ST F8

Instructions Arranged by Name (Part 4 of 4)

Mnemonic	Name	Characteristics										Op Code
	DIAGNOSE		M	DM								83
A	ADD	RX C		A			IF				R	5A
AD	ADD NORMALIZED (long)	RX C		A	SP	U	EO	LS				6A
ADR	ADD NORMALIZED (long)	RR C			SP	U	EO	LS				2A
AE	ADD NORMALIZED (short)	RX C		A	SP	U	EO	LS				7A
AER	ADD NORMALIZED (short)	RR C			SP	U	EO	LS				3A
AH	ADD HALFWORD	RX C		A			IF				R	4A
AL	ADD LOGICAL	RX C		A							R	5E
ALR	ADD LOGICAL	RR C									R	1E
AP	ADD DECIMAL	SS C		A		D	DF				ST	FA
AR	ADD	RR C					IF				R	1A
AU	ADD UNNORMALIZED (short)	RX C		A	SP		EO	LS				7E
AUR	ADD UNNORMALIZED (short)	RR C			SP		EO	LS				3E
AW	ADD UNNORMALIZED (long)	RX C		A	SP		EO	LS				6E
AWR	ADD UNNORMALIZED (long)	RR C			SP		EO	LS				2E
AXR	ADD NORMALIZED (extended)	RR C			SP	U	EO	LS				36
BAL	BRANCH AND LINK	RX								B	R	45
BALR	BRANCH AND LINK	RR								B	R	05
BC	BRANCH ON CONDITION	RX								B		47
BCR	BRANCH ON CONDITION	RR						\$ <sup>1</sup>		B		07
BCT	BRANCH ON COUNT	RX								B	R	46
BCTR	BRANCH ON COUNT	RR								B	R	06
BXH	BRANCH ON INDEX HIGH	RS								B	R	86
BXLE	BRANCH ON INDEX LOW OR EQUAL	RS								B	R	87
C	COMPARE	RX C		A								59
CD	COMPARE (long)	RX C		A	SP							69
CDR	COMPARE (long)	RR C			SP							29
CDS	COMPARE DOUBLE AND SWAP	RS C		A	SP			\$		R	ST	BB
CE	COMPARE (short)	RX C		A	SP							79
CER	COMPARE (short)	RR C			SP							39
CH	COMPARE HALFWORD	RX C		A								49
CL	COMPARE LOGICAL	RX C		A								55
CLC	COMPARE LOGICAL (character)	SS C		A								D5
CLCL	COMPARE LOGICAL LONG	RR C		A	SP			II		R		0F
CLI	COMPARE LOGICAL (immediate)	SI C		A								95
CLM	COMPARE LOGICAL CHARACTERS UNDER MASK	RS C		A								BD
CLR	COMPARE LOGICAL	RR C										15
CLRIO	CLEAR I/O	S C	M					\$				9D01*
CLRP	CLEAR PAGE	S	M	AP				PS			ST	B215
CP	COMPARE DECIMAL	SS C		A		D						F9
CR	COMPARE	RR C										19
CS	COMPARE AND SWAP	RS C		A	SP			\$		R	ST	BA
CTP	CONNECT PAGE	RS C	M	AD	SP			PT		R		B0
CVB	CONVERT TO BINARY	RX		A		D		IK		R		4F
CVD	CONVERT TO DECIMAL	RX		A							ST	4E
D	DIVIDE	RX		A	SP			IK		R		5D
DCTP	DISCONNECT PAGE	S C	M	AD	SP			PT				B21C
DD	DIVIDE (long)	RX		A	SP	U	EO	FK				6D
DDR	DIVIDE (long)	RR			SP	U	EO	FK				2D
DE	DIVIDE (short)	RX		A	SP	U	EO	FK				7D
DEP	DECONFIGURE PAGE	S	M	AD	SP			PT				B21B
DER	DIVIDE (short)	RR			SP	U	EO	FK				3D
DP	DIVIDE DECIMAL	SS		A	SP	D		DK			ST	FD
DR	DIVIDE	RR			SP			IK		R		1D
ED	EDIT	SS C		A		D					ST	DE

Instructions Arranged by Mnemonic (Part 1 of 4)

Mnemonic	Name	Characteristics								Op Code	
EDMK	EDIT AND MARK	SS	C	A	D			R	ST	DF	
EX	EXECUTE	RX		A	SP		EX			44	
HDR	HALVE (long)	RR			SP	U				24	
HDV	HALT DEVICE	S	C	M			\$			9E01*	
HER	HALVE (short)	RR			SP	U				34	
HIO	HALT I/O	S	C	M			\$			9E00*	
IC	INSERT CHARACTER	RX		A				R		43	
ICM	INSERT CHARACTERS UNDER MASK	RS	C	A				R		BF	
IPB	INSERT PAGE BITS	RS		M	AD			R		B4	
IPK	INSERT PSW KEY	S		M				R		B20B	
ISK	INSERT STORAGE KEY	RR		M	AD	SP		R		09	
L	LOAD	RX		A				R		58	
LA	LOAD ADDRESS	RX						R		41	
LCDR	LOAD COMPLEMENT (long)	RR	C			SP				23	
LCER	LOAD COMPLEMENT (short)	RR	C			SP				33	
LCR	LOAD COMPLEMENT	RR	C				IF	R		13	
LCTL	LOAD CONTROL	RS		M	A	SP				B7	
LD	LOAD (long)	RX			A	SP				68	
LDR	LOAD (long)	RR				SP				28	
LE	LOAD (short)	RX			A	SP				78	
LER	LOAD (short)	RR				SP				38	
LFI	LOAD FRAME INDEX	RS	C	M				R		B8	
LH	LOAD HALFWORD	RX			A			R		48	
LM	LOAD MULTIPLE	RS			A			R		98	
LNDR	LOAD NEGATIVE (long)	RR	C			SP				21	
LNER	LOAD NEGATIVE (short)	RR	C			SP				31	
LNR	LOAD NEGATIVE	RR	C					R		11	
LPDR	LOAD POSITIVE (long)	RR	C			SP				20	
LPER	LOAD POSITIVE (short)	RR	C			SP				30	
LPR	LOAD POSITIVE	RR	C				IF	R		10	
LPSW	LOAD PSW	S	L	M	A	SP	\$			82	
LR	LOAD	RR						R		18	
LRDR	LOAD ROUNDED (extended to long)	RR				SP	EO			25	
LRER	LOAD ROUNDED (long to short)	RR				SP	EO			35	
LTDR	LOAD AND TEST (long)	RR	C			SP				22	
LTER	LOAD AND TEST (short)	RR	C			SP				32	
LTR	LOAD AND TEST	RR	C					R		12	
M	MULTIPLY	RX			A	SP		R		5C	
MAD	MAKE ADDRESSABLE	S	C	M	AD			PT		B21D	
MC	MONITOR CALL	SI				SP		MO		AF	
MD	MULTIPLY (long)	RX			A	SP	U	EO		6C	
MDR	MULTIPLY (long)	RR				SP	U	EO		2C	
ME	MULTIPLY (short to long)	RX			A	SP	U	EO		7C	
MER	MULTIPLY (short to long)	RR				SP	U	EO		3C	
MH	MULTIPLY HALFWORD	RX			A			R		4C	
MP	MULTIPLY DECIMAL	SS			A	SP	D		ST	FC	
MR	MULTIPLY	RR				SP		R		1C	
MUN	MAKE UNADDRESSABLE	S	C	M	AD	SP		PT		B21E	
MVC	MOVE (character)	SS			A				ST	D2	
MVCIN	MOVE INVERSE	SS			A				ST	E8	
MVCL	MOVE LONG	RR	C		A	SP		II	R	ST	0E
MVI	MOVE (immediate)	SI			A				ST	92	
MVN	MOVE NUMERICS	SS			A				ST	D1	
MVO	MOVE WITH OFFSET	SS			A				ST	F1	
MVZ	MOVE ZONES	SS			A				ST	D3	

Instructions Arranged by Mnemonic (Part 2 of 4)

Mnemonic	Name	Characteristics										Op Code
MXD	MULTIPLY (long to extended)	RX		A	SP	U	EO					67
MXDR	MULTIPLY (long to extended)	RR			SP	U	EO					27
MXR	MULTIPLY (extended)	RR			SP	U	EO					26
N	AND	RX	C	A						R		54
NC	AND (character)	SS	C	A							ST	D4
NI	AND (immediate)	SI	C	A							ST	94
NR	AND	RR	C							R		14
O	OR	RX	C	A						R		56
OC	OR (character)	SS	C	A							ST	D6
OI	OR (immediate)	SI	C	A							ST	96
OR	OR	RR	C							R		16
PACK	PACK	SS		A							ST	F2
RRB	RESET REFERENCE BIT	S	C	M	AD							B213
RSP	RETRIEVE STATUS AND PAGE	SS	C	M	A						ST	D8
S	SUBTRACT	RX	C	A			IF			R		5B
SCK	SET CLOCK	S	C	M	A	SP						B204
SCKC	SET CLOCK COMPARATOR	S		M	A	SP						B206
SPB	SET PAGE BITS	RS	C	M	AD							B5
SD	SUBTRACT NORMALIZED (long)	RX	C		A	SP	U	EO	LS			6B
SDR	SUBTRACT NORMALIZED (long)	RR	C			SP	U	EO	LS			2B
SE	SUBTRACT NORMALIZED (short)	RX	C		A	SP	U	EO	LS			7B
SER	SUBTRACT NORMALIZED (short)	RR	C			SP	U	EO	LS			3B
SH	SUBTRACT HALFWORD	RX	C		A			IF		R		4B
SIO	START I/O	S	C	M					\$			9C00*
SIOF	START I/O FAST RELEASE	S	C	M					\$			9C01*
SL	SUBTRACT LOGICAL	RX	C		A					R		5F
SLA	SHIFT LEFT SINGLE	RS	C					IF		R		8B
SLDA	SHIFT LEFT DOUBLE	RS	C			SP		IF		R		8F
SLDL	SHIFT LEFT DOUBLE LOGICAL	RS				SP				R		8D
SLL	SHIFT LEFT SINGLE LOGICAL	RS								R		89
SLR	SUBTRACT LOGICAL	RR	C							R		1F
SP	SUBTRACT DECIMAL	SS	C		A		D	DF			ST	FB
SPKA	SET PSW KEY FROM ADDRESS	S		M								B20A
SPM	SET PROGRAM MASK	RR	L									04
SPT	SET CPU TIMER	S		M	A	SP						B208
SR	SUBTRACT	RR	C					IF		R		1B
SRA	SHIFT RIGHT SINGLE	RS	C							R		8A
SRDA	SHIFT RIGHT DOUBLE	RS	C			SP				R		8E
SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS				SP				R		8C
SRL	SHIFT RIGHT SINGLE LOGICAL	RS								R		88
SRP	SHIFT AND ROUND DECIMAL	SS	C		A		D	DF			ST	F0
SSK	SET STORAGE KEY	RR		M	AD	SP						08
SSM	SET SYSTEM MASK	S		M	A	SP			SO			80
ST	STORE	RX			A						ST	50
STC	STORE CHARACTER	RX			A						ST	42
STCAP	STORE CAPACITY COUNTS	S		M	A						ST	B21F
STCK	STORE CLOCK	S	C		A				\$		ST	B205
STCKC	STORE CLOCK COMPARATOR	S		M	A	SP					ST	B207
STCM	STORE CHARACTERS UNDER MASK	RS			A						ST	BE
STCTL	STORE CONTROL	RS		M	A	SP					ST	B6
STD	STORE (long)	RX			A	SP					ST	60
STE	STORE (short)	RX			A	SP					ST	70
STH	STORE HALFWORD	RX			A						ST	40
STIDC	STORE CHANNEL ID	S	C	M					\$			B203
STIDP	STORE CPU ID	S		M	A	SP					ST	B202

Instructions Arranged by Mnemonic (Part 3 of 4)

<u>Mnemonic</u>	<u>Name</u>	<u>Characteristics</u>						<u>Op Code</u>
STM	STORE MULTIPLE	RS		A			ST	90
STNSM	STORE THEN AND SYSTEM MASK	SI	M	A			ST	AC
STOSM	STORE THEN OR SYSTEM MASK	SI	M	A	SP		ST	AD
STPT	STORE CPU TIMER	S	M	A	SP		ST	B209
SU	SUBTRACT UNNORMALIZED (short)	RX	C	A	SP	EO LS		7F
SUR	SUBTRACT UNNORMALIZED (short)	RR	C		SP	EO LS		3F
SVC	SUPERVISOR CALL	RR	L			\$		0A
SW	SUBTRACT UNNORMALIZED (long)	RX	C	A	SP	EO LS		6F
SWR	SUBTRACT UNNORMALIZED (long)	RR	C		SP	EO LS		2F
SXR	SUBTRACT NORMALIZED (extended)	RR	C		SP U	EO LS		37
TCH	TEST CHANNEL	S	C	M		\$		9F00‡
TIO	TEST I/O	S	C	M		\$		9D00*
TM	TEST UNDER MASK	SI	C	A				91
TR	TRANSLATE	SS		A			ST	DC
TRT	TRANSLATE AND TEST	SS	C	A			R	DD
TS	TEST AND SET	S	C	A		\$	ST	93
UNPK	UNPACK	SS		A			ST	F3
X	EXCLUSIVE OR	RX	C	A			R	57
XC	EXCLUSIVE OR (character)	SS	C	A			ST	D7
XI	EXCLUSIVE OR (immediate)	SI	C	A			ST	97
XR	EXCLUSIVE OR	RR	C				R	17
ZAP	ZERO AND ADD	SS	C	A	D	DF	ST	F8

Instructions Arranged by Mnemonic (Part 4 of 4)

Op Code	Name	Mnemonic	Characteristics																	
04	SET PROGRAM MASK	SPM	RR	L																
05	BRANCH AND LINK	BALR	RR																	
06	BRANCH ON COUNT	BCTR	RR																	B R
07	BRANCH ON CONDITION	BCR	RR																	B R
08	SET STORAGE KEY	SSK	RR		M	AD	SP													\$ <sup>1</sup> B
09	INSERT STORAGE KEY	ISK	RR		M	AD	SP													R
0A	SUPERVISOR CALL	SVC	RR	L																\$
0E	MOVE LONG	MVCL	RR	C		A	SP													II R ST
0F	COMPARE LOGICAL LONG	CLCL	RR	C		A	SP													II R
10	LOAD POSITIVE	LPR	RR	C							IF									R
11	LOAD NEGATIVE	LNR	RR	C																R
12	LOAD AND TEST	LTR	RR	C																R
13	LOAD COMPLEMENT	LCR	RR	C							IF									R
14	AND	NR	RR	C																R
15	COMPARE LOGICAL	CLR	RR	C																
16	OR	OR	RR	C																R
17	EXCLUSIVE OR	XR	RR	C																R
18	LOAD	LR	RR																	R
19	COMPARE	CR	RR	C																
1A	ADD	AR	RR	C							IF									R
1B	SUBTRACT	SR	RR	C							IF									R
1C	MULTIPLY	MR	RR				SP													R
1D	DIVIDE	DR	RR				SP					IK								R
1E	ADD LOGICAL	ALR	RR	C																R
1F	SUBTRACT LOGICAL	SLR	RR	C																R
20	LOAD POSITIVE (long)	LPDR	RR	C			SP													
21	LOAD NEGATIVE (long)	LNDR	RR	C			SP													
22	LOAD AND TEST (long)	LTDR	RR	C			SP													
23	LOAD COMPLEMENT (long)	LCDR	RR	C			SP													
24	HALVE (long)	HDR	RR				SP	U												
25	LOAD ROUNDED (extended to long)	LRDR	RR				SP		EO											
26	MULTIPLY (extended)	MXR	RR				SP	U	EO											
27	MULTIPLY (long to extended)	MXDR	RR				SP	U	EO											
28	LOAD (long)	LDR	RR				SP													
29	COMPARE (long)	CDR	RR	C			SP													
2A	ADD NORMALIZED (long)	ADR	RR	C			SP	U	EO	LS										
2B	SUBTRACT NORMALIZED (long)	SDR	RR	C			SP	U	EO	LS										
2C	MULTIPLY (long)	MDR	RR				SP	U	EO											
2D	DIVIDE (long)	DDR	RR				SP	U	EO	FK										
2E	ADD UNNORMALIZED (long)	AWR	RR	C			SP		EO	LS										
2F	SUBTRACT UNNORMALIZED (long)	SWR	RR	C			SP		EO	LS										
30	LOAD POSITIVE (short)	LPER	RR	C			SP													
31	LOAD NEGATIVE (short)	LNER	RR	C			SP													
32	LOAD AND TEST (short)	LTER	RR	C			SP													
33	LOAD COMPLEMENT (short)	LCER	RR	C			SP													
34	HALVE (short)	HER	RR				SP	U												
35	LOAD ROUNDED (long to short)	LRER	RR				SP		EO											
36	ADD NORMALIZED (extended)	AXR	RR	C			SP	U	EO	LS										
37	SUBTRACT NORMALIZED (extended)	SXR	RR	C			SP	U	EO	LS										
38	LOAD (short)	LER	RR				SP													
39	COMPARE (short)	CER	RR	C			SP													
3A	ADD NORMALIZED (short)	AER	RR	C			SP	U	EO	LS										
3B	SUBTRACT NORMALIZED (short)	SER	RR	C			SP	U	EO	LS										
3C	MULTIPLY (short to long)	MER	RR				SP	U	EO											
3D	DIVIDE (short)	DER	RR				SP	U	EO	FK										

Instructions Arranged by Operation Code (Part 1 of 4)

Op Code	Name	Mnemonic	Characteristics					
3E	ADD UNNORMALIZED (short)	AUR	RR	C	SP	EO	LS	
3F	SUBTRACT UNNORMALIZED (short)	SUR	RR	C	SP	EO	LS	
40	STORE HALFWORD	STH	RX		A			ST
41	LOAD ADDRESS	LA	RX					R
42	STORE CHARACTER	STC	RX		A			ST
43	INSERT CHARACTER	IC	RX		A			R
44	EXECUTE	EX	RX		A	SP	EX	
45	BRANCH AND LINK	BAL	RX					B R
46	BRANCH ON COUNT	BCT	RX					B R
47	BRANCH ON CONDITION	BC	RX					B
48	LOAD HALFWORD	LH	RX		A			R
49	COMPARE HALFWORD	CH	RX	C	A			
4A	ADD HALFWORD	AH	RX	C	A		IF	R
4B	SUBTRACT HALFWORD	SH	RX	C	A		IF	R
4C	MULTIPLY HALFWORD	MH	RX		A			R
4E	CONVERT TO DECIMAL	CVD	RX		A			ST
4F	CONVERT TO BINARY	CVB	RX		A	D	IK	R
50	STORE	ST	RX		A			ST
54	AND	N	RX	C	A			R
55	COMPARE LOGICAL	CL	RX	C	A			
56	OR	O	RX	C	A			R
57	EXCLUSIVE OR	X	RX	C	A			R
58	LOAD	L	RX		A			R
59	COMPARE	C	RX	C	A			
5A	ADD	A	RX	C	A		IF	R
5B	SUBTRACT	S	RX	C	A		IF	R
5C	MULTIPLY	M	RX		A	SP		R
5D	DIVIDE	D	RX		A	SP	IK	R
5E	ADD LOGICAL	AL	RX	C	A			R
5F	SUBTRACT LOGICAL	SL	RX	C	A			R
60	STORE (long)	STD	RX		A	SP		ST
67	MULTIPLY (long to extended)	MXD	RX		A	SP	U	EO
68	LOAD (long)	LD	RX		A	SP		
69	COMPARE (long)	CD	RX	C	A	SP		
6A	ADD NORMALIZED (long)	AD	RX	C	A	SP	U	EO
6B	SUBTRACT NORMALIZED (long)	SD	RX	C	A	SP	U	EO
6C	MULTIPLY (long)	MD	RX		A	SP	U	EO
6D	DIVIDE (long)	DD	RX		A	SP	U	EO
6E	ADD UNNORMALIZED (long)	AW	RX	C	A	SP	EO	LS
6F	SUBTRACT UNNORMALIZED (long)	SW	RX	C	A	SP	EO	LS
70	STORE (short)	STE	RX		A	SP		ST
78	LOAD (short)	LE	RX		A	SP		
79	COMPARE (short)	CE	RX	C	A	SP		
7A	ADD NORMALIZED (short)	AE	RX	C	A	SP	U	EO
7B	SUBTRACT NORMALIZED (short)	SE	RX	C	A	SP	U	EO
7C	MULTIPLY (short to long)	ME	RX		A	SP	U	EO
7D	DIVIDE (short)	DE	RX		A	SP	U	EO
7E	ADD UNNORMALIZED (short)	AU	RX	C	A	SP	EO	LS
7F	SUBTRACT UNNORMALIZED (short)	SU	RX	C	A	SP	EO	LS
80	SET SYSTEM MASK	SSM	S		M	A	SP	SO
82	LOAD PSW	LPSW	S	L	M	A	SP	\$
83	DIAGNOSE				M	DM		
86	BRANCH ON INDEX HIGH	BXH	RS					B R
87	BRANCH ON INDEX LOW OR EQUAL	BXLE	RS					B R
88	SHIFT RIGHT SINGLE LOGICAL	SRL	RS					R

Instructions Arranged by Operation Code (Part 2 of 4)

Op Code	Name	Mnemonic	Characteristics					
89	SHIFT LEFT SINGLE LOGICAL	SLL	RS					R
8A	SHIFT RIGHT SINGLE	SRA	RS	C				R
8B	SHIFT LEFT SINGLE	SLA	RS	C			IF	R
8C	SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS			SP		R
8D	SHIFT LEFT DOUBLE LOGICAL	SLDL	RS			SP		R
8E	SHIFT RIGHT DOUBLE	SRDA	RS	C		SP		R
8F	SHIFT LEFT DOUBLE	SLDA	RS	C		SP	IF	R
90	STORE MULTIPLE	STM	RS		A			ST
91	TEST UNDER MASK	TM	SI	C	A			
92	MOVE (immediate)	MVI	SI		A			ST
93	TEST AND SET	TS	S	C	A		\$	ST
94	AND (immediate)	NI	SI	C	A			ST
95	COMPARE LOGICAL (immediate)	CLI	SI	C	A			
96	OR (immediate)	OI	SI	C	A			ST
97	EXCLUSIVE OR (immediate)	XI	SI	C	A			ST
98	LOAD MULTIPLE	LM	RS		A			R
9C00*	START I/O	SIO	S	C	M		\$	
9C01*	START I/O FAST RELEASE	SIOF	S	C	M		\$	
9D00*	TEST I/O	TIO	S	C	M		\$	
9D01*	CLEAR I/O	CLRIO	S	C	M		\$	
9E00*	HALT I/O	HIO	S	C	M		\$	
9E01*	HALT DEVICE	HDV	S	C	M		\$	
9F00‡	TEST CHANNEL	TCH	S	C	M		\$	
AC	STORE THEN AND SYSTEM MASK	STNSM	SI		M	A		ST
AD	STORE THEN OR SYSTEM MASK	STOSM	SI		M	A	SP	ST
AF	MONITOR CALL	MC	SI				SP	MO
B0	CONNECT PAGE	CTP	RS	C	M	AD	SP	PT
B202	STORE CPU ID	STIDP	S		M	A	SP	R
B203	STORE CHANNEL ID	STIDC	S	C	M			\$
B204	SET CLOCK	SCK	S	C	M	A	SP	
B205	STORE CLOCK	STCK	S	C		A		\$
B206	SET CLOCK COMPARATOR	SCKC	S		M	A	SP	
B207	STORE CLOCK COMPARATOR	STCKC	S		M	A	SP	ST
B208	SET CPU TIMER	SPT	S		M	A	SP	
B209	STORE CPU TIMER	STPT	S		M	A	SP	ST
B20A	SET PSW KEY FROM ADDRESS	SPKA	S		M			
B20B	INSERT PSW KEY	IPK	S		M			R
B213	RESET REFERENCE BIT	RRB	S	C	M	AD		
B215	CLEAR PAGE	CLRP	S		M	AP		PS
B21B	DECONFIGURE PAGE	DECP	S		M	AD	SP	PT
B21C	DISCONNECT PAGE	DCTP	S	C	M	AD	SP	PT
B21D	MAKE ADDRESSABLE	MAD	S	C	M	AD		PT
B21E	MAKE UNADDRESSABLE	MUN	S	C	M	AD	SP	PT
B21F	STORE CAPACITY COUNTS	STCAP	S		M	A		ST
B4	INSERT PAGE BITS	IPB	RS		M	AD		R
B5	SET PAGE BITS	SPB	RS	C	M	AD		
B6	STORE CONTROL	STCTL	RS		M	A	SP	ST
B7	LOAD CONTROL	LCTL	RS		M	A	SP	
B8	LOAD FRAME INDEX	LFI	RS	C	M			R
BA	COMPARE AND SWAP	CS	RS	C		A	SP	\$
BB	COMPARE DOUBLE AND SWAP	CDS	RS	C		A	SP	\$
BD	COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS	C		A		
BE	STORE CHARACTERS UNDER MASK	STCM	RS			A		ST
BF	INSERT CHARACTERS UNDER MASK	ICM	RS	C		A		R
D1	MOVE NUMERICS	MVN	SS			A		ST

Instructions Arranged by Operation Code (Part 3 of 4)

Op Code	Name	Mnemonic	Characteristics						
D2	MOVE (character)	MVC	SS		A				ST
D3	MOVE ZONES	MVZ	SS		A				ST
D4	AND (character)	NC	SS	C	A				ST
D5	COMPARE LOGICAL (character)	CLC	SS	C	A				ST
D6	OR (character)	OC	SS	C	A				ST
D7	EXCLUSIVE OR (character)	XC	SS	C	A				ST
D8	RETRIEVE STATUS AND PAGE	RSP	SS	C	M	A			ST
DC	TRANSLATE	TR	SS		A				ST
DD	TRANSLATE AND TEST	TRT	SS	C	A			R	ST
DE	EDIT	ED	SS	C	A	D			ST
DF	EDIT AND MARK	EDMK	SS	C	A	D		R	ST
E8	MOVE INVERSE	MVCIN	SS		A				ST
F0	SHIFT AND ROUND DECIMAL	SRP	SS	C	A	D	DF		ST
F1	MOVE WITH OFFSET	MVO	SS		A				ST
F2	PACK	PACK	SS		A				ST
F3	UNPACK	UNPK	SS		A				ST
F8	ZERO AND ADD	ZAP	SS	C	A	D	DF		ST
F9	COMPARE DECIMAL	CP	SS	C	A	D			ST
FA	ADD DECIMAL	AP	SS	C	A	D	DF		ST
FB	SUBTRACT	SP	SS	C	A	D	DF		ST
FC	MULTIPLY DECIMAL	MP	SS		A	SP	D		ST
FD	DIVIDE DECIMAL	DP	SS		A	SP	D	DK	ST

Instructions Arranged by Operation Code (Part 4 of 4)

## Appendix C. Condition-Code Settings

Instruction	Condition Code			
	0	1	2	3
ADD (and ADD HALFWORD)	zero	< zero	> zero	overflow
ADD DECIMAL	zero	zero	> zero	overflow
ADD LOGICAL	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	—	—
CLEAR I/O	no operation in progress	CSW stored	channel busy	not operational
COMPARE, COMPARE HALFWORD	equal	low	high	—
COMPARE AND SWAP	equal	not equal	—	—
COMPARE DECIMAL	equal	low	high	—
COMPARE DOUBLE AND SWAP	equal	not equal	—	—
COMPARE LOGICAL	equal	low	high	—
COMPARE LOGICAL CHARACTERS UNDER MASK	equal	low	high	—
COMPARE LOGICAL LONG	equal	low	high	—
CONNECT PAGE	successful	already disconnected	unsuccessful	—
DISCONNECT PAGE	successful	already disconnected	—	—
EDIT	zero	< zero	> zero	—
EDIT AND MARK	zero	< zero	> zero	—
EXCLUSIVE OR	zero	not zero	—	—
HALT DEVICE	interruption pending/busy	CSW stored	channel working	not operational
HALT I/O	interruption pending	CSW stored	burst op stopped	not operational
INSERT CHARACTERS UNDER MASK	all zeros	1st bit one	1st bit zero	—
LOAD AND TEST	zero	< zero	> zero	—
LOAD COMPLEMENT (fixed point)	zero	< zero	> zero	overflow
LOAD COMPLEMENT (floating point)	zero	< zero	> zero	—
LOAD FRAME INDEX	addressable	connected	disconnected	address invalid
LOAD NEGATIVE	zero	< zero	—	—
LOAD POSITIVE (fixed point)	zero	—	> zero	overflow
LOAD POSITIVE (floating point)	zero	—	> zero	—
MAKE ADDRESSABLE	successful	already addressable	—	—
MADE UNADDRESSABLE	successful	already connected	—	—
MOVE LONG	length equal	length low	length high	destr overlap
OR	zero	not zero	—	—
RESET REFERENCE BIT	R bit zero, C bit zero	R bit zero, C bit one	R bit one, C bit zero	R bit one, C bit one
RETRIEVE STATUS AND PAGE	valid	—	—	invalid
SET PAGE BITS	R bit zero, C bit zero	R bit zero, C bit one	R bit one, C bit zero	R bit one, C bit one
SET CLOCK	set	secure	—	not operational
SHIFT AND ROUND DECIMAL	zero	< zero	> zero	overflow
SHIFT LEFT DOUBLE	zero	< zero	> zero	overflow
SHIFT LEFT SINGLE	zero	< zero	> zero	overflow
SHIFT RIGHT DOUBLE	zero	< zero	> zero	—
SHIFT RIGHT SINGLE	zero	< zero	> zero	—
START I/O, START I/O FAST RELEASE	successful	CSW stored	busy	not operational
STORE CHANNEL ID	ID stored	CSW stored	busy	not operational

Condition-Code Settings (Part 1 of 2)

Instruction	Condition Code			
	0	1	2	3
STORE CLOCK	set	not set	error	not operational
SUBTRACT (and SUBTRACT HALFWORD)	zero	< zero	> zero	overflow
SUBTRACT DECIMAL	zero	< zero	> zero	overflow
SUBTRACT LOGICAL	—	not zero, no carry	zero, carry	not zero, carry
SUBTRACT NORMALIZED	zero	< zero	> zero	—
SUBTRACT UNNORMALIZED	zero	< zero	> zero	—
TEST AND SET	left zero	left one	—	—
TEST CHANNEL	available	interruption pending	burst mode	not operational
TEST I/O	available	CSW stored	busy	not operational
TEST UNDER MASK	all zeros	mixed	—	all ones
TRANSLATE AND TEST	zero	incomplete	complete	—
ZERO AND ADD	zero	< zero	> zero	overflow

**Explanation:**

> zero	Result is greater than zero
< zero	Result is less than zero
high	First operand compares high
low	First operand compares low
length	Length of first operand

**Note:** The condition code may also be changed by EXECUTE, LOAD PSW, SET PROGRAM MASK, SUPERVISOR CALL, and DIAGNOSE, and by an interruption.

**Condition-Code Settings (Part 2 of 2)**

## Appendix D. Table of Powers of 2

PLUS		MINUS	
1	0	1.0	
2	1	0.5	
4	2	0.25	
8	3	0.125	
16	4	0.0625	
32	5	0.03125	
64	6	0.015625	5
128	7	0.0078125	25
256	8	0.00390625	625
512	9	0.001953125	3125
1,024	10	0.0009765625	65625
2,048	11	0.00048828125	5
4,096	12	0.000244140625	25
8,192	13	0.0001220703125	125
16,384	14	0.00006103515625	5625
32,768	15	0.000030517578125	
65,536	16	0.0000152587890625	5
131,072	17	0.00000762939453125	25
262,144	18	0.000003814697265625	625
524,288	19	0.0000019073486328125	
1,048,576	20	0.00000095367431640625	
2,097,152	21	0.000000476837158203125	5
4,194,304	22	0.0000002384185791015625	25
8,388,608	23	0.00000011920928955078125	
16,777,216	24	0.000000059604644775390625	
33,554,432	25	0.0000000298023223876953125	
67,108,864	26	0.00000001490116119384765625	5
134,217,728	27	0.000000007450580596923828125	25
268,435,456	28	0.0000000037252902984619140625	625
536,870,912	29	0.00000000186264514923095703125	
1,073,741,824	30	0.000000000931322574615478515625	
2,147,483,648	31	0.0000000004656612873077392578125	5
4,294,967,296	32	0.00000000023283064365386962890625	25
8,589,934,592	33	0.000000000116415321826934814453125	125
17,179,869,184	34	0.0000000000582076609134674072265625	
34,359,738,368	35	0.00000000002910383045673370361328125	
68,719,476,736	36	0.000000000014551915228366851806640625	5
137,438,953,472	37	0.0000000000072759576141834259033203125	25
274,877,906,944	38	0.00000000000363797880709171295166015625	625
549,755,813,888	39	0.000000000001818989403545856475830078125	
1,099,511,627,776	40	0.0000000000009094947017729282379150390625	
2,199,023,255,552	41	0.00000000000045474735088646411895751953125	5
4,398,046,511,104	42	0.000000000000227373675443232059478759765625	25
8,796,093,022,208	43	0.0000000000001136868377216160297393798828125	125
17,592,186,044,416	44	0.00000000000005684341886080801486968994140625	
35,184,372,088,832	45	0.000000000000028421709430404007434844970703125	
70,368,744,177,664	46	0.0000000000000142108547152020037174224853515625	5
140,737,488,355,328	47	0.00000000000000710542735760100185871124267578125	25
281,474,976,710,656	48	0.000000000000003552713678800500929355621337890625	
562,949,953,421,312	49	0.0000000000000017763568394002504646778106689453125	
1,125,899,906,842,624	50	0.00000000000000088817841970012523233890533447265625	
2,251,799,813,685,248	51	0.000000000000000444089209850062616169452667236328125	5
4,503,599,627,370,496	52	0.0000000000000002220446049250313080847263336181640625	
9,007,199,254,740,992	53	0.00000000000000011102230246251565404236316680908203125	
18,014,398,509,481,984	54	0.00000000000000005551151231257827021181583404541015625	
36,028,797,018,963,968	55	0.0000000000000000277555756156289135105907917022705078125	
72,057,594,037,927,936	56	0.00000000000000001387778780781445675529539585113525390625	5
144,115,188,075,855,872	57	0.000000000000000006938893903907228377647697925567626953125	
288,230,376,151,711,744	58	0.0000000000000000034694469519536141888238489627838134765625	
576,460,752,303,423,488	59	0.00000000000000000173472347597680709441192448139190673828125	
1,152,921,504,606,846,976	60	0.000000000000000000867361737988403547205962240695953369140625	
2,305,843,009,213,693,952	61	0.0000000000000000004336808689942017736029811203479766845703125	5
4,611,686,018,427,387,904	62	0.00000000000000000021684043449710088680149056017398834228515625	
9,223,372,036,854,775,808	63	0.000000000000000000108420217248550443400745280086994171142578125	
18,446,744,073,709,551,616	64	0.0000000000000000000542101086242752217003726400434970855712890625	

Powers of 2 (Part 1 of 2)

18,446,744,073,709,551,616	64
36,893,488,147,419,102,232	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
19,342,813,113,934,066,795,298,816	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
324,518,553,658,426,725,783,156,020,576,256	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
5,192,296,658,534,827,628,530,496,329,220,096	112
10,384,593,717,069,655,257,060,992,658,440,192	113
20,769,187,434,139,310,514,121,985,316,890,384	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
332,306,998,946,228,969,225,951,765,070,386,144	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
2,658,455,991,569,831,745,907,614,120,560,689,152	121
5,316,911,983,139,663,491,815,228,241,121,378,304	122
10,633,823,966,279,326,983,230,456,482,242,756,608	123
21,267,647,932,558,653,966,460,312,964,485,513,216	124
42,535,295,965,117,307,932,921,825,928,971,026,432	125
85,070,591,730,234,615,865,843,651,857,942,057,864	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

Powers of 2 (Part 2 of 2)

## Appendix E. 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            Decimal  
000 to FFF            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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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 Fractions

HALFWORD												
BYTE				BYTE								
BITS		0123		4567		0123			4567			
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal Equivalent	
.0	.0000	.00	.0000 0000	.000	.0000 0000 0000	.0000	.0000 0000 0000	.0000	.0000 0000 0000	.0000	.0000 0000 0000	
.1	.0625	.01	.0039 0625	.001	.0002 4414 0625	.0001	.0000 1525 8789 0625	.0000	.0000 1525 8789 0625	.0001	.0000 1525 8789 0625	
.2	.1250	.02	.0078 1250	.002	.0004 8828 1250	.0002	.0000 3051 7578 1250	.0002	.0000 3051 7578 1250	.0002	.0000 3051 7578 1250	
.3	.1875	.03	.0117 1875	.003	.0007 3242 1875	.0003	.0000 4577 6367 1875	.0003	.0000 4577 6367 1875	.0003	.0000 4577 6367 1875	
.4	.2500	.04	.0156 2500	.004	.0009 7656 2500	.0004	.0000 6103 5156 2500	.0004	.0000 6103 5156 2500	.0004	.0000 6103 5156 2500	
.5	.3125	.05	.0195 3125	.005	.0012 2070 3125	.0005	.0000 7629 3945 3125	.0005	.0000 7629 3945 3125	.0005	.0000 7629 3945 3125	
.6	.3750	.06	.0234 3750	.006	.0014 6484 3750	.0006	.0000 9155 2734 3750	.0006	.0000 9155 2734 3750	.0006	.0000 9155 2734 3750	
.7	.4375	.07	.0273 4375	.007	.0017 0898 4375	.0007	.0001 0681 1523 4375	.0007	.0001 0681 1523 4375	.0007	.0001 0681 1523 4375	
.8	.5000	.08	.0312 5000	.008	.0019 5312 5000	.0008	.0001 2207 0312 5000	.0008	.0001 2207 0312 5000	.0008	.0001 2207 0312 5000	
.9	.5625	.09	.0351 5625	.009	.0021 9726 5625	.0009	.0001 3732 9101 5625	.0009	.0001 3732 9101 5625	.0009	.0001 3732 9101 5625	
.A	.6250	.0A	.0390 6250	.00A	.0024 4140 6250	.000A	.0001 5258 7890 6250	.000A	.0001 5258 7890 6250	.000A	.0001 5258 7890 6250	
.B	.6875	.0B	.0429 6875	.00B	.0026 8554 6875	.000B	.0001 6784 6679 6875	.000B	.0001 6784 6679 6875	.000B	.0001 6784 6679 6875	
.C	.7500	.0C	.0468 7500	.00C	.0029 2968 7500	.000C	.0001 8310 5468 7500	.000C	.0001 8310 5468 7500	.000C	.0001 8310 5468 7500	
.D	.8125	.0D	.0507 8125	.00D	.0031 7382 8125	.000D	.0001 9836 4257 8125	.000D	.0001 9836 4257 8125	.000D	.0001 9836 4257 8125	
.E	.8750	.0E	.0546 8750	.00E	.0034 1796 8750	.000E	.0002 1362 3046 8750	.000E	.0002 1362 3046 8750	.000E	.0002 1362 3046 8750	
.F	.9375	.0F	.0585 9375	.00F	.0036 6210 9375	.000F	.0002 2888 1835 9375	.000F	.0002 2888 1835 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 6875  
 Find .00C in position 3 .0029 2968 7500  
 .ABC Hex is equal to .6708 9843 7500

To convert fractions beyond the capacity of table, use techniques below:

### HEXADECIMAL FRACTION TO DECIMAL

Convert the hexadecimal fraction to its decimal equivalent using the same technique as for integer numbers. Divide the results by  $16^n$  (n is the number of fraction positions).  
 Example:  $.8A7_{16} = .540771_{10}$

$$\begin{array}{r} 8A7_{16} = 2215_{10} \\ 16^3 = 4096 \quad \quad \quad \frac{.540771}{4096 \overline{)2215.000000}} \end{array}$$

### TO CONVERT .13 DECIMAL TO HEXADECIMAL

1. Find .1250 next lowest to .1300  
 subtract  $-.1250$  = .2 Hex
2. Find .0039 0625 next lowest to .0050 0000  
 subtract  $-.0039 0625$  = .01
3. Find .0009 7656 2500  
 subtract  $-.0010 9375 0000$  = .004  
 subtract  $-.0009 7656 2500$
4. Find .0001 0681 1523 4375  
 subtract  $-.0001 1718 7500 0000$  = .0007  
 subtract  $-.0001 0681 1523 4375$
5. .13 Decimal is approximately equal to  $.2147_{16}$  Hex

### DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.  
 Example:  $.5408_{10} = .8A7_{16}$

$$\begin{array}{r} .5408 \\ \times 16 \\ \hline 8 \leftarrow 8.6528 \\ \times 16 \\ \hline A \leftarrow 10.4448 \\ \times 16 \\ \hline 7 \leftarrow 7.1168 \end{array}$$

## 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

# Index

- A**
- access (*see also* reference)
    - exception 6-13
    - priority of 6-17
    - key 3-6
  - access-control bits 3-4
  - ADD (A, AR) binary instructions 7-7
  - ADD DECIMAL (AP) instruction 8-3
    - example A-26
  - ADD HALFWORD (AH) instruction 7-7
    - example A-6
  - ADD LOGICAL (AL, ALR) instructions 7-7
  - ADD NORMALIZED (AD, ADR, AE, AER, AXR) instructions 9-6
    - example A-30
  - ADD UNNORMALIZED (AU, AUR, AW, AWR) instructions 9-7
    - example A-31
  - address
    - arithmetic 7-3
    - base 5-4
    - failing-storage (*see* failing-storage address)
    - format 3-1
    - generation 5-4
    - invalid 6-9
    - I/O (channel/device) 12-7, 12-14
      - assigned storage location for 3-9
    - numbering 3-2
    - page 3-3
    - virtual 3-1
    - wraparound 3-2
  - addressable (page) state 3-4
  - address-compare controls 13-1
  - addressing, one-level 3-2
  - addressing exception 6-9
    - as an access exception 6-13
    - relation to storage size 3-3
  - AFCC (available-frame-capacity count) 3-5
  - alert
    - class of machine-check conditions 11-4
    - error (in limited channel logout) 12-64
  - allowed interruptions 6-4
  - alter-and-display controls 13-2
  - alteration
    - general-register (PER event) 4-12
    - storage (PER event) 4-12
  - AND (N, NC, NI, NR) instructions 7-7
    - examples A-6
  - architectural mode, indication of 13-3
  - arithmetic
    - binary 7-3
    - decimal (*see* decimal instructions)
  - arithmetic (*continued*)
    - floating-point (*see* floating-point instructions)
    - logical (*see* unsigned binary arithmetic)
  - assembler language A-5
    - instruction formats in (*see* individual instruction descriptions)
  - assigned storage locations 3-8
  - attachment of I/O devices 12-50
  - attention 12-50
  - auxiliary storage 3-1, 3-3
  - available (I/O state) 12-10
  - available-frame-capacity count (AFCC) 3-5
- B**
- B field of instruction 5-1
  - base address 5-4
    - register 2-3
  - basic control (*see* BC mode)
  - BC (basic-control) mode 4-3
    - PSW format in 4-6
  - BC to EC mode conversion 10-11
  - binary (*see also* fixed point)
    - arithmetic 7-3
    - negative zero 7-2
    - number representation, examples of A-2
    - one's complement notation 7-2
    - overflow 7-3
    - sign bit 7-2
    - two's complement notation 7-2
      - example A-2
  - binary-to-decimal conversion 7-16
  - block of I/O data 12-30
    - incorrect length for 12-55
    - self-describing 12-35
  - block-multiplexer channel 12-4
  - block-multiplexing control 12-4
    - bit in CR 0 4-8
    - effect on CLEAR I/O instruction of 12-16
    - effect on START I/O FAST RELEASE instruction of 12-23
  - borrow, binary 7-35
  - boundary alignment 3-2
  - BRANCH AND LINK (BAL, BALR) instructions 7-8
    - example A-7
  - BRANCH ON CONDITION (BC, BCR) instructions 7-9
    - example A-7
  - BRANCH ON COUNT (BCT, BCTR) instructions 7-10
    - example A-8
  - BRANCH ON INDEX HIGH (BXH) instruction 7-10
    - example A-8
  - BRANCH ON INDEX LOW OR EQUAL (BXLE) instruction 7-10

- branching 5-4
- burst mode 12-3
- bus-out check 12-41
- busy
  - as I/O unit-status indication 12-51
  - in I/O operations 12-6
- byte 3-1
- byte-multiplex mode 12-3
- byte-multiplexer channel 12-4

## C

- CAI (channel-available interruption) 12-48
- capacity counts 3-5
- carry 7-2
- CAW (channel-address word) 12-30
- CBC (checking-block code) 11-1
  - invalid
    - in page descriptions 11-3
    - in registers 11-3
    - in storage 11-2
- CC (chain-command) flag 12-31
- CCW (channel-command word) 12-31
  - address in CAW 12-31
  - address in CSW 12-50, 12-58
  - assigned storage location (during IPL) 3-10
  - command code 12-32
  - prefetching 12-33
- CD (chain-data) flag 12-31
- central processing unit (*see* CPU)
- chain-command (CC) flag 12-31
- chain-data (CD) flag 12-31
- chaining 12-33
- chaining check 12-57
- change bit 3-4
- change recording 3-7
- channel 12-2
  - address 12-7
  - block-multiplexer 12-4
  - byte-multiplexer 12-4
  - commands 12-37
  - control check 12-56
  - data check 12-56
  - end 12-53
  - equipment error 12-14
  - identification (ID) 12-26
    - assigned storage location for 3-9
    - in I/O communications area 12-63
  - logout
    - limited 3-9, 12-63
  - masks
    - in BC-mode PSW 4-6
    - in control register 4-8
  - model field 12-26
  - programming error 12-14

- channel (*continued*)
  - selector 12-4
  - serialization 5-10
  - status 12-55
  - time-out 12-4
  - type field 12-26
- channel-address word (CAW) 12-30
- channel-available interruption (CAI) 12-48
- channel-command word (*see* CCW)
- channel-status word (CSW) 12-49
- channel-to-channel adapter 12-2
- characteristic (of floating-point number 9-1
- check 11-1
  - code (*see* CBC)
- check bits 11-1
- check control 13-2
- check-stop indicator 13-2
- check-stop state 4-2
  - effect of check control on 12-3
  - entering of 11-3
- class number, monitor 3-9, 4-9
- CLEAR I/O (CLRIO) instruction 12-16
- clear key
  - load 13-3
  - system-reset 13-5
- CLEAR PAGE (CLRP) instruction 10-3
- clear reset 4-23
- clearing
  - storage
    - by CLEAR PAGE instruction 10-3
    - by clear reset 4-23
- clock, time-of-day (TOD) 4-15
- clock comparator 4-18
  - external interruption 4-18, 6-19
  - mask 4-8
  - save area 3-9
  - validity bit 11-9
- clock unit 4-17
- CLRIO function 12-16
- code
  - check 11-1
  - command 12-32
  - condition (*see* condition code)
  - instruction-length (*see* instruction-length code)
  - monitor 3-9, 4-9
  - operation 5-1
  - PER 3-9
- codes, decimal digit and sign 8-2
- command
  - chaining 12-36
  - code 12-32
  - reject 12-43
    - sense data 12-41
  - command retry 12-42
  - commands 12-37

communications area, I/O 12-63

COMPARE (C, CR) binary instructions 7-11

COMPARE (CD, CDR, CE, CER) floating-point instructions 9-8  
 example A-31

COMPARE AND SWAP (CS) instruction 7-11  
 examples A-32

COMPARE DECIMAL (CP) instruction 8-4  
 example A-26

COMPARE DOUBLE AND SWAP (CDS) instruction 7-11

COMPARE HALFWORD (CH) instruction 7-13  
 example A-9

COMPARE LOGICAL (CL, CLC, CLI, CLR) instructions 7-13  
 examples A-9

COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) instruction 7-13  
 example A-10

COMPARE LOGICAL LONG (CLCL) instruction 7-14  
 example A-11

comparison  
 decimal 8-4  
 floating-point 9-8  
 logical 7-3  
 signed-binary 7-3

compatibility 1-2  
 I/O operation 12-7

completion (instruction ending) 6-7

conceptual sequence 5-5

conclusion (I/O-operation) 12-43

concurrent program-interruption conditions 4-13

condition code  
 deferred 12-19  
 for SIOF function 12-25  
 for I/O operations 12-12  
 in PSW 4-5, 4-6  
 inspection 5-4  
 summary C-1  
 validity bit 11-8

conditions  
 interruption 6-1  
 I/O 12-47  
 program 6-9

CONNECT PAGE (CTP) instruction 10-3

connected (page) state 3-4

connective (*see* logical connective)

consistency, storage operand 5-8

console device 13-1

control 4-1  
 check 13-2  
 command 12-40  
 instructions 10-1  
 summary 10-2  
 interval-timer 13-3  
 page and page-frame 3-5

control (*continued*)  
 rate 13-4  
 register 2-3, 4-7  
 save area for 3-9  
 validity bit 11-9  
 storage size 13-4  
 TOD-clock 13-5  
 unit 12-2  
 sharing of 12-5

controls  
 address 13-1  
 alter-and-display 13-2  
 IML 13-2  
 load-unit-address 13-3  
 power 13-4

control-unit end 12-51

conversion  
 BC-to-EC-mode 10-11  
 binary-to-decimal 7-16  
 decimal-to-binary 7-15  
 fixed- to floating-point, example of A-5

CONVERT TO BINARY (CVB) instruction 7-15  
 example A-12

CONVERT TO DECIMAL (CVD) instruction 7-16  
 example A-12

count field  
 in CCW 12-31  
 in CSW 12-50, 12-59

counter-updating example A-33

counting 7-10

CPU (central processing unit) 2-1  
 identification (ID) 10-13  
 model number 10-13  
 registers 2-3  
 retry 11-1  
 serialization 5-9  
 state 4-1  
 version code 10-13

CPU timer 4-18  
 external interruption 4-18, 6-19  
 mask 4-8  
 save area for 3-9  
 updating of 4-3  
 validity bit 11-9

CR (*see* control register)

CSW (channel-status word) 12-49

current PSW 4-4

**D**

D field of instruction 5-1

damage  
 external 11-7  
 mask bit for 4-8, 11-10

damage (*continued*)  
   instruction-processing 11-7  
   system 11-7  
 data  
   chaining 12-35  
   check 12-41  
   exception 6-9  
   format for  
     decimal instructions 8-1  
     floating-point instructions 9-1  
     general instructions 7-1  
   prefetching 12-33  
 decimal  
   comparison 8-4  
   data format 8-1  
   digit codes 8-2  
   instructions 8-1  
     examples for A-25  
     summary 8-3  
   number representation 8-2  
     examples A-3  
   operand overlap 8-2  
   overflow  
     exception 6-10  
     mask 4-5, 4-6  
   rounding 8-8  
   sign codes 8-2  
 decimal-divide exception 6-10  
 decimal-to-binary conversion 7-15  
 DECONFIGURE PAGE (DEP) instruction 10-4  
 deferred condition code 12-12  
   for SIOF function 12-25  
   in CSW 12-50  
 degradation  
   bit in machine-check interruption code 11-7  
   mask bit for 4-8, 11-10  
 delayed bit 11-8  
 destructive overlap 7-23  
 detect field (in limited channel logout) 12-63  
 device  
   address 7-12  
   console 13-1  
   end 12-53  
   error 12-14  
   I/O 12-2  
     not-ready state 12-10  
     status 12-40  
 DIAGNOSE instruction 10-4  
 digit codes, decimal 8-2  
 digit selector 8-5  
 direct-access storage 3-3  
 disabling  
   for interruptions 6-4  
   the interval timer 4-20  
 disallowed interruption 6-4  
 DISCONNECT PAGE (DCTP) instruction 10-5  
 disconnected (page) state 3-4  
 displacement 5-4  
 displav 13-2  
 DIVIDE (D, DR) binary instructions 7-16  
   example A-13  
 DIVIDE (DD, DDR, DE, DER) floating-point instructions  
 DIVIDE DECIMAL (DP) instruction 8-4  
   example A-26  
 doubleword 3-2  
 dump, standalone 13-3  
  
**E**  
 early exception recognition 6-6  
 EC (extended-control) mode 4-3  
   control bit in PSW 4-5, 4-6  
   PSW format in 4-4  
 ECC (error checking and correction) 11-1  
 ECPS:VSE mode (extended control program support:  
   virtual storage extended mode) 1-1  
   selection of 13-2  
 EDIT (ED) instruction 8-5  
   example A-27  
 EDIT AND MARK (EDMK) instruction 8-8  
   example A-28  
 EDIT functions, summary 8-7  
 EFCC (existing-frame-capacity count) 3-5  
 emulation instruction, PER event for 4-12  
 enabling (for interruptions) 6-4  
 equipment check 12-41  
 error  
   alert (in limited channel logout) 12-64  
   channel-equipment 12-14  
   channel-programming 12-14  
   checking and correction (ECC) 11-1  
   device 12-14  
   state of time-of-day clock 4-16  
   storage 11-8  
   storage-key 11-8  
 event  
   monitor 4-8  
   PER 4-9  
 exception  
   addressing 6-9  
   data 6-9  
   decimal-divide 6-10  
   decimal-overflow 6-10  
   early recognition of 6-6  
   execute 6-10  
   exponent-overflow 6-10  
   exponent-underflow 6-10  
   fixed-point-divide 6-10  
   fixed-point-overflow 6-11  
   floating-point-divide 6-11  
   late recognition of 6-6

- exception (*continued*)
  - operation 6-11
  - page-access 6-11
  - page-state 6-12
  - page-transition 6-12
  - privileged-operation 6-12
  - protection 6-12
  - significance 6-13
  - special-operation 6-13
  - specification 6-13
- exceptions 6-8
  - access 6-13
  - associated with PSW 6-6
- EXCLUSIVE OR (X, XC, XI, XR) instructions 7-17
  - examples A-13
- EXECUTE (EX) instruction 7-17
  - example A-14
  - PER event for target instruction 4-12
- execute exception 6-10
- existent condition 11-4
- existing-frame-capacity count (EFCC) 3-5
- exponent 9-1
  - (*see also* floating-point)
  - overflow 9-6, 9-9, 9-12, 9-13
    - exception 6-10
  - underflow 9-6, 9-9, 9-10, 9-13
    - exception 6-10
    - mask bit for 4-5, 4-6
- extended
  - control (*see* EC mode)
  - control program support (*see* ECPS:VSE mode)
  - floating-point number 9-1
- external
  - damage 11-7
    - mask bit for 4-8, 11-10
  - interruption 6-19
    - clock-comparator 4-18, 6-19
    - CPU-timer 4-18, 6-19
    - external-signal 6-20
    - interrupt-key 6-20
    - interval-timer 4-19, 6-20
    - priority 6-19
    - submask bits 6-19
  - mask 4-5, 4-6
  - signal 6-20
    - mask bit for 4-8
- externally initiated functions 4-20

**F**

- failing-storage address 3-9, 11-9
  - validity bit 11-8
- fetch protection 3-6
- fetch reference to storage 5-7
- fetching, instruction 5-6
- fetch-protection bit 3-4

- FFCC (free-frame-capacity count) 3-6
- field separator 8-5
- fill character 8-5
- fixed- to floating-point conversion, example of A-5
- fixed-point (*see also* binary)
  - divide exception 6-10
  - overflow exception 6-11
  - overflow mask 4-5, 4-6
- flags (in CCW) 12-31
- floating-point (*see also* exponent)
  - characteristic 9-1
  - comparison 9-8
  - data format 9-1
  - divide exception 6-11
  - fraction 9-1
  - instructions 9-1
    - examples for A-30
    - summary 9-5
  - number representation 9-3
    - examples A-4
  - range 9-3
  - register 2-2
    - save area for 3-9
    - validity bit 11-9
  - rounding 9-12
  - shifting (*see* normalization)
  - sign bit 9-3
- format
  - address 3-1
  - data
    - floating-point 9-1
    - general-instruction 7-1
  - information 3-1
  - instruction 5-2
- format error, PSW 6-6
- fraction 9-1
- frame, control 3-5
- frame index 3-5
- free-frame-capacity count (FFCC) 3-6
- fullword 3-1

**G**

- general instructions 7-1
  - data formats of 7-1
  - examples for A-6
- general register 2-2
  - alteration (PER event) 4-12
    - masks for 4-8
  - save area for 3-9
  - validity bit 11-9
- guard digit 9-2

**H**

- halfword 3-1
- HALT DEVICE (HDV) instruction 12-18

HALT I/O (HIO) instruction 12-21  
HALVE (HDR, HER) instructions 9-9  
hexadecimal (hex) representation 5-3  
high-order bits 3-1

## I

I field of instruction 5-1  
IC (instruction counter) (*see* instruction address)  
ID (*see* channel identification, CPU identification)  
ILC (*see* instruction-length code)  
IML (initial microprogram loading) 13-2  
    controls 13-2  
immediate I/O operation 12-44  
immediate operand 5-3  
incorrect length (of I/O block) 12-55  
index 5-4  
index handling, instructions for 7-10  
index register 2-2  
indicator  
    check-stop 13-2  
    load 13-3  
    manual 13-3  
    mode 13-3  
    save 13-4  
    system 13-5  
    test 13-5  
    wait 13-5  
information format 3-1  
initial  
    microprogram loading (IML) 13-2  
    program loading (IPL) 4-24  
        assigned storage locations for 3-9  
    program reset 4-23  
input/output (*see* I/O)  
INSERT CHARACTER (IC) instruction 7-18  
INSERT CHARACTERS UNDER MASK (ICM)  
    instruction 7-18  
    examples A-15  
INSERT PAGE BITS (IPB) instruction 10-5  
INSERT PSW KEY (IPK) instruction 10-5  
INSERT STORAGE KEY (ISK) instruction 10-6  
instruction  
    address (in PSW) 4-5, 4-7  
        validity bit for 11-8  
    classes 2-2  
    execution sequence 5-4  
    fetching 5-6  
    fetching (PER event) 4-12  
        event mask 4-8  
    format 5-2  
    interruptible 6-7  
    length code (ILC) 6-4  
        assigned storage locations for 3-9  
        in BC-mode PSW 4-6  
    lists B-1

instruction (*continued*)  
    modification by EXECUTE instruction 7-17  
    prefetching 5-6  
    processing damage 11-7  
    step  
        effect on CPU state of 4-2  
        rate-control setting 13-4  
    use examples A-5  
instruction counter (*see* instruction address)  
instructions  
    control 10-1  
    decimal 8-1  
        examples A-25  
    floating-point 9-1  
        examples A-30  
    general 7-1  
        examples A-6  
    I/O 12-15  
    privileged 4-2  
    storage-control 3-6  
integer  
    binary 7-1  
        examples A-2  
    decimal 8-2  
integral boundary 3-2  
interface-control check 12-57  
intermittent failure 11-3  
internal storage 2-2  
interrupt key 13-2  
    external interruption 6-20  
    mask 4-8  
interruptible instructions 6-7  
    COMPARE LOGICAL LONG 7-14  
    MOVE LONG 7-23  
    stopping of 4-3  
interruption 6-1  
    action  
        I/O 12-49  
        machine-check 11-4  
    code 6-2  
        assigned storage locations for 3-9  
        in BC-mode PSW 4-6  
        machine-check 11-6  
    conditions 6-1  
        I/O 12-47  
    external 6-19  
    input/output 6-20  
    machine-check 6-8  
        code 11-6  
    mask bits 6-4  
    pending 6-4  
        I/O state for 12-8  
        machine-check 11-5

- interruption (*continued*)
  - priority 6-21
  - program 6-8
  - restart 6-21
  - supervisor-call 6-8
- interruptions, string of 6-22
- interval timer 4-19
  - control 13-3
  - damage 11-7
  - disabling of 4-20
  - external interruption 4-19, 6-20
  - location 80 for 4-19
  - mask 4-8
  - update reference 5-9
  - updating of 4-3
- intervention required 12-41
- invalid
  - address 6-9
  - CBC 11-1
    - in page descriptions 11-3
    - in registers 11-3
    - in storage 11-2
  - channel programs 12-55
  - operation code 6-11
- inverse move 7-22
- I/O
  - devices and control units 2-4
  - operations 2-3
- I/O (input/output) 12-1
  - address 12-7, 12-14
    - assigned storage location for 3-9
  - commands 12-37
  - communications area (IOCA) 12-63
  - data block 12-30
  - device 12-2
    - address 12-7
    - attachment 12-50
    - attachment, end 12-53
    - attachment, error 12-14
    - attachment, not-ready state 12-10
    - attachment, status 12-40
  - instructions 12-15
  - interface position 12-48
  - interruption 6-20
    - action 12-49
    - conditions 12-47
    - priority 12-48
  - mask in PSW 4-5, 4-6
  - operations
    - compatibility of 7-12
    - conclusion of 12-43
    - storage-area designation for 12-32
  - selective reset 12-18
  - state 12-8
  - status 12-50, 12-55

- I/O (input/output) (*continued*)
  - system reset 12-18
    - as part of program reset 4-21
  - system state 12-8
- IOCA (I/O communications area) 12-63
- IPL (initial program loading) 4-24
  - assigned storage locations for 3-9

## K

- key
  - access 3-6
  - interrupt 13-2
  - load-clear 13-3
  - load-normal 13-3
  - machine-save 13-3
  - restart 13-4
  - start 13-4
  - stop 13-4
  - storage (*see also* page description)
  - storage 3-3
  - system-reset-clear 13-5
  - system-reset-normal 13-5
- key-controlled protection 3-6

## L

- L field of instruction 5-1
- late exception recognition 6-6
- left-to-right addressing 3-2
- length
  - instruction 5-2
  - I/O-block (*see also* count, I/O)
  - I/O-block 12-55
  - operand 5-1
- limited channel logout 3-9, 12-63
- link information (for BRANCH AND LINK instruction) 7-8
- linkage, subroutine 5-4
- load
  - clear key 13-3
  - indicator 13-3
  - normal key 13-3
  - state 4-2, 4-24
    - assigned storage in 3-9
    - unit-address controls 13-3
- LOAD (L, LR) binary instructions 7-19
  - example A-16
- LOAD (LD, LDR, LE, LER) floating-point instructions 9-10
- LOAD ADDRESS (LA) instruction 7-19
  - examples A-16
- LOAD AND TEST (LTDR, LTER) floating-point instructions 9-10
- LOAD AND TEST (LTR) binary instruction 7-19

LOAD COMPLEMENT (LCDR, LCER) floating-point instructions 9-11

LOAD COMPLEMENT (LCR) binary instruction 7-20

LOAD CONTROL (LCTL) instruction 10-6

LOAD FRAME INDEX (LFI) instruction 10-6

LOAD HALFWORD (LH) instruction 7-20  
examples A-16

LOAD MULTIPLE (LM) instruction 7-20

LOAD NEGATIVE (LNDR, LNER) floating-point instructions 9-11

LOAD NEGATIVE (LNR) binary instruction 7-21

LOAD POSITIVE (LPDR, LPER) floating-point instructions 9-11

LOAD POSITIVE (LPR) binary instruction 7-21

LOAD PSW (LPSW) instruction 10-7

LOAD ROUNDED (LRDR, LRER) instructions 9-12

loading  
initial microprogram 13-2  
initial program 4-24

location not provided 6-9

location 80 (for interval timer) 4-19

logical  
arithmetic (*see* unsigned binary arithmetic)  
comparison 7-3  
connective  
AND 7-8  
EXCLUSIVE OR 7-17  
OR 7-27  
data 7-1

logout  
limited channel 3-9, 12-63  
pending 12-49

long  
floating-point number 9-1  
I/O block 12-55

lookaside for storage keys 11-3

loop control 5-5

low-order bits 3-1

## M

machine malfunction 11-1  
effect on manual operation of 13-1

machine save 4-25  
key 13-3

machine status  
retrieval 10-8  
saving 4-25

machine-check  
detection 11-1  
interruption 6-8  
action 11-4  
code (MCIC) 11-6  
mask in PSW 4-5, 4-6  
submask bits 11-9

main storage (*see also* storage)

main storage 2-1, 3-1

MAKE ADDRESSABLE (MAD) instruction 10-7

MAKE UNADDRESSABLE (MUN) instruction 10-8

malfunction  
machine 11-1  
effect on manual operation of 13-1

manual indicator 13-3

mask  
interruption 6-4  
clock-comparator 4-8  
CPU-timer 4-8  
decimal-overflow 4-5, 4-6  
degradation-report 4-8, 11-10  
exponent-underflow 4-5, 4-6  
external 4-5, 4-6  
external-damage-report 4-8, 11-10  
external-signal 4-8  
fixed-point-overflow 4-5, 4-6  
general-register-alteration-event 4-8  
instruction-fetching-event 4-8  
interrupt-key 4-8  
interval-timer 4-8  
I/O 4-5, 4-6  
machine-check 4-5, 4-6  
PER 4-5, 4-9  
program 4-5, 4-6  
recovery-report 4-8, 11-10  
significance 4-5, 4-6  
storage-alteration-event 4-8  
successful-branching-event 4-8  
warning 4-8, 11-10

mask field  
for BRANCH ON CONDITION instruction 7-9  
for COMPARE LOGICAL CHARACTERS UNDER MASK instruction 7-14  
for INSERT CHARACTERS UNDER MASK instruction 7-18  
for STORE CHARACTERS UNDER MASK instruction 7-32

masks  
channel  
in BC-mode PSW 4-6  
in control register 4-8  
machine-check 11-9  
monitor 4-8  
PER general-register 4-8, 4-9  
PER-event 4-9

maximum negative number 7-2

MCIC (machine-check-interruption code) 11-6

message character 8-5

microprogram, initial loading of 13-2

mode  
 BC 4-3  
 burst 12-3  
 byte-multiplex 12-3  
 EC 4-3  
   control bit in PSW 4-5, 4-6  
 ECPS:VSE 1-1  
 indicator 13-3  
 System/370 1-1

model  
 channel 12-26  
 CPU 10-13

modifier bits, in CCW command code 12-32

monitor  
 class number 3-9, 4-9  
 code 3-9, 4-9  
 event 4-8  
   as a program interruption 6-11  
 masks 4-8

MONITOR CALL (MC) instruction 7-21

monitoring 4-8

MOVE (MVC, MVI) instructions 7-22  
 examples A-14, A-17

MOVE INVERSE (MVCIN) instruction 7-22

MOVE LONG (MVCL) instruction 7-23  
 example A-18

MOVE NUMERICS (MVN) instruction 7-25  
 example A-18

MOVE WITH OFFSET (MVO) instruction 7-25  
 example A-19

MOVE ZONES (MVZ) instruction 7-26  
 example A-19

MULTIPLY (M, MR) binary instructions 7-26  
 examples A-20

MULTIPLY (MD, MDR, ME, MER, MXD, MXDR, MXR)  
 floating-point instructions 9-12

MULTIPLY DECIMAL (MP) instruction 8-8  
 example A-28

MULTIPLY HALFWORD (MH) instruction 7-27  
 example A-20

multiprocessing, considerations for 7-4, 8-2, A-32

multiprogramming examples A-32

## N

near-valid CBC 11-1

negative zero  
 binary 7-2  
 decimal, example of A-4

new PSW 4-4  
 assigned storage locations for 3-8

nonshared subchannel 12-4

nontransparent nullification 6-16

no-operation (I/O control command) 12-40

normal key  
 load 13-3  
 system-reset 13-5

normalization 9-3

not-available state (I/O system) 12-8

not-operational state  
 of I/O system 12-8  
 of time-of-day clock 4-16

not-ready state (I/O device) 12-10

not-set state (time-of-day clock) 4-16

nullification (instruction ending) 6-7  
 exception to rule for 6-16  
 nontransparent 6-16

number, CPU model 10-13

number representation  
 binary, examples of A-2  
 decimal 8-2  
   examples of A-3  
 floating-point 9-3  
   examples A-4

numbering  
 addresses (byte locations) 3-2  
 bits 3-1

numeric bits 8-1

## O

old PSW, assigned storage locations for 3-8

one's complement binary notation 7-2  
 used for SUBTRACT LOGICAL instruction 7-35

one-level addressing 3-2

op code (operation code) 5-1

operand 5-1  
 address generation 5-4  
 immediate 5-3  
 length 5-1  
 overlap 7-2  
   decimal 8-2  
 register 5-3  
 storage 5-3

operating state 4-2  
 assigned storage in 3-8

operation  
 code (op code) 5-1  
   invalid 6-11  
   exception 6-11  
   unit of 6-7

operational (I/O state) 12-8

operator facilities 2-4, 13-1

OR (O, OC, OI, OR) instructions 7-27  
 example of problem with OR immediate A-32  
 examples A-21

organization, system 2-1

overflow  
 binary 7-3  
 decimal 6-10

overflow (*continued*)

exponent 6-10  
fixed-point 6-11

overlap

destructive 7-23  
operand 7-2  
decimal 8-2

overrun 12-41

**P**

PACK (PACK) instruction 7-28

example A-21

packed decimal number 8-2

examples A-3

padding byte

for COMPARE LOGICAL LONG instruction 7-14  
for MOVE LONG instruction 7-23

page 3-3

address 3-3

bits 3-4

control 3-5

description 3-3

sequence of references for 5-6  
validation of 11-3

frame 3-3

state 3-4

page-access exception 6-11

as an access exception 6-13

page-capacity count (PCC) 3-5

page-state exception 6-12

page-transition exception 6-12

page-0

retrieval 10-8

saving 4-25

parity bit 11-1

pattern 8-5

PCC (page-capacity count) 3-5

PCI (program-controlled interruption) 12-36

channel status 12-55

flag 12-31

pending interruption 6-4

PER (program-event recording) 4-9

address 3-9

wraparound 4-11

code 3-9

ending address 4-8, 4-10

event 4-9

concurrent interruption conditions 4-13

for emulation instruction 4-12

masks 4-9

priority 4-11

program interruption 6-12

storage-area designation 4-12

successful branching 4-12

PER (program-event recording) (*continued*)

general-register masks 4-8, 4-9

mask 4-9

in PSW 4-5

starting address 4-8, 4-10

point of interruption 6-7

for machine check 11-5

postnormalization 9-3

power controls 13-4

power-on reset 4-23

precision 9-1

preferred sign codes 8-2

prefetching

for I/O 12-33

instruction 5-6

prenormalization 9-3

pretest for accessibility 6-16

priority

access exceptions 6-17

interruption 6-21

external 6-19

I/O 12-48

program 6-17

PER event 4-11

privileged instructions 4-2

for control 10-1

for I/O 12-15

privileged-operation exception 6-12

problem state 4-2

bit in PSW 4-5, 4-6

program

check 12-55

event (*see* PER event)

exceptions 6-8

initial loading of 4-24

interruption 6-8

priority 6-17

mask 4-5, 4-6

validity bit for 11-8

reset 4-21

status word (PSW) 4-4

program event recording (PER) 4-9

program-controlled interruption (PCI) 12-36

channel status 12-55

flag 12-31

protection

check 12-56

exception 6-12

as an access exception 6-13

storage 3-6

PSW (program-status word) 4-4

assigned storage location (during IPL) 3-9

BC-mode 4-6

EC-mode 4-4

exceptions associated with 6-6

PSW (program-status word) (*continued*)

- format error 6-6
- validity bits 11-8
- PSW key 3-6
  - in PSW 4-5, 4-6

**R**

- R field of instruction 5-1
- range, floating-point 9-3
- rate control 13-4
- read command 12-39
- read-backward command 12-39
- real storage 3-3
- recovery
  - condition 11-4
  - mask bit for 4-8, 11-10
  - system 11-7
- redundancy 11-1
- reference
  - sequence
    - for instructions 5-5
    - for operands 5-7
    - for page description 5-6
    - for storage 5-5
  - single-access 5-8
- reference bit 3-4
- reference recording 3-7
- register
  - base-address 2-2
  - index 2-2
  - operand 5-3
  - save areas 3-9, 11-9
  - validation 11-3
  - validity bits 11-9
- remote operating stations 13-1
- report masks 4-8, 11-10
- repressible condition 11-4
- repressible-damage condition 11-4
- RESET REFERENCE BIT (RRB) instruction 10-8
- resets 4-20
  - effect on CPU state of 4-3
  - I/O 12-18
- resolution, time-of-day clock 4-16
- restart
  - effect on CPU state of 4-2
  - interruption 6-21
  - key 13-4
- retrieval
  - of machine status 10-8
  - of page 0 10-8
- RETRIEVE STATUS AND PAGE (RSP) instruction 10-8
- retry, CPU 11-1

- rounding
  - decimal 8-8
  - factor 8-9
  - floating-point 9-12
- RR instruction format 5-2
- RS instruction format 5-2
- running
  - of CPU (*see* wait state)
  - of time-of-day clock 4-16
- RX instruction format 5-2

**S**

- S instruction format 5-2
- save
  - areas for registers 3-9, 11-9
  - indicator 13-4
  - machine (status and page 0) 4-25
- selective (I/O) reset 12-18
- selector channel 12-4
- self-describing (I/O) block 12-35
- sense
  - command 12-40
  - data 12-40
- sequence
  - code (in limited channel logout) 12-64
  - conceptual 5-5
  - instruction-execution 5-4
  - of storage references 5-5
- serialization
  - channel 5-10
  - CPU 5-9
- SET CLOCK (SCK) instruction 10-9
- SET CLOCK COMPARATOR (SCKC) instruction 10-9
- SET CPU TIMER (SPT) instruction 10-10
- SET PAGE BITS (SPB) instruction 10-10
- SET PROGRAM MASK (SPM) instruction 7-28
- SET PSW KEY FROM ADDRESS (SPKA) 10-10
- set state (time-of-day clock) 4-16
- SET STORAGE KEY (SSK) instruction 10-11
- SET SYSTEM MASK (SSM) instruction 10-11
- shared control unit and subchannel 12-5
- SHIFT AND ROUND DECIMAL (SRP) instruction 8-8
  - examples A-29
- SHIFT LEFT DOUBLE (SLDA) instruction 7-29
  - example A-21
- SHIFT LEFT DOUBLE LOGICAL (SLDL) instruction 7-2
- SHIFT LEFT SINGLE (SLA) instruction 7-30
  - example A-22
- SHIFT LEFT SINGLE LOGICAL (SLL) instruction 7-30
- SHIFT RIGHT DOUBLE (SRDA) instruction 7-30
- SHIFT RIGHT DOUBLE LOGICAL (SRDL) instruction 7-31
- SHIFT RIGHT SINGLE (SRA) instruction 7-31
- SHIFT RIGHT SINGLE LOGICAL (SRL) instruction 7-32
- shifting, floating-point (*see* normalization)

- short
  - floating-point number 9-1
  - I/O block 12-55
- SI instruction format 5-2
- sign bit
  - binary 7-2
  - floating-point 9-3
- sign codes, decimal 8-2
- signal-in lines 6-20
- signed binary
  - arithmetic 7-3
  - comparison 7-3
  - integer 7-1
    - example A-2
- significance
  - exception 6-13
  - loss 9-7
  - mask 4-5, 4-6
  - starter 8-5
- single-access reference 5-8
- SIO function 12-23
- SIOF function 12-23
- size, storage 3-2
- skip flag 12-31
- skipping (during I/O) 12-36
- SLI (suppress-length indication) 12-31
- solid failure 11-3
- source field (in limited channel logout) 12-63
- special-operation exception 6-13
- specification exception 6-13
- SS instruction format 5-2
- SSM (SET SYSTEM MASK) suppression 10-11
  - control 6-13
    - bit in CR 0 4-8
- standalone dump 13-3
- standard epoch 4-16
- start
  - function 4-2
  - key 13-4
- START I/O (SIO) instruction 12-23
- START I/O FAST RELEASE (SIOF) instruction 12-23
- state
  - CPU 4-1
  - page 3-4
  - time-of-day clock 4-16
- states, I/O system 12-8
- status
  - device 12-40
  - in CSW 12-50, 12-60
  - modifier 12-50
  - system 13-5
  - word, program 4-4
- stop
  - function 4-2
  - key 13-4
- stopped state 4-2
- storage 3-1
  - address wraparound 3-2
    - for MOVE INVERSE instruction 7-22
    - for MOVE LONG instruction 7-23
  - addressing 3-2
  - alteration (PER event) 4-12
    - event mask 4-8
  - area designation
    - for I/O operations 12-32
    - for PER events 4-12
  - assigned locations in 3-8
  - auxiliary 3-1, 3-3
  - clearing
    - by CLEAR PAGE instruction 10-3
    - by clear reset 4-23
  - control instructions 3-6
  - control unit (in limited channel logout) 12-63
  - direct-access 3-3
  - error 11-8
  - internal 2-2
  - key 3-3
    - (see also page description)
    - error 11-8
  - location not provided 3-3
  - logical validity bit 11-9
  - main 3-1
  - operand 5-3
    - consistency 5-8
    - fetch reference 5-7
    - store reference 5-7
    - update reference 5-7
  - protection 3-6
  - real 3-3
  - sequence of references to 5-5
  - sharing, examples of A-32
  - size 3-3
  - size control 13-4
  - validation 11-2
    - by CLEAR PAGE instruction 10-3
  - virtual 3-3
  - volatile 3-2
- STORE (ST) binary instruction 7-32
- STORE (STD, STE) floating-point instructions 9-14
- STORE CAPACITY COUNTS (STCAP) instruction 10-11
- STORE CHANNEL ID (STIDC) instruction 12-26
- STORE CHARACTER (STC) instruction 7-32
- STORE CHARACTERS UNDER MASK (STCM)
  - instruction 7-32
  - examples A-22
- STORE CLOCK (STCK) instruction 7-33
- STORE CLOCK COMPARATOR (STCKC)
  - instruction 10-12
- STORE CONTROL (STCTL) instruction 10-12
- STORE CPU ID (STIDP) instruction 10-12
- STORE CPU TIMER (STPT) instruction 10-13

**STORE HALFWORD (STH) instruction** 7-33  
**STORE MULTIPLE (STM) instruction** 7-33  
 example A-23  
 store reference to storage 5-7  
**STORE THEN AND SYSTEM MASK (STNSM)**  
 instruction 10-13  
**STORE THEN OR SYSTEM MASK (STOSM)**  
 instruction 10-14  
 string of interruptions 6-22  
 subchannel 12-4  
 key 3-6  
   in CAW 12-30  
   in CSW 12-49, 12-58  
   nonshared 12-5  
   shared 12-5  
   subchannel 12-5  
 submask bits  
   external-interruption 6-19  
   machine-check 11-9  
 subroutine linkage 5-4  
**SUBTRACT (S, SR) binary instructions** 7-34  
**SUBTRACT DECIMAL (SP) instruction** 8-10  
**SUBTRACT HALFWORD (SH) instruction** 7-34  
**SUBTRACT LOGICAL (SL, SLR) instructions** 7-34  
**SUBTRACT NORMALIZED (SD, SDR, SE, SER, SXR)**  
 instructions 9-14  
**SUBTRACT UNNORMALIZED (SU, SUR, SW, SWR)**  
 instructions 9-14  
 successful branching  
   event mask 4-8  
   PER event 4-12  
**SUPERVISOR CALL (SVC) instruction** 7-35  
 supervisor state 4-2  
 supervisor-call interruption 6-8  
 suppression (instruction ending) 6-7  
 suppress-length-indication (SLI) flag 12-31  
 swapping 7-11  
 system  
   damage 11-7  
   indicator 13-5  
   mask 4-4  
   organization 2-1  
   recovery 11-13  
   reset (*see also* clear reset, program reset)  
     clear key 13-5  
     I/O 12-18  
     normal key 13-5  
   status 13-4, 13-5  
 System/370 mode 1-1  
   selection of 13-2

## T

target of EXECUTE instruction 7-18  
   PER event for 4-12  
 termination (instruction ending) 6-7  
 termination code (in limited channel logout) 12-64  
**TEST AND SET (TS) instruction** 7-35  
**TEST CHANNEL (TCH) instruction** 12-27  
 test indicator 13-5  
**TEST I/O (TIO) instruction** 12-27  
**TEST UNDER MASK (TM) instruction** 7-36  
 example A-23  
**TIC (transfer in channel) command** 12-42  
 time-of-day (TOD) clock 4-15  
   control 13-5  
   resolution 4-16  
   running of 4-16  
   states 4-16  
   updating of 4-3  
   validation 11-3  
 time-out, channel 12-4  
 timer  
   CPU (*see* CPU timer)  
   interval (*see* interval timer)  
 timing facilities 4-15  
 timing-facility-damage bit 11-7  
**TIO function** 12-27  
   performed by CLEAR I/O instruction 12-16  
**TOD clock** (*see* time-of-day clock)  
**TOD-clock control** 13-5  
**transfer-in-channel (TIC) command** 12-42  
**TRANSLATE (TR) instruction** 7-36  
 example A-23  
**TRANSLATE AND TEST (TRT) instruction** 7-37  
 example A-24  
 true zero 9-3  
 two's complement binary notation 7-2  
 example A-2

## U

unit  
   check 12-53  
   exception 12-54  
   of operation 6-7  
   status 12-50  
 unnormalized floating-point number 9-3  
**UNPACK (UNPK) instruction** 7-38  
 example A-25  
 unsigned binary  
   arithmetic 7-3  
   integer 7-1  
   example A-3

update reference to storage 5-7  
usage meter 13-5

## V

valid CBC 11-1  
validation  
  of page descriptions 11-3  
  of registers 11-3  
  of storage 11-2  
    by CLEAR PAGE instruction 10-3  
  of time-of-day clock 11-3  
  usage of 11-3  
validity bits 11-8  
validity flags (in limited channel logout) 12-64  
version code, CPU 10-13  
virtual, address 3-1  
virtual storage 3-3  
volatile storage 3-2

## W

wait indicator 13-5  
wait state 4-2  
  bit in PSW 4-5, 4-6  
warning  
  bit in machine-check interruption code 11-7  
  mask 4-8, 11-10

word 3-1  
  program status 4-4  
working (I/O state) 12-8  
wraparound  
  of PER addresses 4-12  
  of register numbers  
    for LOAD MULTIPLE instruction 7-20  
    for STORE MULTIPLE instruction 7-33  
  of storage addresses 3-2  
    for MOVE INVERSE instruction 7-22  
    for MOVE LONG instruction 7-23  
write command 12-39

## X

X field of instruction 5-1

## Z

zero, true 9-3  
ZERO AND ADD (ZAP) instruction 8-10  
  example A-30  
zero instruction-length code 6-5  
zone bits 8-1  
zoned decimal number 8-1  
  examples A-3

IBM 4300 Processors  
Principles of Operation  
for ECPS:VSE Mode  
Order No. GA22-7070-0

**READER'S  
COMMENT  
FORM**

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

**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.*

Possible topics for comment are:

Clarity    Accuracy    Completeness    Organization    Coding    Retrieval    Legibility

If you wish a reply, give your name and mailing address:

---

---

---

*Note:* Staples can cause problems with automated mail sorting equipment.  
Please use pressure-sensitive or other gummed tape to seal this form.

What is your occupation? \_\_\_\_\_

Number of latest Newsletter associated with this publication: \_\_\_\_\_

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the front cover or title page.)

Reader's Comment Form

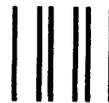
Cut or Fold Along Line

IBM 4300 Processors Principles of Operation for ECPS:VSE Mode (File No. 4300-01) Printed in U.S.A. GA22-7070-0

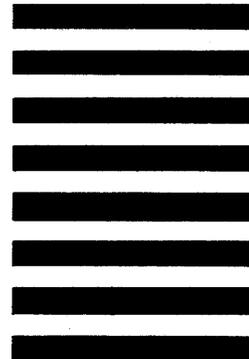
Fold and tape

Please Do Not Staple

Fold and tape



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES



**BUSINESS REPLY MAIL**  
FIRST CLASS    PERMIT NO. 419    POUGHKEEPSIE, NY

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation  
Department B98  
P.O. Box 390  
Poughkeepsie, New York 12602

Fold and tape

Please Do Not Staple

Fold and tape



International Business Machines Corporation  
Data Processing Division  
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation  
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation  
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Cut or Fold Along Line



International Business Machines Corporation  
Data Processing Division  
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation  
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation  
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601