GA22-7000-4
File No. S/370-01

Systems

IBM System/370 Principles of Operation

IBM

Fourth Edition (September 1974)

This major revision obsoletes GA22-7000-3. The revision includes format changes, corrections, and additions.

Significant technical changes are indicated by a vertical line to the left of the change.

Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

Changes are made periodically to the information herein; before using this publication in connection with the operation of the System/370, refer to the latest *IBM System/370 Bibliography,* GC20-0001, and associated technical newsletter, for the editions that are applicable and current.

This manual has been prepared by the IBM System Products Division, Product Publications, Dept. B98, PO Box 390, Poughkeepsie, N.Y., 12602. A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be sent to the above address. Comments and suggestions become the property of IBM.

Preface

This publication provides, for reference purposes, a detailed definition of the machine functions performed by System/370.

The manual describes each function to the level of detail that must be understood in order to prepare an assembly language program that relies on that function. It does not, however, describe the notation and conventions that must be employed in preparing such a program, for which the user must instead refer to the appropriate assembly language manual, such as the *IBM System/360 Operating System Assembly Language*, GC28-6514.

The information in this publication is provided principally for use by assembly language programmers, although anyone concerned with the functional details of System/370 will find it useful.

Note that this manual is written as a reference document and should not be considered to be an introduction or a textbook for System/370. It assumes the user has a basic knowledge of data processing systems and, specifically, the System/370, such as can be derived from the *Introduction to IBM Data Processing Systems*, GC20-1684, and the *IBM System/370 System Summary*, GA22-7001. Persons intending to use the information presented here in preparing computer programs should also become familiar with the publications dealing with the programming language to be used. The language publications available in the System/370 System Library, as well as all publications relating to other aspects of the system, are listed and described in the *IBM System/370 Bibliography*, GC20-0001.

All facilities discussed in this manual are not necessarily available on every model of System/370. 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. Examples of such capabilities are the provisions for the number of channel mask bits in the control register, for the size of the processor address, and for the number of CPUs sharing main storage. 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 System/370 model, use the functional characteristics manual for that model. The availability of features on System/370 models is summarized in the *IBM System/370 System Summary,* GA22-7001.

The information presented in this manual is grouped into 14 chapters and several appendixes:

IBM System/370 highlights some of the major features of System/370--particularly those that constitute advances beyond System/360.

System Organization describes the major groupings within the system--the central processing unit, main storage, and input/output--with some attention paid to the composition and characteristics of those groupings.

Program Execution explains the role of instructions in program execution, looks in detail at instruction formats, and describes briefly the use of the program status word (PSW), of branching, and of interruptions. It also details the aspects of program execution on one CPU as observed by channels or another CPU.

System Control describes in depth the facilities for the switching of system status, for program protection, for special externally initiated operations, and for certain system enhancements. It deals specifically with CPU states, control modes, the PSW, control registers, protection, monitoring, program-event recording, timing facilities, resets, store status, and initial program loading.

Dynamic Address Translation explains the operation of the machine facility which, coupled with special programming support, makes the use of a virtual storage possible in System/370. Dynamic address translation (DAT) eliminates the need to assign a program to a fixed location in real main storage and thus reduces the addressing constraints on system and problem programs.

Interruptions details the System/370 mechanism that permits the CPU to change its state as a result of conditions external to the system, within the system, or within the CPU itself. Six classes of interruptions are identified and described: machine-check interruptions, program interruptions, supervisor-call interruptions, external interruptions, input/output interruptions, and restart interruptions.

Multiprocessing describes the facilities required for the sharing of main storage by multiple CPUs and associated I/O.

System Control Instructions contains detailed descriptions of all of the instructions, except for the I/O instructions, that are available only to the control program.

General Instructions contains detailed descriptions of all of the instructions in the standard instruction set that are available to all programs.

Decimal Instructions describes in detail the instructions provided by the decimal feature.

Floating-Point Instructions contains detailed descriptions of the instructions provided by the floating-point feature and by the extended-precision floating-point feature.

Machine-Check Handling describes the System/370 mechanism for detecting, correcting, and reporting machine malfunctions.

Input/Output Operations explains the programmed control of I/O devices by the channel and by the CPU. It includes detailed descriptions of the I/O instructions, channel command words, and other I/O control formats.

System Console describes the basic manual functions and controls available for operating and controlling the system.

The *Appendixes* include:

- A list of the System/370 features
- A discussion of certain functions that differ from System/360
- Lists of the instructions arranged in several sequences
- Summaries of important formats and of condition-code settings
- Table of the powers of 2
- Tabular information helpful in dealing with hexadecimal numbers
- An EBCDIC chart
- Information about number representation
- Instruction-use examples

Largely because the manual is arranged for reference purposes, certain words and phrases appear, of necessity, earlier in the manual than the principal discussions explaining them. The reader who encounters a problem of this sort should refer to the index, which will indicate the location of the key description.

Contents

IBM System/370 9
General-Purpose Design 9
Compatibility. 10
System Program 10
Availability 11

System Organization 13
Main Storage 13
Information Formats 14
Addressing 14
Information Positioning 15
Central Processing Unit 15
General Registers 16
Floating-Point Registers 16
Control Registers 16
Input and Output 17
Channels 17
Input/Output Interface 17
Input/Output Devices and Control Units 18
System Console 18

Program Execution 19
Instructions 19
Operands 19
Instruction Format 20
Address Generation 21
Program Status Word 22
Instruction Execution 22
Branching. 22
Interruptions. 22
Sequence of Storage References 23
Instruction Fetch 23
DAT Table Fetches 24
Key-in-Storage Accesses 24
Storage-Operand References 25
Storage-Operand Consistency 26
Relation Between Operand Accesses 27
Serialization 28

System Control 29
CPU States 30
Wait and Running States 30
Problem and Supervisor States. 30
Stopped and Operating States 30
Control Modes 31
BC Mode 32
EC Mode 32
Set-System-Mask Suppression 32
Program Status Word 32
Program Status Word Format in BC Mode 33
Program Status Word Format in EC Mode 34
Exceptions Associated with the PSW 35
Control Registers 36
Key in Storage 38
Protection 38
Protection Action 38
Accesses Protected 39
Monitoring 39

Program-Event Recording 39
Control Register Allocation 40
Operation. 40
Storage Area Designation 42
Program Events 42
Indication of Events Concurrently with Other Interruption Conditions 43
Direct Control 46
Time-of-Day Clock 46
Format 46
States 46
Setting and Inspection of Value 47
Clock Comparator 47
CPU Timer 48
Interval Timer 49
Externally Initiated Functions 50
Resets. 50
Store Status 54
Initial Program Loading. 54

Dynamic Address Translation 57
Logical Storage Addressing. 58
Control 58
PSW 58
Control Register 0 58
Control Register 1 59
Translation Tables 59
Segment-Table Entries 59
Page-Table Entries 60
Translation 60
Types of Translation 60
Translation Process 60
Addresses Translated 62
Interlocks Between Logical and Real Storage References . 63
Table Manipulation 64
Translation-Lookaside Buffer 64
States of Translation-Table Entries 65
Use of the Translation-Lookaside Buffer 65
Modification of Translation Tables 66
Reference and Change Recording 67
Address-Translation Exceptions 68
Summary of Dynamic Address Translation Formats . . . 68

Interruptions 69
Interruption Action 70
Source Identification 70
Enabling and Disabling 70
Instruction-Length Code 71
Point of Interruption 74
Instruction Execution 74
Machine-Check Interruption 75
Program Interruption 75
Program Interruption Conditions 75
Recognition of Access Exceptions 80
| Handling of Multiple-Program-Interruption Conditions. . 83
Supervisor-Call Interruption 84
External Interruption 84
| Input/Output Interruption 88

Restart 88
Priority of Interruptions 89
Assigned Main-Storage Locations 90
  Real Main Storage 90
  Absolute Main Storage 91

Multiprocessing 95
Shared Main Storage 95
Prefixing 95
CPU Signaling and Response 97
  Orders 97
  Conditions Determining Response 98
TOD Clock Synchronization 101
CPU Address Identification 101

System-Control Instructions 103
  DIAGNOSE 103
  INSERT PSW KEY 104
  INSERT STORAGE KEY 105
  LOAD CONTROL 105
  LOAD PSW 105
  LOAD REAL ADDRESS 106
  PURGE TLB 107
  READ DIRECT 107
  RESET REFERENCE BIT 107
  SET CLOCK 108
  SET CLOCK COMPARATOR 108
  SET CPU TIMER 109
  SET PREFIX 109
  SET PSW KEY FROM ADDRESS 109
  SET STORAGE KEY 110
  SET SYSTEM MASK 110
  SIGNAL PROCESSOR 110
  STORE CLOCK COMPARATOR 111
  STORE CONTROL 111
  STORE CPU ADDRESS 112
  STORE CPU ID 112
  STORE CPU TIMER 113
  STORE PREFIX 113
  STORE THEN AND SYSTEM MASK 113
  STORE THEN OR SYSTEM MASK 114
  WRITE DIRECT 114

General Instructions 115
Data Format 116
Number Representation 116
Instructions 117
  ADD 117
  ADD HALFWORD 117
  ADD LOGICAL 120
  AND 120
  BRANCH AND LINK 121
  BRANCH ON CONDITION 121
  BRANCH ON COUNT 122
  BRANCH ON INDEX HIGH 122
  BRANCH ON INDEX LOW OR EQUAL 123
  COMPARE 123
  COMPARE AND SWAP 123
  COMPARE DOUBLE AND SWAP 124
  COMPARE HALFWORD 125
  COMPARE LOGICAL 125
  COMPARE LOGICAL CHARACTERS UNDER MASK 126
  COMPARE LOGICAL LONG 126
  CONVERT TO BINARY 127
  CONVERT TO DECIMAL 128
  DIVIDE 128
  EXCLUSIVE OR 128
  EXECUTE 129
  INSERT CHARACTER 130
  INSERT CHARACTERS UNDER MASK 130
  LOAD 130
  LOAD ADDRESS 131
  LOAD AND TEST 131
  LOAD COMPLEMENT 131
  LOAD HALFWORD 131
  LOAD MULTIPLE 132
  LOAD NEGATIVE 132
  LOAD POSITIVE 132
  MONITOR CALL 132
  MOVE 133
  MOVE LONG 133
  MOVE NUMERICS 135
  MOVE WITH OFFSET 135
  MOVE ZONES 136
  MULTIPLY 136
  MULTIPLY HALFWORD 136
  OR 137
  PACK 137
  SET PROGRAM MASK 138
  SHIFT LEFT DOUBLE 138
  SHIFT LEFT DOUBLE LOGICAL 139
  SHIFT LEFT SINGLE 139
  SHIFT LEFT SINGLE LOGICAL 139
  SHIFT RIGHT DOUBLE 140
  SHIFT RIGHT DOUBLE LOGICAL 140
  SHIFT RIGHT SINGLE 140
  SHIFT RIGHT SINGLE LOGICAL 141
  STORE 141
  STORE CHARACTER 141
  STORE CHARACTERS UNDER MASK 141
  STORE CLOCK 141
  STORE HALFWORD 142
  STORE MULTIPLE 142
  SUBTRACT 143
  SUBTRACT HALFWORD 143
  SUBTRACT LOGICAL 143
  SUPERVISOR CALL 144
  TEST AND SET 144
  TEST UNDER MASK 145
  TRANSLATE 145
  TRANSLATE AND TEST 145
  UNPACK 146

Decimal Instructions 147
Data Format 147
  Zoned Format 147
  Packed Format 147
Number Representation 148
Instructions 148
  ADD DECIMAL 149
  COMPARE DECIMAL 149
  DIVIDE DECIMAL 149
  EDIT 150
  EDIT AND MARK 152
  MULTIPLY DECIMAL 153
  SHIFT AND ROUND DECIMAL 153
  SUBTRACT DECIMAL 154
  ZERO AND ADD 155

Floating-Point Instructions 157
Data Format 157
Guard Digit 158

Number Representation 159
Normalization 159
Instructions 160
ADD NORMALIZED 160
ADD UNNORMALIZED 162
COMPARE 163
DIVIDE 163
HALVE 164
LOAD. 165
LOAD AND TEST 165
LOAD COMPLEMENT 165
LOAD NEGATIVE 166
LOAD POSITIVE 166
LOAD ROUNDED 166
MULTIPLY 167
STORE 168
SUBTRACT NORMALIZED 169
SUBTRACT UNNORMALIZED 169

Machine-Check Handling 171
Machine-Check Detection 171
Recovery Mechanisms 172
Redundancy Correction 172
CPU Retry 172
Unit Deletion 172
Handling of Machine Checks 172
Handling of Invalid CBC in Storage 172
Handling of Invalid CBC in Keys in Storage 173
Handling of Invalid CBC in Registers 173
Check-Stop State 175
Machine-Check Interruption Conditions 175
Machine-Check Interruption 175
Interruption Action 175
Point of Interruption 176
Machine-Check Logout 177
Machine-Check Extended Interruption Information. . . 177
Machine-Check Interruption Code 178
Machine-Check Control Registers 181
Control Register 14 181
Control Register 15 182
Summary of Machine-Check Masking 183

Input/Output Operations 185
Attachment of Input/Output Devices. 186
Input/Output Devices 186
Control Units 187
Channels 187
System Operation 189
Compatibility of Operation 191
Control of Input/Output Devices 191
Input/Output Device Addressing 192
States of the Input/Output System 192
Resetting of the Input/Output System 194
Condition Code 195
Instruction Formats 197
List of Instructions 197
CLEAR I/O 198
HALT DEVICE 199
HALT I/O 202
START I/O 204
START I/O FAST RELEASE 204
STORE CHANNEL ID 206
TEST CHANNEL 207
TEST I/O. 208
Input/Output Instruction Exception Handling 210
Execution of Input/Output Operations 210
Blocking of Data. 210
Channel Address Word 210
Channel Command Word 211
Command Code 212
Designation of Storage Area 212
Chaining 213
Skipping 215
Program-Controlled Interruption 215
Channel Indirect Data Addressing 216
Commands 217
Conclusion of Input/Output Operations 222
Types of Conclusion 222
Input/Output Interruptions 226
Priority of Interruptions 227
Channel Status Word 228
Unit Status Conditions 229
Channel Status Conditions 233
Contents of Channel Status Word 235
Channel Logout 238
I/O Communications Area 239

System Console 243
Operator Section 243
Address-Compare Controls 244
Check-Stop Indicator 245
Configuration Controls 245
Display-and-Enter Controls. 245
Emergency-Pull Switch 245
Enable-System-Clear Key 245
IMPL Controls 245
Interrupt Key 245
Load Indicator 245
Load Key. 245
Load-Unit-Address Controls 246
Manual Indicator 246
Power-Off Key 246
Power-On Key 246
Rate Control. 246
Restart Key 246
Start Key 247
Stop Key 247
Store-Status Key. 247
System Indicator. 247
System-Reset Key 247
Test Indicator 247
Thermal/CB Power-Check Indicator 247
TOD Clock Key 247
Wait Indicator 248
Remote Operator-Control Panel 248
Customer-Engineer Section. 248

Appendix A. System/370 Features 249
Central Processing Unit 249
I/O Channels 250
Availability of CPU Features on System/370 Models . . . 250

Appendix B. Functions That Differ From System/360 251
Removal of USASCII-8 Mode 251
Handling Invalid Decimal Sign 251
Recognizing Protection Exception in Edit 251
Operation Code for HALT DEVICE 251
Extent of Logout Area 251
Command Retry. 251
Logout on Channel Data Check 252
Channel Prefetching. 252

Appendix C. Lists of Instructions 253
Arranged by Name 254
Arranged by Mnemonic 258
Arranged by Operation Code 262
Arranged by Feature 266
Standard Instruction Set 266
Decimal Feature Instructions 268
Floating-Point Feature Instructions 269
CPU-Timer and Clock-Comparator Feature Instructions . 270
Direct-Control Feature Instructions 270
Extended-Precision Floating-Point Feature Instructions . 270
Translation Feature Instructions 270
Multiprocessing Feature Instructions 270
Conditional-Swapping Feature Instructions 270
PSW-Key-Handling Feature Instructions 270

Appendix D. Formats 271
Program Status Word 271
Assignment of Control Register Fields 272
Assigned Locations in Real Main Storage 273
Assigned Locations in Absolute Main Storage 274

Appendix E. Condition-Code Settings 275

Appendix F. Table of Powers of 2 277

Appendix G. Hexadecimal Tables 279

Appendix H. EBCDIC Chart 287

Appendix I. Number Representation and Instruction-Use Examples. 289

Index. 315

IBM System/370 is a product of the experience gained with System/360. It preserves compatibility forward from System/360--that is, makes it possible to move from System/360 to System/370 with the same ease that is possible in moving from a lower model to a higher model within the system--and at the same time includes advanced information processing capabilities.

The latest in solid-state and monolithic technologies is reflected in most phases of System/370 design. Complementing this design are new facilities that assist the user in extending his computer applications: dynamic address translation, channel indirect data addressing, multiprocessing, timing facilities, extended-precision floating point, program-event recording, monitoring, and the block-multiplexer channel--together with substantially improved reliability, availability, and serviceability.

- *Dynamic address translation*, a CPU facility that generally eliminates the need to assign a program to fixed locations in real main storage and thus reduces the addressing constraints on both system and problem programs, provides greater freedom in program design and permits a more efficient and effective utilization of main storage. When one of the operating systems for virtual storage is employed, dynamic address translation allows the use of up to 16,777,216 bytes of virtual storage.
- *Channel indirect data addressing*, a companion facility to dynamic address translation, provides assistance in translating data addresses for I/O operations. It permits a single channel command word to control the transmission of data that spans noncontiguous areas of real main storage.
- *Multiprocessing* provides for the interconnection of CPUs to enhance system availability and share data and resources. It includes facilities for shared main storage, for programmed and special machine signaling between CPUs, and for the programmed reassignment of the first 4,096 bytes of real main storage for each CPU.
- *Timing facilities* include a time-of-day clock, a clock comparator, and a CPU timer, along with an interval timer that is also available in System/360. The time-of-day clock provides a measure of elapsed time suitable for the indication of date and time; it has a cycle of approximately 143 years and a resolution such that the incrementing rate is comparable to the instruction execution rate of the model. The clock comparator provides for an interruption when the time-of-day clock reaches a program-specified value. The CPU timer is a high-resolution timer that initiates an interruption upon being decremented past zero.
- *Extended-precision floating point* includes the facilities for addition, subtraction, and multiplication of floating-point numbers with a fraction of 28 hexadecimal digits. Included in the feature are instructions for rounding from extended to long and from long to short formats.
- *Program-event recording* provides program interruptions on a selective basis as an aid in program debugging.
- *Monitoring* provides for passing control to a monitoring program when selected indicators are reached in the monitored program. It can be used, for example, in analyzing which programs get executed, how often, and in what length of time.
- The *block-multiplexer channel*, which permits concurrent processing of multiple channel programs, provides an efficient means of handling I/O devices that transfer data on the I/O interface at a high rate but have relatively long periods of channel inactivity in between transfers.

To accommodate many of these new functions, program-addressable *control registers* and a new CPU mode -- *extended-control mode* -- are provided.

System/370 provides the capability of running System/360 operating systems, as well as those specifically designed for its advanced features, with little or no change in application programs and data. At the same time, System/370 provides several CPU models at different performance levels, making many of the new information processing capabilities available to all System/370 users--from the smallest to the very large. The wide choice of models and support, together with the new and advanced features, makes System/370 one of the most versatile systems offered today.

General-Purpose Design

Like System/360, System/370 is a general-purpose system that can readily be tailored for a variety of applications. A *standard* instruction set, which is expanded over that of System/360 and includes the protection facility, provides the basic processing capabilities of the system. To this, a decimal feature, which includes decimal shifting, may be added to provide a *commercial* instruction set. Joining the floating-point feature to the standard instruction set provides a *scientific* instruction set, which in turn can be augmented by the inclusion of the *extended-precision floating-point* feature. If both the decimal feature and the floating-point feature are installed with the standard instruction set, a *universal* instruction set is obtained. Adding other features, such as the conditional-swapping feature, extends the processing capabilities of the system still further.

Along with System/360, System/370 has the capability of addressing a main storage of 16,777,216 bytes, and the System/370 translation feature, used with appropriate programming support, provides a user with up to this maximum address space despite the attachment of a lesser amount of real main storage. This feature and this support permit a System/370 model with limited real main storage to be used for a much wider set of applications, and they make many applications with requirements for extensive main storage more practical and convenient. Additionally, for many System/370 models, the speed of accessing main storage is improved by the use of a cache. The cache is a buffer--not apparent to the user--that often provides information requested from main storage without the delay associated with accessing main storage itself.

Another major aspect of the general-purpose design of System/370 is the capability provided to attach a wide variety of I/O devices through several types of multiplexing channels. Like System/360, System/370 has a byte-multiplexer channel for the attachment of unbuffered devices and of a large number of communications devices. Additionally, System/370 offers a block-multiplexer channel, which is particularly well-suited for the attachment of buffered devices and high-speed cyclic devices.

An individual System/370 installation is obtained by selecting the system components best suited to the applications from a wide variety of alternatives in internal performance, functional ability, and input/output.

Compatibility

Although models of System/370 differ in implementation and physical capabilities, logically they are upward and downward compatible. That is, within the limitations of compatibility, as described below, any program gives identical results on any model. Compatibility allows for ease in systems backup, and simplicity in education.

The compatibility rule has four limitations:

1. The systems facilities used by the program should be the same in each case. For example, the optional CPU features and the storage capacity, as well as the quantity and type of I/O equipment, should be equivalent.
2. The program should be independent of the relation between instruction execution times, I/O data rates, access times, CCW execution times, and elapsed time values.
3. The program should not depend on functions that are identified in this manual as model-dependent, on results that are defined to be unpredictable, or on special-purpose functions that are not described in this manual.
4. The program should not use or depend on unassigned fields unless they are explicitly made available for program use. Additionally, the program should not be designed to cause interruptions by means of format errors, such as the use of invalid operation codes or invalid command codes.

System/370 is forward compatible from System/360, and System/360 programs that are to be run on System/370 must observe both the preceding limitations and the following three limitations:

1. The program must not use PSW bit 12 as an ASCII bit, which is a function that is provided only for System/360.
2. The program must not depend on main-storage locations assigned specifically for System/370, such as the interruption-code areas, the machine-check save areas, and the extended-logout area.
3. The program associated with input/output operations must take into account the effects of channel prefetching, command retry, logout on channel data check, and the operation-code assignment for HALT DEVICE.

System Program

The system is designed to operate with a supervisory program that coordinates the use of system resources and executes all I/O instructions, handles exceptional conditions, and supervises scheduling and execution of multiple programs.

System/370 can operate with several different types of supervisory programs. Some of these programs provide support for the new System/370 instructions, for the advanced reliability, availability, and serviceability features, and for the new I/O capabilities. Additionally, some of these programs provide for system and application programs to operate in a virtual-storage environment.

System/370 can also operate in the mode of System/360 and run all of the supervisory and application programs written for System/360 that satisfy the conditions described in "Compatibility."

Availability

Availability is the capability of a system to accept and successfully process an individual job. System/370 machine facilities permit increased availability by (1) allowing a larger number and a broader range of jobs to be processed concurrently, thus making the system more readily accessible to any particular job, and (2) limiting the effect of an error and identifying more precisely its cause, with the result that the number of jobs affected by errors is minimized and the correction of the errors is facilitated.

Several design aspects make this possible.

- A program is checked for the correctness of instructions and data as the program is executed, and program errors are indicated separately from equipment errors. Such checking and reporting assists in locating failures and isolating effects.
- The protection facility, in conjunction with dynamic address translation, permits the protection of the contents of main storage from destruction or misuse caused by erroneous or unauthorized storing or fetching by a program. This provides increased security for the user, thus permitting applications with different security requirements to be processed concurrently with other applications.
- Dynamic address translation allows isolation of one application from another, still permitting them to share common resources. Also, it permits the implementation of virtual machines, which may be used in the design and testing of new versions of operating systems along with the concurrent processing of application programs. Additionally, it provides for the concurrent operation of incompatible operating systems.
- Multiprocessing permits better use of storage and processing capabilities, more efficient communication between CPUs, and duplication of resources, thus aiding in the continuation of system operation in the event of machine failures.
- Monitoring, program-event recording, and the high-resolution timing facilities permit the testing and debugging of programs without manual intervention and with little effect on the concurrent processing of other programs.
- Emulation is performed under supervisory program control, thus making it possible to perform emulation concurrently with other applications.
- On most models, error checking and correction (ECC) in main storage, instruction retry, and command retry provide for circumventing intermittent equipment malfunctions, thus reducing the number of equipment failures.
- An enhanced machine-check handling mechanism provides model-independent fault isolation, which reduces the number of programs impacted by uncorrected errors. Additionally, it provides model-independent recording of machine-status information. This leads to greater machine-check handling compatibility between models and improves the capability for loading and running a program on a different model when a system failure occurs.
- A small number of manual controls are required for basic system operation, permitting most operator-system interaction to take place *via* a unit operating as an I/O device and thus reducing the possibility of accidental operator errors.

System Organization

Contents

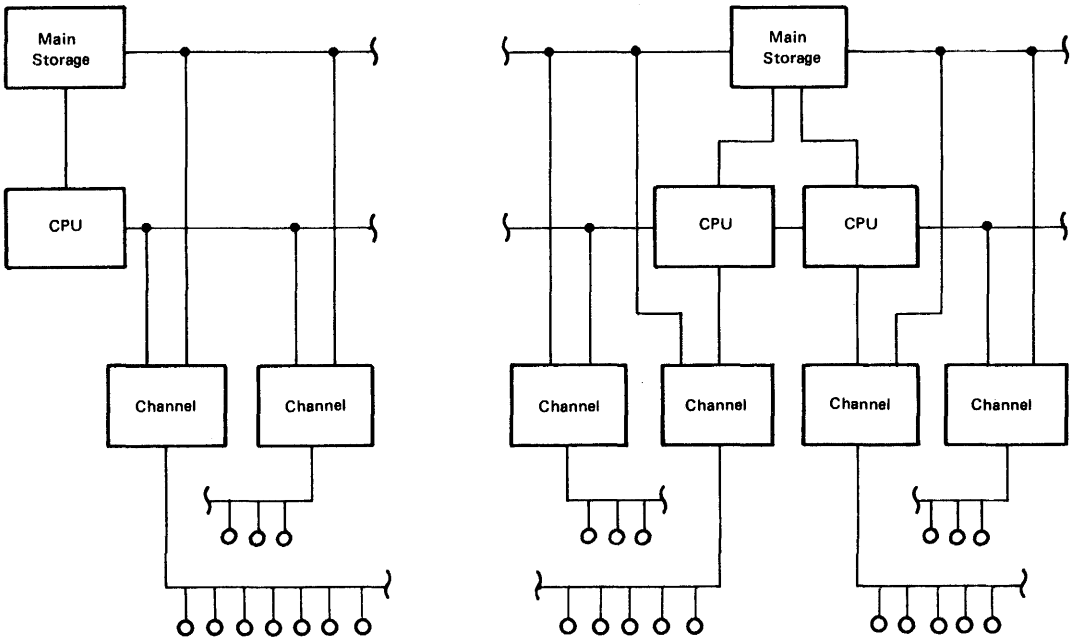
Main Storage 13
Information Formats 14
Addressing 14
Information Positioning 15
Integral Boundaries 15
Byte-Oriented-Operand Feature 15
Central Processing Unit 15
General Registers 16
Floating-Point Registers 16
Control Registers 16
Input and Output 17
Channels 17
Input/Output Interface 17
Input/Output Devices and Control Units 18
System Console 18

Logically, System/370 consists of main storage, a central processing unit (CPU), selector and multiplexer channels, and input/output devices, usually attached to channels through control units. The physical identity of these functions may vary between models. It is possible for systems to communicate with each other by means of shared I/O devices, a channel, or shared storage. The accompanying illustration depicts the logical structure for a single-CPU system and for a two-CPU multiprocessing system.

Main Storage

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

Main storage may be either physically integrated with a CPU or may be constructed as standalone units. Additionally, main storage may be composed of large-volume storage and a faster access buffer storage, sometimes called a cache. Each CPU may have an associated cache. The effects, except on



IBM System/370 Logical Structures

performance, of the physical construction and the use of distinct storage media are not observable by the program.

Fetching and storing of data by the CPU are not affected by any concurrent I/O data transfer or by concurrent reference to the same storage location by another CPU. When concurrent requests to a main-storage location occur, access normally is granted in a sequence that assigns highest priority to references by channels and that alternates priority between CPUs. If the first reference changes the contents of the location, any subsequent storage fetches obtain the new contents.

Main storage may be volatile or nonvolatile. If it is volatile, the contents of main storage are not preserved when power is turned off. If it is nonvolatile, turning power off or on does not affect the contents of main storage, provided the CPU is in the stopped state and no references are made to main storage by channels when power is turned off. In both types of main storage, the contents of the keys in storage associated with protection are not necessarily preserved when the power for main storage is turned off.

Information Formats

The system transmits information between main storage and a CPU or 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 consecutively, left to right, 0 through 7. Within any program format or any fixed-length operand 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 are transmitted with each byte or with a group of bytes. The check bits are generated automatically by the system and cannot be directly controlled by the program. References in this manual to the size of data fields and registers exclude mention of the associated check bits. All storage capacities are expressed in number of bytes provided, without regard to storage width.

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* 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 either 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 main 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 may be wider than the field being stored.

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. The addressing arrangement uses a 24-bit binary address to accommodate a maximum of 16,777,216 byte addresses.

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

For purposes of addressing main storage, three types of addresses are recognized: absolute, real, and logical.

Absolute addresses are the lowest level of program-recognizable addresses, and in this manual they are considered to be the addresses of actual storage locations. On some models, storage-configuration controls may be provided which permit the operator to change the correspondence between absolute addresses and the actual physical storage locations. However, at any one time, a physical storage location is not associated with more than one absolute address.

When the multiprocessing feature is included in a CPU, an address reassignment mechanism is provided that permits the first 4,096 bytes of real main storage for each CPU to be assigned to different absolute storage locations. This reassignment mechanism is called "prefixing." Most addresses generated in the CPU are monitored by the prefixing mechanism and reassigned when necessary. Addresses subject to monitoring by the prefixing mechanism are referred to as "real" addresses. When prefixing is not installed, a real address and the corresponding absolute address are identical.

When dynamic address translation is invoked, addresses specified by the program are normally translated to real addresses before main storage is accessed. The address specified by the program is referred to as a logical address. When dynamic address translation is not invoked, a logical address and the corresponding real address are identical.

All CPUs and channels having access to a common main-storage location have access to the entire 2,048-byte block containing that location and the associated key in storage. All CPUs and channels refer to a shared main-storage location by using the same absolute address.

Available storage is normally assigned to contiguous absolute addresses starting at address 0, and is always assigned in multiples of 2,048 bytes. An exception condition is recognized when an attempt is made to access main storage by using an absolute address that does not correspond to a physical location. Normally, the exception condition is recognized only when the information associated with the absolute address is actually required and not when the operation can be completed without using the information.

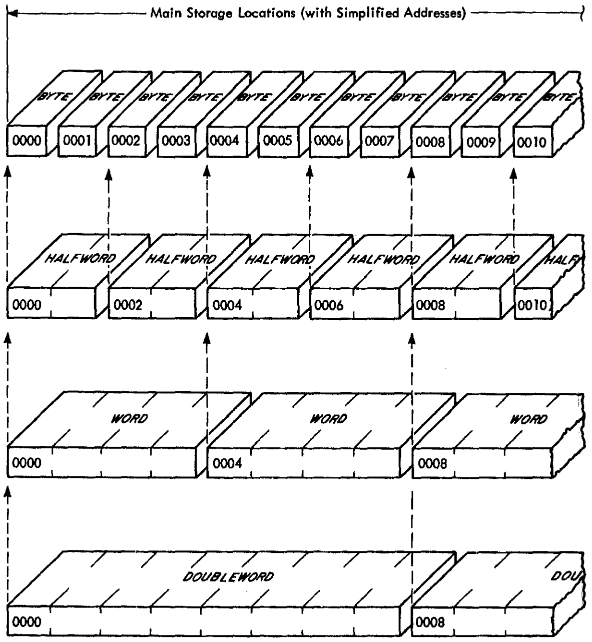
Information Positioning

Integral Boundaries

Certain units of information must be located in main 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. For example, a word (four bytes) is on an integral boundary when it is located in storage so that its address is a multiple of the number 4. A halfword (two bytes) is on an integral boundary when it has an address that is a multiple of the number 2, and a doubleword (eight bytes) is on an integral boundary when it has an address that is a multiple of the number 8.

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 operands of certain privileged instructions must appear on integral boundaries.



Integral Boundaries for Halfwords, Words, and Doublewords

Byte-Oriented-Operand Feature

The byte-oriented-operand feature is standard on System/370. This feature permits storage operands of most unprivileged operations to appear on any byte boundary.

The feature does not pertain to instruction addresses, or to the operands for COMPARE AND SWAP (CS) and COMPARE DOUBLE AND SWAP (CDS). Instructions must appear on even byte boundaries. The low-order bit of a branch address must be zero, and the instruction EXECUTE must designate the subject instruction at an even byte address. COMPARE AND SWAP must designate a word boundary, and COMPARE DOUBLE AND SWAP must designate a doubleword boundary.

Programming Note

Significant performance degradation is possible when storage operands are not positioned at addresses that are integral multiples of the operand length. To ensure optimum performance, storage operands should be aligned on integral boundaries, and the use of unaligned operands should be reserved for exceptional cases.

Central Processing Unit

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

The physical makeup of the CPU controls in the various models of the System/370 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: system-control, general, decimal, floating-point, and input/output instructions. The system-control and input/output instructions are privileged instructions that can be executed only when the CPU is in the supervisor state. The general instructions are used in performing fixed-point, logical, branching, and other control and data-manipulation operations. The decimal instructions operate on data in the decimal format, and the floating-point instructions on data in the floating-point format.

To perform its functions, the CPU uses a certain amount of internal storage other than main storage. Portions of this storage can be designated by the program, such as the current program status word (PSW), the general registers, the floating-point registers, the control registers, the prefix register, 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 system in relation to the program currently being executed. Registers associated with the timing facilities contain the time-of-day clock, the clock comparator, and the CPU timer. The general, floating-point, and control registers are discussed separately in the following paragraphs. The instruction operation code determines which type of register is to be used in an operation.

General Registers

The CPU can address information in 16 general registers. The general registers can 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 (see accompanying illustration). Some instructions provide for addressing multiple general registers by having several R fields.

For some operations, two adjacent general registers are coupled together, providing a 64-bit format. In these operations, the program must designate an even-numbered register, which contains the high-order bits. The next higher numbered register contains the low-order 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 X or B field specifies no index or base is to be applied, and, thus, general register 0 cannot be designated as containing an index or base address.

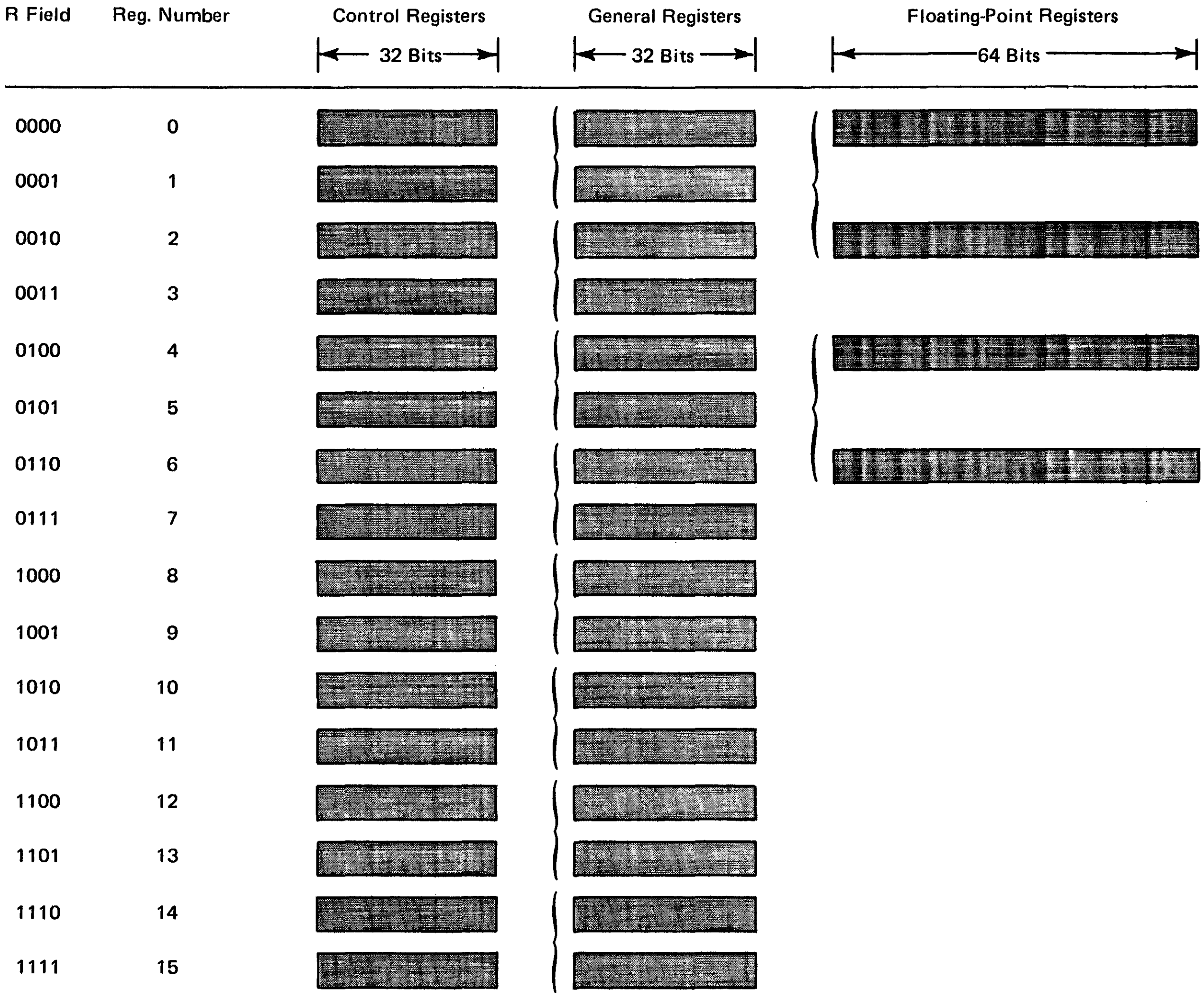
Floating-Point Registers

Four floating-point registers are available for floating-point operations. They are identified by the numbers 0, 2, 4, and 6 (see illustration). Each floating-point register contains 64 bits and can contain either a short (32-bit) or a long (64-bit) floating-point operand. A short operand occupies the high-order bit positions of a floating-point register. The low-order portion of the register is ignored and remains unchanged in arithmetic calling for short operands. Two pairs of adjacent floating-point registers can be used for extended operands: registers 0, 2, and registers 4,6. Each of these pairs provides a 128-bit format.

Control Registers

The CPU can designate 16 control registers, each 32 bit positions in length. The bit positions in the registers are assigned to particular facilities in the system, such as program-event recording, and are used either to specify whether an operation can take place or to provide special information required by the facility. On any particular model, only those bit positions are necessarily provided which are required by the installed facilities.

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



Note: The braces indicate that the two registers may be coupled as a double-register pair, designated by the R field of the lower-numbered register. For example, the general register pair 0 and 1 is designated by the R field of register 0.

General, Floating-Point, and Control Registers

Input and Output

Input/output (I/O) operations involve the transfer of information between main storage and an I/O device. I/O devices attach to channels, which control the transfer of data between the devices and main storage.

Channels

The channel connects with the CPU and main storage and, usually by means of the I/O interface, with control units. The 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 may be an independent unit, complete with necessary logical and 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.

System/370 has three types of channels: byte-multiplexer, block-multiplexer, and selector channels.

Input/Output Interface

For most devices, communication between the control unit and the channel takes place over a connection called the *I/O interface.* The I/O interface provides an information format and control signal sequences that are independent of the type of control unit and channel and provide a uniform means of attaching and controlling various types of I/O devices.

I/O devices that do not use the I/O interface employ the same information format and control signal sequences.

Input/Output Devices and Control Units

Input/output devices include such equipment as card readers and punches, magnetic tape units, disk storage, drum storage, typewriter-keyboard devices, printers, teleprocessing devices, and sensor-based equipment.

Many I/O devices function with an external document, such as a punched card or a reel of 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.

System Console

The system console provides the functions necessary to operate and control the system. It consists of a system control panel and, in most cases, an associated console device, which may also be used as an I/O device for communicating with the supervisory program and problem programs. The need for operator manipulation of manual controls is held to a minimum by the system design and the governing supervisory program.

The main functions provided by the system console include power-on/off, reset, initial-program-loading, start/stop, and display and enter functions.

Program Execution

Contents

Instructions . . . 19
Operands . . . 19
Instruction Format . . . 20
Register Operands . . . 20
Immediate Operands . . . 20
Storage Operands . . . 20
Address Generation . . . 21
Program Status Word . . . 22
Instruction Execution . . . 22
Branching . . . 22
Interruptions . . . 22
Sequence of Storage References . . . 23
Instruction Fetch . . . 23
DAT Table Fetches . . . 24
Key-in-Storage Accesses . . . 24
Storage-Operand References . . . 25
Storage-Operand Fetch References . . . 25
Storage-Operand Store References . . . 25
Storage-Operand Update References . . . 25
Storage-Operand Consistency . . . 26
Single-Access References . . . 26
Block-Concurrent References . . . 27
Consistency Specification . . . 27
Relation Between Operand Accesses . . . 27
Serialization . . . 28

Normally, operation of the CPU is controlled by instructions taken in sequence. This sequence is governed by the program status word (PSW), which contains the primary information required for proper program execution. 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, 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 main 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 operation 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 operation code.

Immediate operands are contained within the instruction, and the eight-bit field containing the immediate operand is called the I field.

Operands in main 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 main 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:

- Specify a complete address by using an abbreviated notation.
- Perform address manipulation using instructions which employ general registers for operands.
- Modify addresses by program means without alteration of the instruction stream.
- Operate independently of the location of data areas by directly using addresses received from other programs.

The address used to refer to main 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 name, 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 main storage on an integral halfword boundary. Each instruction is in one of six basic formats: RR, RX, RS, SI, S, and SS, with two variations of SS.

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 such exceptions are explicitly identified in the individual instruction descriptions.

The format names express, in general terms, the classes of operands which participate in the operation: RR denotes a register-to-register operation; RX, a register-and-indexed-storage operation; RS, a register-and-storage operation; SI, a storage-and-immediate operation; and SS, a storage-to-storage operation. The S format denotes an operation using an implied operand and storage.

The first byte and, in the S format, the first two bytes of an instruction contain the operation code (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 operation code specify the length and format of an 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/RX
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.

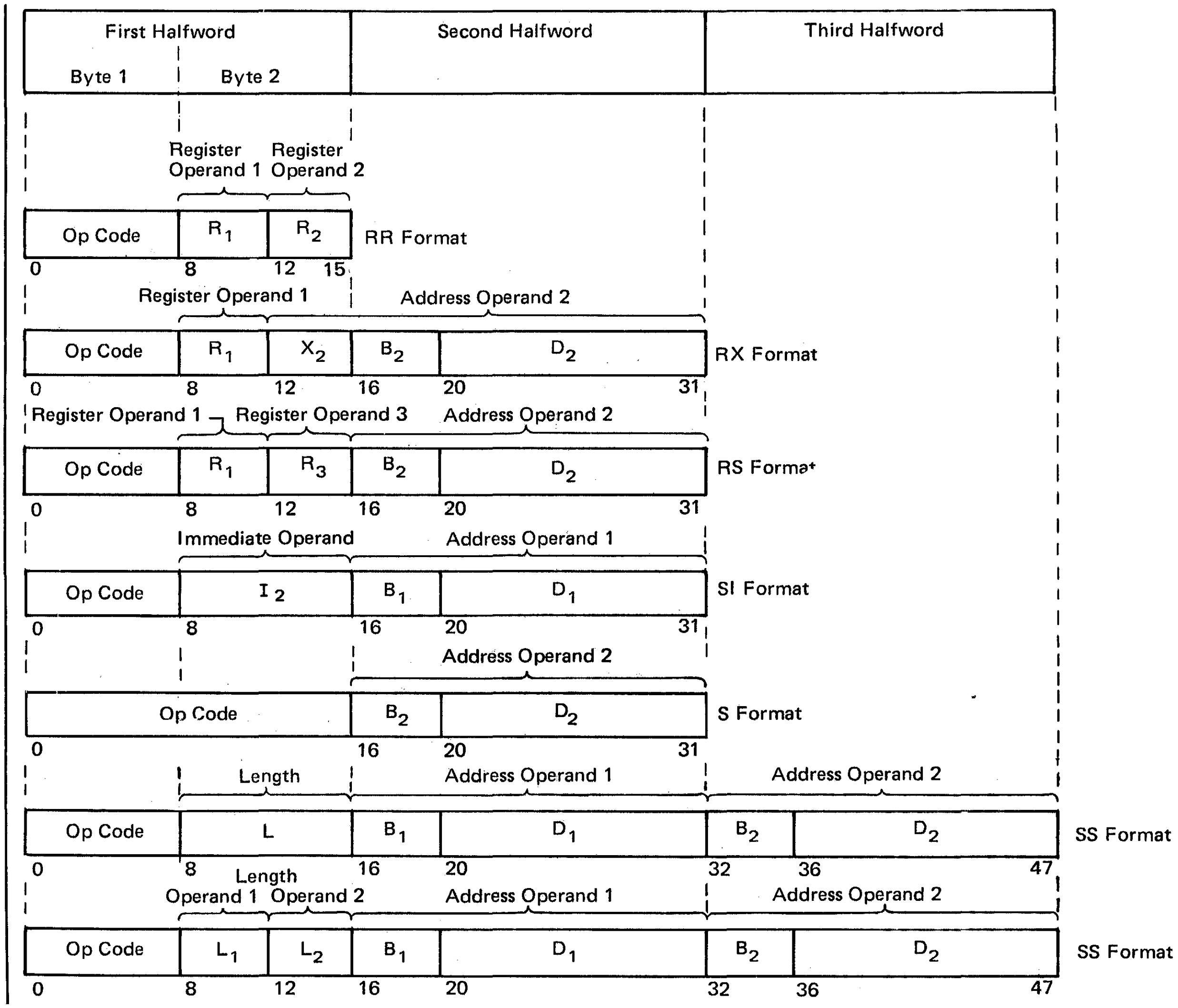
Immediate Operands

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. In the event the first operand is longer than the second,



Six Basic Instruction Formats

the second operand is extended with high-order zeros up to the length of the first operand. Such 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.

Address Generation

The address used to refer to main storage either is contained in a register designated by the R field in the instruction or is calculated from the following three binary numbers:

Base Address is a 24-bit number contained in a general register specifed 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 main storage. The base address may also be used for indexing purposes.

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.

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 unsigned 24-bit positive binary integers. The displacement is similarly treated as a 12-bit positive 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 X_2, B_1, or B_2 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 main storage. When a main-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.

Program Status Word

The program status word (PSW) is 64 bits in length and contains the information required for proper program execution. The PSW includes the instruction address, condition code, and other fields. In general, the PSW is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently being executed. The active or controlling PSW is called the current PSW. By storing the current PSW during an interruption, the status of the CPU can be preserved for subsequent inspection. By loading a new PSW or part of a PSW, the state of the CPU can be initialized or changed.

Instruction Execution

In program execution, the 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.

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 and BRANCH ON INDEX are provided. These branches, being specialized, provide increased performance for these tasks.

Interruptions

The interruption system permits the CPU to change state as a result of conditions external to the system, in input/output (I/O) units, 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 real main-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 status information of the CPU 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, and the execution of the instruction specified by a successful branch follows the execution of the branch. Similarly, an interruption takes place between executions of instructions.

The sequence of events implied by the processing just described is sometimes called the *conceptual* sequence or *conceptual* order.

Even though physical storage width and overlap of instruction execution with storage accessing may cause actual processing to be different, as observed by a CPU itself, each operation is performed sequentially, with one instruction being fetched after the preceding operation is completed and before the execution of the current operation is begun. With certain exceptions discussed in the section "Interlocks Between Logical and Real Storage References" in the chapter "Dynamic Address Translation," the results generated are those that would have been obtained had the operation been performed in the conceptual sequence. Thus, it is possible to modify an instruction in storage by the immediately preceding instruction.

In very simple machines in which operations are not overlapped, the conceptual and actual order 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 order to differ considerably from the conceptual order. 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 order. However, as observed by channels and other CPUs, the sequence may appear to differ from the conceptual order.

When only a single CPU is involved, 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 may consist 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 and storing a result. As a consequence, another CPU or a channel may be able to observe intermediate, or partially completed, results.

When the program on one CPU interacts with a program on a channel or another CPU, 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, DAT table fetches, storage operand references, and key-in-storage accesses.

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 subject 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 availability of dynamic-address-translation tables or for inspection for program-event exceptions, and it may be refetched for actual execution.

Instructions are not necessarily fetched in the order 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 or by other CPUs does not necessarily change the copy of prefetched instructions. However, if a store that is conceptually earlier occurs on the same CPU using the same logical address as that by which the instruction is fetched, the updated information is obtained.

All copies of prefetched instructions are discarded when the CPU enters or leaves translation mode, when the DAT parameters are changed in control registers 0 and 1 at a time when translation mode is specified, by a serializing operation, and as the CPU enters the operating state.

Programming Notes

As observed by a CPU itself, instruction prefetching is not normally apparent; the only exception occurs when more than one logical page address is translated to a single real page address. This is described in the section "Interlocks Between Logical and Real Storage References" in the chapter "Dynamic Address Translation."

The following are some effects of instruction prefetching on the execution of a program as viewed by another CPU.

If a program in one CPU changes the contents of a storage location and then sets a flag to indicate that the change has been made, a program in another CPU can test and find the flag set but subsequently can branch to the modified locations and execute their original contents. Additionally, when a channel or another CPU modifies an instruction, it is possible for a CPU to recognize the changes to some but not all bit positions of the instruction.

It is possible for a CPU to prefetch an instruction and subsequently, before the instruction is executed, for another CPU to change the key in storage. As a result, a CPU may appear to execute instructions from a storage location that is protected.

DAT Table Fetches

Fetching of dynamic address translation (DAT) table entries may occur as follows:

1. DAT entries may be prefetched into the translation-lookaside buffer (TLB) and used from the TLB without refetching from storage, until a PURGE TLB (PTLB) instruction is executed. DAT entries may be fetched at any time they are attached and valid, including during the execution of conceptually previous instructions, and are not necessarily fetched in the order conceptually called for.
2. A DAT table entry may be fetched piecemeal, a byte at a time, from main storage. However, no operand stores by this CPU or any other CPU are permitted, to the same location, between the fetches of the bytes.
3. A DAT table entry may be fetched even after some operand references for the instruction have already occurred. The fetch may occur as late as just prior to the actual byte access requiring the DAT entry.
4. A DAT table entry may be fetched for each use of the address, including pretesting, if performed, and for each reference to each byte of each operand.
5. The DAT page-table-entry fetch precedes the reference to the page. When a page-table entry goes from inactive to active status, the fetch of the associated segment-table entry precedes the fetch of the page-table entry.

For translation of the second operand of LOAD REAL ADDRESS, the segment-table-entry fetch precedes the page-table-entry fetch. The entries are fetched using the same rule as (2) above. The relationship of these two fetches to other references follows the rules for storage-operand fetches.

Key-in-Storage Accesses

References to the key in storage are handled as follows:

1. Whenever a reference to main storage is made and protection applies to the reference, the five access control bits associated with the storage location are inspected concurrently with the reference to the storage location.
2. When storing is performed, the change bit is set in the associated key in storage concurrently with the store operation.
3. The instruction SET STORAGE KEY causes the five access control bits and the change bit to be set concurrently in the key in storage. The access to the key in storage 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 provides a consistent image of the field consisting of the five access control bits and the change bit. The access to the key in storage for INSERT STORAGE KEY follows the sequence rules for storage-operand fetch references, and is a single-access reference.
5. The instruction RESET REFERENCE BIT modifies only the reference bit. All other bits of the key in storage remain unchanged. The access to the key in storage for RESET REFERENCE BIT follows the sequence rules for

storage-operand update references. The reference bit is the only bit which is updated.

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.

Storage-Operand References

A storage-operand reference is the fetching or storing of the explicit operand or operands in the main-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 main storage. Stores caused by channels or by other CPUs 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 the individual instruction definition 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 main storage. Unless otherwise specified, the bytes are not necessarily fetched in any particular order. The fetch reference for the operands of some instructions is specified to be concurrent within a block. In this case, no stores by any other CPU are permitted, to the same location, between the fetches of the bytes within a block.

Storage-Operand Store References

When the bytes of a storage operand participate in the instruction execution only 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 the individual instruction definition 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 main storage. Unless otherwise specified, the bytes are not necessarily stored in any particular order. The store reference for some instructions is specified to be concurrent within a block. In this case, no stores or fetches by any other CPU are permitted, to the same location, between the stores of bytes within a block.

A CPU may delay storing results into main 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 main storage. The results of one instruction are placed in main storage after the results of all preceding instructions have been placed in main storage and before any results of the succeeding instructions are stored. The results of any one instruction are stored in the order specified for that instruction.

A CPU does not fetch operands, or dynamic-address-translation table entries, from a main-storage location until all information destined for that real main-storage location by that CPU has been placed in main storage. Prefetched instructions may appear to be updated prior to the information appearing 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. In most cases, no special interlock is provided between the fetch and store, and accesses by channels and other CPUs 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 made contiguously, and it is possible for another CPU or channel to make one or more interleaved accesses to the same location. The interleaved accesses can be either fetches or stores and can be associated with either an update or an interlocked-update reference.

Three instructions perform an update which is interlocked against accesses to the same location during the execution of the instruction. The instruction TEST AND SET (TS) causes an interlocked update, and the instructions COMPARE AND SWAP (CS) and COMPARE DOUBLE AND

SWAP (CDS) cause an interlocked update when they set condition code 0.

The fetch and store accesses associated with an interlocked-update reference are not necessarily made contiguously, but restrictions are made on accesses to the location. The fetch access of an interlocked update by another CPU, and all store accesses by another CPU, are prevented from occurring between the fetch and the store accesses of an interlocked update. CPU fetches which are not part of an interlocked update, including the fetches of a CS or CDS instruction which results in condition code 1, may be made from the location during the interlock period. I/O accesses, either fetch or store, may occur during the interlock period.

Within the limitations of the above requirements, the fetch and store accesses associated with an update follow the same rules as the fetches and stores described in the previous sections.

Programming Notes

When two CPUs attempt to update information at a common main-storage location by an instruction that causes fetching and subsequently storing of the updated information, it is possible for both CPUs to fetch the information and subsequently take the store access. The change made by the first CPU to store the result in such a case is lost. Similarly, if one CPU updates the contents of a field but another CPU makes a store operation to that field between the fetch and store parts of the update reference, the effect of the store is lost. If, instead of a store access, a CPU makes an interlocked-update reference to the common storage field between the fetch and store portions of an update due to another CPU, any change in the contents produced by the interlocked update is lost.

Only those bytes which are included in the result field of both operations are considered to be part of the common main-storage location. However, all bits within a common byte are considered to be common even if the bits modified by the two operations do not overlap. As an example, if one CPU executes the instruction OR (OI) with the value 80 (hex) in the immediate field and the other CPU executes AND (NI) with an immediate operand of FE (hex) on the same byte, one of the updates can be lost.

When the store access is part of an update reference by the CPU, the execution of the storing is not contingent on whether the information to be stored is different from the original contents of the location. In particular, the contents of all designated byte locations are replaced, and, for each byte in the field, the entire contents of the byte are replaced.

An access to store information is performed, for example, in the following cases:

a. Execution of the OR instruction (OI or OC) with a second operand of all zeros.

b. Execution of OR (OC) with the first- and second-operand fields coinciding.

c. For those locations of the first operand of TRANSLATE where the argument and function values are the same.

The instructions TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP facilitate updating of a common storage field by two CPUs. In order for the change by either CPU not to be lost, both CPUs must use an instruction providing an interlocked update. It is possible, however, for a channel to make an access to the same storage location between the fetch and store portions of an interlocked update.

Storage-Operand Consistency

Single-Access References

With the exception of instructions operating on decimal data, storage-operand references are single-access references. A fetch reference is said to be a single-access reference if the result of the operation comprises a value 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 the fetch and store accesses are each single-access.

The storage references associated with the following instructions are not necessarily single-access references: the decimal-feature instructions and the instructions CONVERT TO BINARY, CONVERT TO DECIMAL, MOVE WITH OFFSET, PACK, and UNPACK.

When a storage-operand 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 or other CPUs.

Programming Note

When multiple fetch accesses are made to a single byte that is being changed by a channel or another CPU, 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.

Block-Concurrent References

For some references, the accesses to all bytes within a group of contiguous storage locations, or a block, appear to be concurrent to another CPU, but the accesses do not necessarily appear to include more than a byte at a time to I/O. When a fetch-type reference is concurrent within a block to CPUs, no store access by another CPU is permitted to the block during the time that bytes contained in the block are being fetched. I/O accesses may occur to the bytes within the block between the fetches. When a store-type reference is concurrent within a block to CPUs, no access, either fetch or store, is permitted to the block during the time that the bytes within the block are being stored. I/O accesses may occur to the bytes in the block between the stores.

Consistency Specification

The storage-operand references associated with all S format instructions and all RX format instructions with the exception of EXECUTE, CONVERT TO DECIMAL, and CONVERT TO BINARY, are block-concurrent, as observed by all CPUs, if the operand is addressed on a boundary which is integral to the size of the operand.

For the instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP all accesses to the storage operand appear to be concurrent as observed by all CPUs.

The under-mask instructions COMPARE LOGICAL CHARACTERS UNDER MASK, INSERT CHARACTERS UNDER MASK, and STORE CHARACTERS UNDER MASK, and the instructions LOAD MULTIPLE and STORE MULTIPLE, access the storage operand in a left-to-right direction, and all bytes accessed within each doubleword appear to all CPUs to be accessed concurrently.

When destructive overlap does not exist, the operands of MOVE (MVC) are accessed as follows:

- The first operand is accessed in a left-to-right direction, and all bytes accessed within a doubleword appear to all CPUs to be accessed concurrently.
- The second operand is accessed left to right, and all bytes within a doubleword in the second operand that are moved into a single doubleword in the first operand appear to all CPUs to be fetched concurrently. Thus, if the first and second operands begin on the same byte offset within a doubleword, the second operand appears to be fetched doubleword-concurrent. If the offsets within a doubleword differ by four, the second operand appears to be fetched word-concurrent.

Destructive overlap is said to exist when the result location is used as a source after the result has been stored, assuming processing to be performed a single byte at a time.

The operands for MOVE LONG and COMPARE LOGICAL LONG appear to all CPUs to be accessed doubleword-concurrent when both operands start on doubleword boundaries and are an integral number of doublewords in length, and, for MOVE LONG, the operands do not overlap.

For EXCLUSIVE OR (XC), when the first and second operands coincide, the operands appear to all CPUs to be accessed doubleword-concurrent.

Programming Note

It should be noted that, in the case of XC designating operands which coincide exactly, the bytes within the field may appear to be accessed three times, by two fetches and one store: once as the fetch portion of the first operand update, once as the second-operand fetch, and then once as the store portion of the first-operand update. Each of the three accesses appears to all CPUs to be doubleword-concurrent, but the three accesses do not necessarily appear to occur one immediately after the other.

Relation Between Operand Accesses

Storage-operand fetches associated with one instruction execution precede all storage-operand references for conceptually subsequent instructions. A storage-operand store specified by one instruction precedes 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 does precede a conceptually subsequent storage-operand fetch to the same real 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 Notes

The independent fetching of a single location for each of two operands may affect the program execution in the following situation.

When the same main-storage location is designated by two operand addresses of an instruction, and a channel or another CPU 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.

Serialization

All interruptions, and the execution of certain instructions, cause serialization of CPU operation. Execution of a serialization function consists in completing all conceptually prior storage accesses by this CPU, as observed by channels and other CPUs, before the conceptually following storage accesses occur. Serialization affects the order of all accesses by this CPU to storage and to the key in storage, except for those associated with DAT-table-entry fetch.

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

1. These general instructions: BRANCH ON CONDITION (BCR) with the R_1 and R_2 fields containing all ones and all zeros, respectively, and COMPARE AND SWAP, COMPARE DOUBLE AND SWAP, STORE CLOCK, SUPERVISOR CALL, and TEST AND SET.
2. LOAD PSW.
3. PURGE TLB and SET PREFIX , which also cause the translation-lookaside buffer to be purged.
4. All I/O instructions.
5. The signaling instructions: READ DIRECT, WRITE DIRECT, and SIGNAL PROCESSOR.

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

- All conceptually previous CPU storage accesses by this CPU are completed, as observed by channels and other CPUs. This includes all conceptually previous stores and changes to keys in storage.
- 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 LPSW and SPX, in which the operands 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 continue.

A serializing function affects the order of storage accesses that are under the control of the CPU in which the serializing function takes place. It does not affect the order of storage accesses caused by a program in a channel or another CPU.

Programming Notes

The following are some effects of a serializing operation:

1. When an instruction changes the contents of a storage location that is used as a source of a following instruction and when different addresses are used to designate the location for storing the result and fetching the instruction, a serializing operation following the change ensures that the modified instruction is executed.
2. When a serializing operation takes place, channels and any other CPU observe instruction and operand fetching and result storing to take place in the order established by the serializing operation.

Storing into a location from which a serializing instruction is fetched does not necessarily affect the execution of the serializing instruction unless a serializing function has been performed after the storing and before the execution of the serializing instruction.

System Control

Contents

CPU States 30
Wait and Running States 30
Problem and Supervisor States 30
Stopped and Operating States 30
Control Modes 31
BC Mode 32
EC Mode 32
Set-System-Mask Suppression 32
Program Status Word 32
Program Status Word Format in BC Mode 33
Program Status Word Format in EC Mode 34
Exceptions Associated with the PSW 35
Early Exception Recognition 35
Late Exception Recognition 35
Control Registers 36
Key in Storage 38
Protection 38
Protection Action 38
Accesses Protected 39
Monitoring 39
Program-Event Recording 39
Control Register Allocation 40
Operation 40
Identification of Cause 41
Priority of Indication 41
Storage Area Designation 42
Program Events 42
Successful Branching 42
Instruction Fetching 42
Storage Alteration 42
General-Register Alteration 43
Indication of Events Concurrently with Other Interruption Conditions 43
Direct Control 46
Time-of-Day Clock 46
Format 46
States 46
Setting and Inspection of Value 47
Clock Comparator 47
CPU Timer 48
Interval Timer 49
Externally Initiated Functions 50
Resets 50
CPU Reset 51
Initial CPU Reset 51
I/O System Reset 51
Program Reset 51
Initial Program Reset 52
System-Clear Reset 53
Power-On Reset 53
Store Status 54
Initial Program Loading 54

This chapter provides the detailed description of a number of facilities that provide for switching the status of the system, for protecting a program from interference by another program, for initiating certain operations externally, and, in general, for enhancing the efficiency, utility, and programmability of the system.

The information determining the state and operation of the CPU resides in the program status word (PSW) and in control registers. Additional status and control information appears in low-order locations of main storage. By providing a supervisor state and a set of instructions that are valid only in the supervisor state for changing the contents of the PSW and control registers, a means is provided for avoiding unauthorized or inadvertent change to the system state.

The protection facility permits the protection of the contents of main storage from destruction or misuse caused by erroneous or unauthorized storing or fetching by a program.

Four timing facilities are provided for measuring time: the time-of-day clock permits indication of calendar time with a resolution of 1 microsecond and a period in excess of one hundred years; the clock comparator permits a program to be alerted at a particular instant of real time; and the CPU timer and interval timer provide a means for a program to be alerted after a specified time interval has elapsed.

Additionally, the following three facilities are provided: monitoring, program-event recording, and direct control. The monitoring facility is useful for performing various measurement functions, whereas program-event recording provides a means to assist in debugging programs.

A set of externally initiated functions is provided for initializing the system or for inspecting its state. These functions include reset, store status, and initial program loading.

CPU States

Excluding facilities that are provided for the maintenance of equipment, three types of state alternatives in the CPU are distinguished: wait-running, problem-supervisor, and stopped-operating. These states differ in the way they affect CPU functions and in the way their status is indicated and switched.

Wait and Running States

In the wait state no instructions are processed, whereas in the running state instruction fetching and execution proceed 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 running state.

The wait state is indicated in the operator section of the system console by the wait light. No explicit operator control is provided for changing the state.

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

Problem and Supervisor States

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

In the supervisor state all instructions are 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 or model-dependent functions. The instructions that are not valid in the problem state are called *privileged* instructions; they include those which modify or inspect keys in storage, those which modify or inspect the system control fields in the PSW and in control registers, and those which pertain to timing facilities, prefixing, inter-CPU communication, 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 Notes

The CPU may be switched between wait-running and problem-supervisor states only by introducing an entire new PSW. This may be performed by LOAD PSW, an interruption (including a supervisor-call interruption), or initial program loading.

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

In the wait state the CPU does not make repeated references to main storage; therefore, wait state is suitable for delaying operation until an interruption occurs. References, however, may be made due to I/O operations and for updating the interval timer. To leave wait state without manual intervention, the CPU must be enabled for the interruption source.

Stopped and Operating 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 mask bits and in the manner specified by the setting of the rate control on the system console.

A change between the stopped and operating states can be effected by manual intervention or by use of the SIGNAL PROCESSOR instruction. The stopped state is not controlled or identified by a bit in the PSW.

The state of the CPU is changed from stopped to operating by the following events:

- When the start key on the system console is activated or when the CPU accepts the start order specified by a SIGNAL PROCESSOR instruction addressing this CPU. However, the effect of start is unpredictable when the stopped state has been entered by means of a reset.
- When a restart interruption occurs, either as a result of the activation of the restart key or the execution of the SIGNAL PROCESSOR restart order.
- When initial program loading is successfully completed.

The state of the CPU is changed from operating to stopped by the performance of the stop function. The execution of the stop function is initiated:

- When the stop key on the system console is activated while the CPU is in the operating state.
- When the CPU accepts a stop or stop-and-store-status order specified by a SIGNAL PROCESSOR instruction addressing this CPU while the CPU is in the operating state.
- When the CPU has finished the execution of an instruction 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 provided no interruptions are pending for which the CPU is enabled. In the case of the interruptible instructions, the amount of data processed in a unit of operation depends on the particular instruction and may depend on the model.

All interruptions pending and not disallowed 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 may be ignored or remain pending, the action being the same as when the CPU is disabled for the conditions.

The CPU is placed in the stopped state also:

- After the completion of CPU reset, 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 CPU reset is described in "Resets" in this chapter, and the stopping due to address comparison is described in "Address-Compare Controls" in the chapter "System Console."

Additionally, the CPU may, depending on the model, temporarily enter the stopped state when the restart interruption is initiated with the CPU in the operating state.

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

Two other alternatives to the stopped and operating states exist: the load state and the check-stop state. The CPU is in the load state during the initial-program-loading operation. The check-stop state is entered on certain types of machine malfunctioning and is described in the chapter "Machine-Check Handling."

A CPU may have other alternatives to the stopped and operating states for maintenance and diagnostic functions and for the purpose of displaying and entering information *via* the console.

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.

Programming Notes

Except for the relationship between execution time and real time, the execution of a program is not affected by stopping the CPU.

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 in the case of an unending interruption sequence resulting from a PSW with a format error or from a direct interruption condition, such as one due to the CPU timer.

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.

Control Modes

Two modes are provided for the formatting and use of control and status information: basic-control (BC) mode and extended-control (EC) mode. The mode is specified by the contents of bit position 12 of the program status word (PSW).

The two modes determine the allocation of bit positions within the PSW, the use of permanently assigned locations in main storage for storing the interruption code and the instruction-length code on some classes of interruptions, the controlling of I/O interruptions for channels 0-5, and the handling of reference and change bits by INSERT STORAGE

KEY. Furthermore, program-event recording and dynamic address translation can be specified only in the EC mode, as the corresponding control bits in the PSW are provided only in the EC mode.

BC Mode

In the BC mode, the PSW has the same format as in System/360, and, except for the old PSW stored on a machine-check interruption, the interruption code and the instruction-length code appear in the PSW. As in System/360, interruptions from channels 0-5 are subject to the control by PSW bits 0-5, and INSERT STORAGE KEY provides zeros in bit positions 29 and 30 that correspond to the reference and change bits. A number of additional permanently assigned storage locations, however, are used during interruptions associated with extended or new functions, including those for storing the machine-check interruption code and those for storing the monitor code and monitor class number as the result of a monitor-call event.

The BC mode is specified when PSW bit 12 is 0. The BC mode of operation is provided on all CPUs.

EC Mode

In the EC mode, fields for channel masks 0-5, for the interruption code, and for the instruction-length code have been removed from the PSW, and the program-mask and condition-code fields have been allocated different bit positions within the PSW. Two additional control bits have been introduced into the PSW--the program-event-recording mask and the translation-mode bit. The interruption code and instruction-length code have been assigned separate main-storage locations for certain classes of interruptions, and I/O interruptions from channels 0-5 are subject to control by PSW bit 6, as well as by channel masks in control register 2. The instruction INSERT STORAGE KEY provides the reference and change bits.

The EC mode is made available with the extended-control facility. It is specified when PSW bit 12 is one. When extended control is installed, the CPU can operate either in the BC mode or EC mode, depending on the value of PSW bit 12. When PSW bit 12 is one and extended control is not installed, a specification exception is recognized.

Programming Note

The choice between BC and EC 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 facilities that are not associated with the control bits provided in the PSW only in the EC mode, and it does not affect the validity of any instructions. Although dynamic address translation cannot be specified in the BC mode, the instructions LOAD REAL ADDRESS, RESET REFERENCE BIT, and PURGE TLB are valid and perform the specified function in the BC mode. 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 set of program masks regardless of the PSW bit positions occupied by the mask.

Set-System-Mask Suppression

When the SSM-suppression bit, bit 1 of control register 0, is one, the execution of SET SYSTEM MASK is suppressed, and a program interruption for a special-operation exception occurs. The initial value of the SSM-suppression bit is zero.

The SSM-suppression control, when installed, is effective in the BC, as well as the EC, mode.

Programming Note

The facility to suppress the execution of SET SYSTEM MASK may be used to assist in converting a program written for BC-mode PSW to operate with an EC-mode PSW.

Program Status Word

The program status word (PSW) contains the control information that is switched by an interruption. Additional control and status information is contained in control registers and permanently assigned main-storage locations.

In certain circumstances all of the PSW is stored or loaded; in others, only part of it. The entire PSW is stored and a new PSW is introduced when the CPU is interrupted. The instruction LOAD PSW introduces a new 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 introduce new bits into 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 stores the PSW key; STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK store the leftmost byte of the PSW; and BRANCH AND LINK stores the instruction-length code, condition code, program mask, and instruction address.

The new PSW as introduced by an interruption or instruction becomes active (that is, the information introduced into the current PSW assumes control over the system) at the completion of the interruption or at the completion of the execution of the instruction, respectively. The interruption for program-event recording associated with an instruction that changes the PSW occurs under control of the PSW mask that is effective at the beginning of the operation.

The figures below show PSW formats in the BC and EC modes.

Program Status Word Format in BC Mode

The BC mode is specified by a zero in PSW bit position 12. The following is a summary of the functions of the PSW fields.

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

Input/Output 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 the channel-mask bit is zero, the channel cannot cause an I/O interruption; when the channel-mask bit is one, a condition at the channel can cause an interruption.

External Mask (E): Bit 7 controls whether the CPU is enabled for interruption by conditions included in the external class. When the bit is zero, an external interruption cannot occur. When the bit is one, an external interruption is subject to the corresponding external subclass-mask bits in control register 0: when the subclass-mask bit is zero, conditions associated with the subclass cannot cause an interruption; when the subclass-mask bit is one, an interruption in that subclass can occur.

Protection Key: Bits 8-11 form the CPU protection key. The key is matched with a key in storage whenever information is stored, or whenever information is fetched from a location that is protected against fetching.

Channel Masks 0-5	I O	E	Key	0	M	W	P	Interruption Code	
0	6		8	12				16	31

ILC	CC	Program Mask	Instruction Address	
32	34	36	40	63

PSW Format in BC Mode

0	R	0	0	0	T	I O	E	Key	1	M	W	P	0 0	CC	Program Mask	0 0 0 0 0 0 0 0	
0								8	12				16	18	20	24	31

0 0 0 0 0 0 0 0	Instruction Address	
32	40	63

PSW Format in EC Mode

Extended Control Mode: Bit 12 controls the format of the PSW and the mode of operation of the CPU. When the bit is zero, the PSW format and the CPU operation are as defined for the basic control (BC) mode. When the bit is one, the extended control (EC) mode is specified.

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, and interruptions due to other machine-check 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 in the running 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 stored on 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:

Program Mask Bit	Program Exception
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.

Program Status Word Format in EC Mode

The EC mode is specified by a one in PSW bit position 12. The following is a summary of the functions of the PSW fields:

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

Translation Mode (T): Bit 5 controls whether implicit translation of storage addresses by use of segment and page tables takes place. When the bit is zero, storage addresses are not translated. When the bit is one, the dynamic-address-translation mechanism is invoked.

Input/Output 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 the channel-mask bit is zero, the channel cannot cause an interruption; when the channel-mask bit is one, a condition at the channel can cause an interruption.

External Mask (E): Bit 7 controls whether the CPU is enabled for interruption by conditions included in the external class. Its meaning is the same as in the BC mode.

Protection Key: Bits 8-11 form the CPU protection key. The key is matched against a key in storage whenever information is stored, or whenever information is fetched from a location that is protected against fetching.

Extended-Control Mode: Bit 12 controls the format of the PSW and the mode of operation of the CPU. When the bit is zero, the PSW format and the CPU operation are as defined for the basic-control (BC) mode. When the bit is one, the extended-control (EC) mode is specified.

Machine-Check Mask (M): Bit 13 controls whether the CPU is enabled for interruption by machine-check conditions. Its meaning is the same as in the BC mode.

Wait State (W): When bit 14 is one, the CPU is in the wait state. When bit 14 is zero, the CPU is in the running 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. The meaning of these bits is the same as that of bits 36-39 of the BC PSW.

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

Bit positions 0, 2-4, 16-17, and 24-39 are unassigned. A specification exception is recognized when these bit positions do not contain zeros.

Exceptions Associated with the PSW

Exceptions associated with the 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.

Early Exception Recognition

For the following error conditions, a program interruption for specification exception occurs immediately after the PSW becomes active.

- A one is introduced into an unassigned bit position of the EC-mode PSW.
- The EC mode is specified (PSW bit 12 is one) in a CPU that does not have the EC facility installed.

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. If the EC facility is not installed, bits 0-15 and 34-63 of the invalid PSW are stored unchanged into the corresponding bit positions of the program old PSW, and the interruption code and instruction-length code are stored into bit positions 16-33 of the program old PSW.

When the execution of LOAD PSW or an interruption introduces a PSW with one of the above 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 error conditions is introduced by execution of SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the instruction-length code is set to 2, and the instruction address is 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 light remains on.

Late Exception Recognition

For the following conditions, the exception is recognized as part of the execution of the next instruction.

- An instruction address is introduced in which PSW bit 63 is one (specification exception).
- An access (addressing, protection, segment-translation, page-translation, or translation-specification) exception is associated with the location designated by the instruction address or the second or third halfword of the instruction starting at the designated address.

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.

For specification, addressing, protection, and translation-specification exceptions, the instruction-length code (ILC) stored upon the program interruption is 1, 2, or 3, indicating the number of halfword locations by which the instruction address has been updated. Whether the ILC is 1, 2, or 3 is unpredictable. For segment-translation and page-translation exceptions, the instruction address is not updated, and the ILC is 1, 2, or 3, the indication being unpredictable. In other respects, the current PSW, except for the interruption code and the ILC in the BC mode, is stored unmodified as the old PSW and contains the invalid value causing the interruption.

Programming Notes

The execution of LPSW, SSM, STNSM, and STOSM is suppressed on an addressing or protection exception, and hence the program old PSW provides information concerning the program causing the exception.

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.

| If the new PSW introduced by an interruption contains a format error, a series of interruptions occurs. See the section "Priority of Interruptions" in the chapter "Interruptions."

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 registers for control purposes. These registers are not part of addressable storage. The instruction LOAD CONTROL provides a means for loading control information from main storage into control registers, whereas STORE CONTROL permits information to be transferred from control registers to main storage. These instructions operate in a manner similar to LOAD MULTIPLE and STORE MULTIPLE.

One or more specific bit positions in control registers are assigned to each facility requiring such register space. When the facility and the associated register positions are installed, the bit performs the indicated control function, and STORE CONTROL returns the information placed in the register position by LOAD CONTROL or on reset. When STORE CONTROL is executed, the value corresponding to the unassigned register positions is unpredictable.

At the time the registers are loaded, the information is not checked for exceptions, such as invalid segment-size or page-size code or an address designating an unavailable or a protected location. The validity of the information is checked and the exceptions, if any, are indicated at the time the information is used.

Only the general structure of control registers is described here; a definition of the meaning of register positions appears with the description of the facility with which the register position is associated. A summary of control register allocation appears in the table "Assignment of Control Register Fields." This table shows the facility with which the field is associated and the initial value placed in the field upon execution of reset.

Programming Note

To ensure that presently written programs run if and when new facilities using additional control register positions are installed, only zeros should be loaded in unassigned control register positions. Similarly, although on some CPUs STORE CONTROL may provide zeros in the bit positions corresponding to the unassigned register positions, the program should not depend on such zeros being provided.

Word	Bits	Name of Field	Associated With	Initial Value
0	0	Block-Multiplexing Control	Block-Multiplexing	0
0	1	SSM-Suppression Control	SSM Suppression	0
0	2	TOD Clock Sync Control	Multiprocessing	0
0	8-9	Page-Size Control	Dynamic Addr. Translation	0
0	10	Unassigned, must be zero	Dynamic Addr. Translation	0
0	11-12	Segment-Size Control	Dynamic Addr. Translation	0
0	16	Malfunction-Alert Mask	Multiprocessing	0
0	17	Emergency-Signal Mask	Multiprocessing	0
0	18	External-Call Mask SIGP	Multiprocessing	0
0	19	TOD-Clock-Sync-Check Mask external	Multiprocessing	0
0	20	Clock-Comparator Mask int.	Clock Comparator	0
0	21	CPU-Timer Mask masks	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
1	0-7	Segment-Table Length	Dynamic Addr. Translation	0
1	8-25	Segment-Table Address	Dynamic Addr. Translation	0
2	0-31	Channel Masks	Channels	1
8	16-31	Monitor Masks	Monitoring	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[1] 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	Check-Stop Control	Machine-Check Handling	1
14	1	Synchronous-MCEL[2] Control	Machine-Check Handling	1
14	2	I/O-Extended-Logout Control	I/O Extended Logout	0
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
14	8	Asynchronous-MCEL Control	Machine-Check Handling	0
14	9	Asynchronous-Fixed-Log Control	Machine-Check Handling	0
15	8-28	MCEL Address	Machine-Check Handling	512[3]

Explanation:

The fields not listed are unassigned.

Except for bit 10 of control register 0, the initial value of unassigned register positions is unpredictable.

[1] PER means program-event recording.

[2] MCEL means machine-check extended logout.

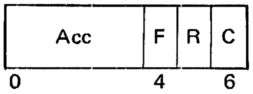
[3] Bit 22 is set to one, with all other bits set to zero, thus yielding a decimal byte address of 512.

Assignment of Control Register Fields

Key in Storage

For purposes of protection and recording of references and changes, main storage is divided into blocks of 2,048 bytes, each block having an address that is a multiple of 2,048. A control field, called "key in storage," is associated with each block of storage.

The key in storage has the following format:



The bit positions in the key are allocated as follows:

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

Fetch-Protection Bit (F): 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 protection key is permitted; a one indicates that protection applies to both fetching and storing. No distinction is made between the fetching of instructions and of operands.

Reference Bit (R): Bit 5 normally is set to one each time a location in the corresponding storage block is referred to either for storing or for fetching of information. This bit is associated with dynamic address translation.

Change Bit (C): Bit 6 is set to one each time information is stored into the corresponding storage block. This bit is associated with dynamic address translation.

The key in storage is not part of addressable storage. The program can explictly place information in all seven bits of the key by SET STORAGE KEY, and the contents of the key can be inspected by INSERT STORAGE KEY. Additionally, the instruction RESET REFERENCE BIT provides a means of inspecting the reference and change bits and of setting the reference bit to zero.

Protection

The protection facility is provided to protect the contents of main storage from destruction or misuse caused by erroneous or unauthorized storing or fetching by the program. It provides protection against improper storing or against both improper storing and fetching, but not against improper fetching alone.

Protection Action

When protection applies to a storage access, the key in storage is compared with the protection key associated with the request for storage access. A store is permitted only when the key in storage matches the protection key. The keys are said to match when the four high-order bits of the key in storage are equal to the protection key or when the protection key is zero. A fetch is permitted when the keys match or when bit 4 of the key in storage is zero. The protection action is summarized in the table "Summary of Protection Action."

Conditions		Is Access to Storage Permitted ?	
Bit 4 of Key in Storage	Key Relation	Fetch	Store
0	Match	Yes	Yes
0	Mismatch	Yes	No
1	Match	Yes	Yes
1	Mismatch	No	No

Explanation:

Match The four high-order bits of the key in storage are equal to the protection key, or the protection key is zero.

Yes Access is permitted.

No Access is not permitted; on fetching, the information is not made available to the program, and, 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 protection key of the current PSW is used as the comparand. The protection key of the CPU occupies bit positions 8-11 of the PSW. When the reference is made by a channel, and protection applies, the protection key associated with the I/O operation is used as the comparand. The protection key for an I/O operation is specified in bit positions 0-3 of the channel address word (CAW) and is recorded in bit positions 0-3 of the channel status word (CSW) stored as a result of an 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, a protection-check condition is indicated in the channel status word (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. On fetching, the protected information is not loaded into an addressable register, moved to another storage location, or provided to an I/O device.

The protection system 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 main-storage accesses to locations that are explicitly designated by the program and that are used by the CPU or channel to store or fetch information are subject to protection.

Protection is not applied to accesses that are implicitly made by the CPU or channel for such sequences as interruptions, updating the interval timer, logout, dynamic address translation, fetching the CAW during execution of an I/O instruction, storing the CSW by an I/O instruction or interruption, storing channel identification during execution of STORE CHANNEL ID, and the initial-program-loading and store-status functions. Similarly, protection does not apply to accesses initiated *via* the system console for entering or displaying information. However, when the program explicitly designates these locations, they are subject to protection.

Monitoring

The monitoring facility 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.

The monitoring facility 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.

The monitoring facility is available in both the BC and EC modes.

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

Control Register 8:

	Monitor Masks
0	16 31

The mask 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 placed at locations 148-149 and 156-159 of main storage. The format of the information placed at locations 148-149 and 156-159 is the same in BC and EC modes and is as follows:

Locations 148-149:

00000000	Monitor Class No.
0	8 15

Locations 156-159:

00000000	Monitor Code
0	8 31

The contents of bit positions 8-15 of MONITOR CALL are placed at location 149 and constitute the monitor class number. The address specified by the B_1 and D_1 fields of the instruction forms the monitor code, which is placed at locations 157-159. Zeros are placed 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 events:

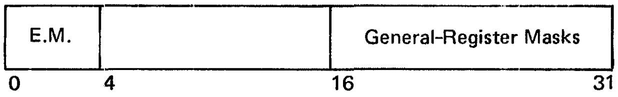
- Successful execution of a branch instruction.
- Alteration of the contents of designated general registers.
- Fetching of an instruction from designated main-storage locations.
- Alteration of the contents of designated main-storage locations.

The program has control over the conditions that are considered events for recording purposes and can specify selectively one or more events to be monitored. The information concerning a program event is provided to the program by means of a program interruption, with the cause of the interruption being identified in the interruption code.

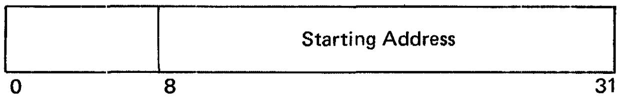
Control Register Allocation

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

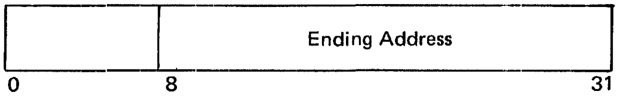
Control Register 9:



Control Register 10:



Control Register 11:



PER Event Masks: 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, are made to correspond one for one with the 16 registers, in the order of ascending addresses. When the bit is one, the register is included in monitoring 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 main-storage area.

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

Programming Note

Most models operate at reduced performance while monitoring for program events. In order to ensure that CPU performance is not degraded due to the operation of the program-event-recording facility, programs that do not utilize 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 and addresses in control registers 9-11 does not necessarily assure avoidance of performance degradation due to the use of the facility.

Operation

Program-event recording (PER) is available only in the EC mode and is under control of PSW bit 1, the PER mask; when the mask is zero, no program 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 an interruption. In BC mode the PER mask has, in effect, a value of zero, and program-event recording is disabled.

An interruption due to a program 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.

A program-event condition cannot be kept pending. When the CPU is disabled for a particular program event at the time it occurs, either by the mask in the PSW or by the masks in control register 9, the interruption condition is lost.

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. When the CPU is enabled for some program event and an instruction causes the CPU to be disabled for that particular event, the event causes an interruption 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 EC mode to BC mode, the PER interruption is taken with the old PSW specifying 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 in locations 150-155.

Program-event recording applies to all instructions, including the special-purpose instructions, such as those provided for emulation. The latter class of instructions indicates 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, a special-purpose 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

The cause of the interruption is identified by setting bit 8 of the interruption code to one and by the information placed in locations 150-155 of main storage. The interruption code on a program-event interruption may indicate concurrently a program event and another program-interruption condition. The format of the information stored in locations 150-155 is as follows:

Locations 150-151:

P.C.	000000000000
0	4 15

Locations 152-155:

00000000	PER Address
0	8 31

The event causing a program-event interruption is identified by a one in bit positions 0-3 of location 150, the PER code, 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. When a PER interruption occurs and more than one designated program event has been recognized, all recognized program events are concurrently indicated in the PER code.

The PER address at locations 153-155 identifies the location 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 the execution of an interruptible instruction is due to be interrupted by an I/O, external, or machine-check-recovery condition, the program-event interruption 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 a dynamic-address-translation (DAT) exception is encountered, the pending PER condition is indicated concurrently with the DAT condition. Normally a program event does not cause premature interruption of the interruptible instruction unless some other event is due to cause an asynchronous interruption. However, 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 on SUPERVISOR CALL, the PER interruption occurs immediately after the supervisor-call interruption.

Programming Notes

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.

1. 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. In all these cases, the old program PSW associated with the program-event interruption may indicate that the CPU was disabled for the interruption.
2. 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.

No instructions can both change the values of general-register alteration masks and cause a general-register alteration event to be recognized.

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.

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 program events--instruction fetching and storage alteration--involve the designation of an area in main 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.

When dynamic address translation is specified, the storage area is designated by logical addresses; when dynamic address translation is suppressed, control registers 10 and 11 contain real addresses.

The set of locations designated for monitoring purposes wraps around at location 16,777,215; that is, location 0 is considered to follow location 16,777,215. 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 the largest address in the system 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 main-storage alteration and instruction fetching is performed by carrying out the address comparison on all 24 bits of the addresses.

Program 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 special-purpose instructions, such as those provided for emulation, when the special-purpose instruction causes a branch. That is, the location of the next instruction executed by the CPU after leaving emulation mode does not immediately follow the location of the instruction which caused the CPU to enter the mode.

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

Instruction Fetching

Fetching the first byte of an instruction from the main-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 program event 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 program event is recognized when the first byte of the EXECUTE instruction or the subject instruction or both is located in the monitored area.

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

Storage Alteration

Storing of data by the CPU in the main-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 main storage are considered to have been altered whenever the CPU executes an instruction that causes the whole operand or part of it to be stored within the monitored area of main storage. Alteration is considered to take place whenever storing is considered to take place for purposes of indicating protection exceptions. (See "Recognition of Access Exceptions" in the chapter "Interruptions.") An arithmetic or movement operation is considered to fetch the operand, perform the indicated operation, if any, and then store the result. Such storing into main storage 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 timer updating, interruptions, execution of I/O instructions, and machine-check logout, including the interval timer, PSW, CSW, and logout 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 key storage is not considered part of main storage, and hence monitoring does not apply to alterations made by SET STORAGE KEY and RESET REFERENCE BIT.

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

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

The event is identified 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 into the register. Recognition of the event is not contingent on the new value being different from the previous one. A register-to-register format arithmetic or movement operation 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 stored into 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 character is used, when both operands have a 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 identified by setting bit 3 of the PER code to one.

Programming Notes

The following are some specifics concerning 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 BXH and BXLE 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 program events caused by an instruction that has caused also 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, or general-register alteration are not indicated for an operation or, in the case of the interruptible

instruction, for a unit of operation that is suppressed or nullified.

4. When the execution of the instruction is terminated, general-register and storage alteration is indicated whenever the event has occurred. Additionally, 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 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 exception being recognized.

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

Programming Notes

The execution of the interruptible instructions MOVE LONG (MVCL) and COMPARE LOGICAL LONG (CLCL) can cause events for general-register alteration and instruction fetch. 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:

1. The instruction-fetching event is indicated whenever the instruction is fetched for execution, regardless of whether it is the initial execution or resumption.
2. The general-register-alteration event is indicated on initial execution and on each resumption and does not depend on whether or not the register actually is changed.
3. 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. The event for address match on data storing for a single byte location can be recognized only once in the execution of MOVE LONG.

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:

1. 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.
2. If both conditions are met, delete instruction-fetching and register-alteration events.
3. 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.

		PER Event			
Exception	Type Of Ending	Branch	Instruction Fetch	Storage Alter	General Register Alteration
Operation	S	–	X[1]	–	–
Privileged Operation	S	–	X[1]	–	–
Execute	S	–	X[1]	–	–
Protection					
Instruction	S	–	–[1]	–	–
Operand	S or T	–[2]	X	X +	X +
Addressing					
DAT entry for instruction address	S	–	–[1]	–	–
Instruction	S	–	–[1]	–	–
DAT entry for operand address	S	–[2]	X	X[3]	X[3]
Operand	S or T	–[2]	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	–	–
Segment Translation					
Instruction address translation	N	–	–[1]	–	–
Operand address translation	N	–[2]	X	X[3]	X[3]
Page Translation					
Instruction address translation	N	–	–[1]	–	–
Operand address translation	N	–[2]	X	X[3]	X[3]
Translation Specification					
Instruction address translation	S	–	–[1]	–	–
Operand address translation	S	–[2]	X	X[3]	X[3]
Special Operation	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 along with the exception if the event has occurred; that is, the contents of the monitored storage location or general register were changed, or an attempt was made to execute an instruction whose first byte is located in the monitored area.

+ A model may indicate the event, but does not necessarily, if the event was called for (would have occurred had the operation been completed) but the event did not take place because the execution of the instruction was terminated.

– The event is not indicated.

[1] 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.

[2] This condition may occur for some special-purpose instructions, such as those provided for emulation.

[3] 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 when the exception causes the next unit of operation to be suppressed or nullified.

Indication of Program Events

Direct Control

The direct-control feature provides two instructions, READ DIRECT and WRITE DIRECT, and an external-signal facility, consisting of six external interruption lines. This feature operates independently of the facilities for performing I/O operations. The read and write instructions provide for the transfer of a single byte of information, normally for controlling or synchronizing purposes, between two cable-connected processing units or a cable-connected processing unit and external devices. Each of the six external lines, when pulsed, sets up the conditions for an external interruption.

Note: Some models provide the external-signal facility as a separate feature (without the READ DIRECT and WRITE DIRECT instructions).

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

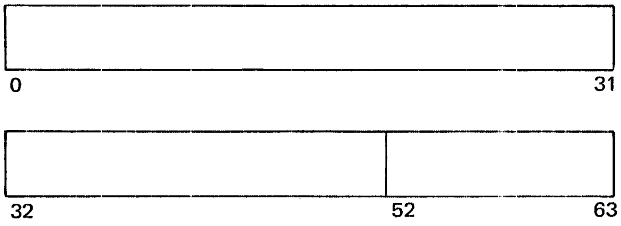
Time-of-Day Clock

The time-of-day clock provides a consistent measure of elapsed time suitable for the indication of date and time. The cycle of the clock is approximately 143 years.

In an installation with more than one CPU, depending on the model, each CPU may have a separate time-of-day clock, or more than one CPU may share a clock. In all cases, each CPU accesses a single clock.

Format

The time-of-day clock is a binary counter with a format as shown in the following illustration. The bit positions of the clock are numbered 0 to 63, corresponding to the bit positions of an unsigned fixed-point number of double precision. Time is measured by incrementing the value of the clock, following the rules for unsigned fixed-point arithmetic.



In the basic form, the clock is incremented by adding a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is incremented at such a frequency that the rate of advancement of the clock is the same as if a one were added in bit position 51 every microsecond. The resolution of the time-of-day clock is such that the incrementing rate is comparable to the instruction execution rate of the model.

When more than one time-of-day clock exists in a configured system, the stepping rates are synchronized such that all time-of-day clocks in the configuration are incremented at the exact same rate.

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

The operation of the clock is not affected or inhibited by any normal activity or event in the system. The clock runs when the CPU is in the wait or stopped state, or in the instruction-step, single-cycle, or test mode, and its operation is not affected by CPU, initial-CPU, program, initial-program, or system-clear resets or by the IPL procedure. Depending on the implementation, the clock may or may not run with the CPU power off.

States

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

The clock is in the not-operational state when its power is down or when it is disabled for maintenance. It depends on the model if the clock can be placed in this state.

When the power for the clock is turned on, the value of the clock is set to zero, and the clock enters the not-set state. With the clock in this state, STORE CLOCK causes condition code 1 to be set.

The clock enters the stopped state when SET CLOCK causes the clock's contents to be set, that is, when SET CLOCK is executed without encountering any exceptions and with the TOD-clock switch in the enable-set position. The clock can be placed in the stopped state from the set, not-set, and error states. The clock is not incremented while in the stopped state. When the clock is in the stopped state, STORE CLOCK causes the value of the stopped clock to be stored and condition code 3 to be set. This is distinguished from the not-operational state, where condition code 3 is set and a value of zero is stored.

The clock enters the set state only from the stopped state. This is under control of the time-of-day clock synchronization control bit, which is contained in control register 0, bit position 2. The initial value of this bit is zero. When the bit is zero or the clock-synchronization facility is not installed, the clock enters the set state at the completion of the SET CLOCK instruction. When the bit is one, the clock remains in the stopped state until either the bit is set to zero or until any other running time-of-day clock in the configured system is incremented to a value of all zeros in bit positions 32-63. Incrementing of the clock begins with the first stepping pulse after the clock enters the set state. If a clock is set to a value of all zeros in bit positions 32-63 and enters the set state as the result of a signal from another clock, bits 32-63 of the two clocks are in synchronism. The SET CLOCK instruction results in condition code 0 when the clock is set, regardless of whether the clock remains in the stopped state or enters the set state at the completion of the instruction.

The clock enters the error state when a malfunction is detected that is likely to have affected the validity of the clock's 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.

Setting and Inspection of Value

The clock can be inspected by means of the instruction STORE CLOCK which causes the current 64-bit clock value to be stored in main storage. The execution of STORE CLOCK is interlocked such that successive executions, either from the same CPU or from different CPUs, do not provide the same clock value if the clock is running. In multiprocessing configurations, this unique value may be obtained by storing additional bits of lower order than the resolution of the clock. These bits are not stored when the clock is in the stopped or not-operational state. With the exception of these bits, the clock provides only those bits which are incremented. Zeros are stored for the low-order bits not provided by the clock.

The clock can be set to a specific value by means of SET CLOCK, which causes the current clock value to be replaced by the operand designated by the instruction. The instruction SET CLOCK causes the value of the clock to be changed only when the TOD-clock switch on the system console is set to permit changing the value of the clock. In a multiprocessing system, the TOD-clock switch in each CPU which is configured to this CPU is ORed with the switch on this CPU. Thus, the operator can enable the setting of all clocks in the configuration by using the switch of any CPU in the configuration.

In a system where more than one CPU accesses the same clock, SET CLOCK is interlocked such that the entire contents appear to be updated at once. That is, if SET CLOCK instructions are issued simultaneously by two CPUs, the final result is either one or the other value. If SET CLOCK is issued on one CPU and STORE CLOCK on the other, the result is either the entire old value or the entire new value. When SET CLOCK is issued by one CPU, a STORE CLOCK issued on another CPU may find the clock in the stopped state even when the time-of-day clock synchronization control bit is zero. Since the clock enters the set state before incrementing, the first STORE CLOCK issued after the clock enters the set state may still find the original value introduced by SET CLOCK.

Programming Notes

Bit position 31 of the clock is incremented every 1.048576 seconds; hence for timing applications involving human responses, the high-order clock word may provide sufficient resolution.

To provide compatible operation from one system to another requires the establishing of a standard time origin, or epoch; that is, the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 A.M. Greenwich Mean Time is recommended as the standard epoch for the clock, although some early support of the TOD clock is not based on this epoch.

A program using the clock's value as a time-of-day and calendar indication may have to be aware of the support under which it is running. With the standard epoch, bit 0 of the TOD clock turns on May 11, 1971 at 11:56:53.685248 A.M. GMT. Normally a test of the high-order bit is sufficient to determine if the TOD clock value is the standard epoch: a one in this bit position indicates the standard epoch.

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

Because of the inaccuracies in setting the clock value on the basis of a synchronization signal provided by the operator, the low-order bit positions of the clock, expressing fractions of seconds, normally are not valid as indications of time of day. However, they permit elapsed time measurements of high resolution.

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.

In a multiprocessing system, each CPU has a separate clock comparator.

The clock comparator has the same format as the time-of-day clock. In the basic form, the clock comparator consists of bits 0-47, which are compared with the corresponding bits of the time-of-day clock. In some models, higher resolution is obtained by comparing more than 48 bits. 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 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 exists:

- 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, both comparands being considered binary unsigned quantities
- The clock comparator is installed and the time-of-day clock is in the error state or not operational

A request for a clock-comparator interruption does not remain pending when the value of the clock comparator is made equal to or larger 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.

Programming Note

The instruction STORE CLOCK may store a value which is larger 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. However, in this situation the interruption occurs at the completion of the execution of the instruction.

An interruption request for clock comparator persists as long as the clock comparator value is less than that of the TOD clock or as long as the TOD clock is not operational or in the error state. In view of this, after an external interruption for clock comparator has occurred, either the value of the clock comparator has to be replaced or the clock-comparator submask has to be set to zero before the CPU is again enabled for external interruptions. Otherwise, loops of external interruptions are formed.

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.

In a multiprocessing system, each CPU has a separate CPU timer.

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 by subtracting a one in bit position 51 every microsecond. In models having a higher or lower resolution, a different bit position is decremented at such a frequency that the rate of reduction of the CPU timer is the same as if a one were subtracted in bit position 51 every microsecond. The resolution of the CPU timer is such that the stepping rate is comparable to the instruction execution rate of the model.

The CPU timer causes an external interruption with the interruption code 1005 (hex). A request for a CPU-timer interruption exists 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 the decrementing of the CPU timer is not affected by concurrent I/O activity. However, in some models the CPU timer may stop during extreme I/O activity and other similar interference situations. In these cases, the time recorded by the CPU timer 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 rate switch on the system console is in the instruction-step position, the CPU timer is decremented only during the time in which 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, stopped, 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.

Programming Notes

The instruction STORE CPU TIMER may store a negative value even though the CPU is enabled for the interruption. This is because the timer value may be decremented one or more times between the instants when instruction execution is begun and when the CPU timer is accessed. However, in this situation the interruption occurs at the completion of the execution of the instruction.

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 that the old value is stored and a new value is set the CPU is disabled and the CPU-timer value goes from positive to negative.

The fact that CPU-timer interruptions are requested whenever the CPU timer is negative rather than just when the timer goes from positive to negative eliminates the requirement to test a value to ensure that it is positive before setting the CPU timer. A previously stored CPU-timer value could be negative if the CPU timer goes from positive to negative while external interruptions are disallowed between the time a non-CPU-timer interruption is taken and the CPU timer is stored.

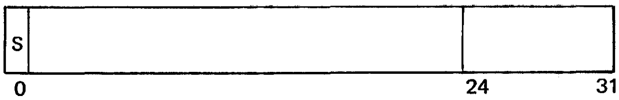
The persistence of the CPU timer interruption request means, however, that after an external interruption for CPU timer has occurred, either the value of the CPU timer has to be replaced or the CPU timer submask has to be set to zero before the CPU is again enabled for external interruptions. Otherwise, loops of external interruptions are formed.

The CPU timer in association with a program may be used as both a CPU-execution-time clock and a CPU interval timer.

The time measured for the execution of a sequence of instructions may depend on the effects of such factors as I/O interference, the remoteness of storage, and use of the cache, dynamic address translation, and instruction retry. Hence, repeated measurements of the same sequence on the same installation may differ.

Interval Timer

The interval timer occupies a 32-bit word at real main-storage location 80 and has the following format:



In a multiprocessing system, each CPU has an associated interval timer.

The interval timer is a binary counter that is treated as a signed integer by following the rules for fixed-point arithmetic. In the basic form, the contents of the 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 give counting at 300 cycles per second in bit position 23. The cycle of the 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 zero. Bits 9-15 are zero unless set to one for another condition that is concurrently indicated.

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

The timer is not necessarily synchronized with line frequency or the time-of-day clock, and its tolerance is not necessarily related to the tolerance of the line frequency or the clock.

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

The value of the timer is accessible by fetching the word at location 80 as an operand, provided the location is not protected against fetching. The 32-bit timer value may be changed at any time by storing a new value at location 80. When location 80 is protected, any attempt to change the value of the timer causes a program interruption for protection exception. When protection exception is indicated, the timer value remains unchanged.

The value of the timer may be changed without losing the real-time count by loading the new value in byte locations 84-87 and then shifting bytes 80-87 into byte locations 76-83 by means of the instruction MOVE (MVC), thus placing in a single operation the new timer value into word location 80 and making the old value available at location 76. The MVC instruction may designate locations 76-87 by real addresses 76-87 or by any logical addresses that translate to real addresses 76-87.

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

The timer value is not decremented when the CPU is not in the operating state, or when the rate switch on the system console is set to the instruction-step position.

Programming Notes

The interval timer, in association with a program, can serve both as a real-time clock and as an interval timer.

If any means other than the instruction MOVE (MVC) are used to interrogate and then replace the value of the timer, including MOVE LONG or two separate instructions, the program may lose a time increment if an updating cycle occurs between fetching and storing.

When the value of the interval timer is to be recorded on an I/O device, the program should first store the timer value in a temporary storage location to which the I/O operation subsequently refers. When the channel fetches the timer value directly from location 80, the value obtained is unpredictable.

Externally Initiated Functions

Resets

Two types of CPU-reset functions are provided: CPU reset and initial CPU reset. By combining the two CPU-reset functions with the I/O-system-reset function and clearing of storage, the following three system resets are provided: program reset, initial program reset, and system-clear reset. The table "Manual Initiation of System Resets" at the end of the description of resets summarizes how each type of system reset is manually initiated. Power-on reset is performed as part of powering on.

CPU reset provides a means of clearing equipment-check indications and the resultant unpredictability, if any, in the CPU state with the least amount of information destroyed. It is intended in particular for clearing check conditions when the system state is to be preserved for analysis or resumption of the operation.

Initial CPU reset performs the same functions as CPU reset but additionally initializes the contents of control fields. In particular, it initializes the prefix and control registers, which is normally necessary for initial program loading.

		Function Performed On[1]	
Key Activated	Position of Enable-System-Clear Key	CPU on Which Key Was Activated	Other CPUs Configured for Propagation of Manual Reset
System reset			
• Without store-status facility	Normal	Initial-program reset	*
• With store-status facility	Normal	Program reset	Program reset
System reset	Clear	System-clear reset	System-clear reset
Load	Normal	Initial-program reset, followed by IPL	Program reset
Load	Clear	System-clear reset, followed by IPL	System-clear reset

Explanation:

* This situation cannot occur, since the store-status facility is provided in a CPU equipped for multiprocessing.

[1] Activation of the system-reset or load key may change the configuration, including the connection with channels, storage units, and other CPUs.

Manual Initiation of System Resets

Program reset and initial program reset cause CPU reset and initial CPU reset, respectively, to be performed, and additionally cause I/O system reset to be performed.

System-clear reset causes initial program reset to be performed and, additionally, initializes or clears all registers and storage locations whose contents can be modified by a program. Such clearing is useful in debugging programs and to ensure user privacy.

Power-on reset initializes the contents of all control fields and either clears to zeros with valid checking-block code, or introduces valid checking-block code on, registers and storage locations that lose their contents when power is down. It eliminates the possibility of machine-check conditions due to random values introduced by powering on.

CPU Reset

CPU reset causes the following actions:

1. The execution of the current instruction or other processing sequence, such as interruption, is terminated, and all program and supervisor-call interruption conditions are cleared.
2. Pending external-interruption conditions are cleared.
3. Pending machine-check-interruption conditions and error indications are cleared.
4. The translation-lookaside buffer is cleared of entries.
5. Any buffers containing prefetched instructions or operands or results due to be stored are cleared of entries.
6. The CPU is placed in the stopped state after actions 1-5 have been completed.

See the table "Summary of Reset Action" for a detailed description of the effect of this reset on other parts of the system.

The CPU-reset function is performed as part of the three system resets and when the CPU accepts the CPU-reset order specified by a SIGNAL PROCESSOR instruction addressing this CPU. On some CPUs, model-dependent controls may be provided for initiating CPU reset.

Initial CPU Reset

Initial CPU reset causes CPU reset to be performed and additionally causes the following actions prior to placing the CPU in the stopped state:

1. The contents of the PSW, prefix, CPU timer, and clock comparator are set to zeros with valid checking-block code.
2. The contents of control registers are set to their initial values with valid checking-block code.

By setting the contents of the PSW to zero, the initial-CPU-reset function causes the PSW to assume the BC-mode format. The contents of the instruction-length-code and interruption-code fields remain unpredictable, as these values are not retained when a new PSW is introduced.

See the table "Summary of Reset Action" for a detailed description of the effect of this reset on other parts of the system.

The initial-CPU-reset function is performed as part of the initial-program and system-clear resets and when the CPU accepts the initial-CPU-reset order specified by a SIGNAL PROCESSOR instruction addressing this CPU. On some CPUs, model-dependent controls may be provided for initiating initial CPU reset.

I/O System Reset

I/O system reset causes the I/O-system-reset function to be performed in the channel (see the chapter "I/O Operations"). As part of this reset, 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 Systems Reference Library (SRL) or System Library (SL) publication. In general, a system reset resets only those functions in a shared control unit or device that are associated with the CPU signaling the reset.

The I/O-system-reset function is performed as part of the three system resets and normally cannot be initiated by itself.

Program Reset

Program reset causes CPU reset to be performed and causes I/O system reset to be performed in all channels configured to the CPU. See the table "Summary of Reset Action" for a detailed description of the effect of the reset on other parts of the system.

Execution of the program-reset function is initiated in a CPU by any of the following:

1. On a model that has the store-status facility installed, by activating the system-reset key on that CPU with the enable-system-clear key in the normal position.
2. By activating the following keys in any other configured CPU in a multiprocessing system:
   - The system-reset key with the enable-system-clear key in the normal position, or

- The load key with the enable-system-clear key in the normal position.

3. When the CPU accepts the program-reset order specified by a SIGNAL PROCESSOR instruction addressing this CPU.

Initial Program Reset

Initial program reset causes initial CPU reset to be performed and causes I/O system reset to be performed in all channels configured to the CPU. See the table "Summary of Reset Action" for a detailed description of the effect of the reset on other parts of the system.

	Reset Function					
Area Affected	CPU Reset	Program Reset	Initial CPU Reset	Initial Program Reset	System Clear Reset	Power On Reset
CPU state	S	S	S	S[1]	S[1]	S
Configured channels	N	R	N	R	R	R
PSW	U/V	U/V	C*	C*[1]	C*[1]	C*
Prefix	U/V	U/V	C	C	C	C
CPU timer	U/V	U/V	C	C	C	C
Clock comparator	U/V	U/V	C	C	C	C
Control registers	U/V	U/V	I	I	I	I
General registers	U/V	U/V	U/V	U/V	C/V	C/X
Floating-point registers	U/V	U/V	U/V	U/V	C/V	C/X
Keys in storage	U	U	U	U	C	C/X[3]
Volatile main storage	U	U	U	U	C	C/X[3]
Nonvolatile main storage	U	U	U	U	C	U
TOD clock	U[2]	U[2]	U[2]	U[2]	U[2]	C[3]

Explanation:

S CPU reset is performed. At the completion of this sequence, the CPU is in the stopped state.

N The state of the channel is not affected, and I/O-interruption conditions are not cleared, provided the CPU initially is in the stopped state.

When the reset function in the CPU is initiated at the time the CPU is executing an I/O instruction, is in the process of taking an I/O interruption, or is performing the initial-program-loading function, the communication between the CPU and the channel may be terminated, and the resultant state of the associated channel, subchannel, and I/O device is unpredictable. In this case, an I/O-interruption condition may appear to have been cleared, or an additional I/O-interruption condition may be generated.

R I/O system reset is performed in the configured channels, and pending 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 the channel.

U The contents, including the checking-block code, remain unchanged, provided the field is not being accessed at the time the reset function is performed. The subsequent contents of a field are unpredictable if it is accessed at the time of the reset.

U/V The contents remain unchanged, provided the field is not being accessed at the time the reset function is performed. However, on some models the checking-block code of the contents may be made valid. The subsequent contents of a field are unpredictable if it is accessed at the time of the reset.

C The contents are cleared to zero with valid checking-block code.

C/V The checking-block code of the contents is made valid. The contents normally are cleared to zeros but in some models may be left unchanged.

C/X The checking-block code of the contents is made valid. The contents normally are cleared to zeros but in some models may be left unpredictable.

I The contents are set to their initial values with valid checking-block code.

* Clearing the contents of the PSW to zero causes the CPU to assume the BC-mode format. The contents of the instruction-length-code and interruption-code fields remain unpredictable, as these values are not retained when a new PSW is introduced.

1 When the IPL sequence follows the reset function on that CPU, the CPU does not enter the stopped state, and the PSW is not necessarily cleared to zeros.

2 Access to the TOD clock by means of STORE CLOCK at the time a reset function is performed does not cause the value of the TOD clock to be affected.

3 When these units are separately powered, the action is performed only when the power for the unit is turned on.

Summary of Reset Action

Execution of the initial-program-reset function is initiated in a CPU by one of the following:

1. On a model that does *not* have the store-status facility installed, by activating the system-reset key on that CPU, with the enable-system-clear key in the normal position.
2. By activating the load key on that CPU, with the enable-system-clear key in the normal position. (The initial-program-reset function is immediately followed by the initial-program-loading operation.)
3. When the CPU accepts the initial-program-reset order specified by a SIGNAL PROCESSOR instruction addressing this CPU.

System-Clear Reset

System-clear reset causes initial CPU reset to be performed, causes I/O system reset to be performed in all channels configured to the CPU, and causes the contents to be set to zeros with valid checking-block code in that *part* of main storage and of keys in storage that is configured to the CPU. Additionally, the checking-block code of the contents of general registers and floating-point registers is made valid. In most models the contents of the registers are cleared to zeros, but in some the contents may be left unchanged except for making the checking-block code valid.

See the table "Summary of Reset Action" for a detailed description of the effect of the reset on other parts of the system.

Execution of the system-clear-reset function is initiated in a CPU by one of the following:

1. By activating the system-reset key on that CPU, with the enable-system-clear key in the clear position.
2. By activating the load key on that CPU, with the enable-system-clear key in the clear position. (The system-clear function is immediately followed by the initial-program-loading operation.)
3. By performing either of the above on any other configured CPU in a multiprocessing system.

Programming Notes

In order for the CPU-reset and initial-CPU-reset operations 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 interval timer, CPU timer, and clock comparator 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. In order to avoid the possibility of causing a CPU reset at the time the timing facilities are being updated or a machine-check interruption occurs, the CPU must be in the stopped state.

Resetting of the CPU does not affect the value and operation of the time-of-day clock.

System-clear reset causes all bit positions of the interval timer to be cleared to zeros.

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, in general, when CPU reset or initial CPU reset is executed in a CPU that is in the check-stop state, the contents of the PSW, addressable registers, and storage locations, including the keys, accessed at the time of the error are not reliable.

Power-On Reset

The power-on-reset function for a component of the system is performed as part of the power-on sequence for that component.

The power-on sequences for the TOD clock, main storage, and channels may be included as part of the CPU power-on sequence, or the power-on sequence for these units may be initiated separately. The following sections describe the power-on resets for the CPU, TOD clock, and main storage. See also "I/O Operations" and the appropriate Systems Reference Library (SRL) or System Library (SL) publication for channels, control units, and I/O devices.

CPU Power-On Reset: The power-on reset causes initial CPU reset to be performed and causes I/O system reset to be performed in all channels configured to the CPU. The checking-block code on the contents of general registers and floating-point registers is made valid. In most models the contents are cleared to zero, but in some models the contents may be left unpredictable except for the checking-block code.

TOD Clock Power-On Reset: The power-on reset causes the value of the time-of-day clock to be set to zero and causes the clock to enter the not-set state.

Main-Storage Power-On Reset: For volatile main storage (one that does not preserve its contents when power is down) and for keys in storage, power-on reset causes valid checking-block code to be placed in these fields. In most models the contents are cleared to zeros, but in some models the contents may be left unpredictable except for the checking-block code. The contents of nonvolatile main storage, including the checking-block code, remain unchanged.

Store Status

The store-status facility includes the following:

1. A change to the operation of the system-reset key when the enable-system-clear key is in the normal position. With the store-status facility installed, pressing the system-reset key causes a program reset; without this facility, initial program reset is performed.
2. An operator-initiated store-status function.

The store-status operation consists in placing the contents of the current PSW and the program-addressable registers in permanently assigned locations within the first 512 bytes of main storage. In the BC mode, the instruction-length code in the PSW is unpredictable, and an interruption code of zero is stored. The information provided for control register positions not associated with an installed facility is unpredictable. If the CPU timer, clock comparator, prefix register, or floating-point facility is not installed, the contents of the corresponding locations in main storage remain unchanged.

The word beginning at absolute storage address 268 is reserved for storing additional status as required by certain model-dependent features. If no feature requiring this field is installed, the contents of the field remain unchanged upon execution of the store-status function.

The following table lists the fields that are stored, their length, and their location in main storage.

Field	Length in Bytes	Absolute Address[1]
CPU timer	8	216
Clock comparator	8	224
Current PSW	8	256
Prefix	4	264
Model-dependent feature	4	268
F-P registers 0-6	32	352
General registers 0-15	64	384
Control registers 0-15	64	448

Explanation:

[1] Decimal address of the first byte of the field in absolute main storage.

Permanently Assigned Storage for Store Status

The contents of the registers are not changed. If an error is encountered during the operation, the CPU enters the check-stop state.

The store-status operation can be initiated by the operator on the system console. The operator controls and the procedure for initiating the function may differ among models and are described in the System Library (SL) publication for the model. In a multiprocessing system, the store-status operation can also be initiated at the addressed CPU by executing SIGNAL PROCESSOR, specifying the stop-and-store-status order.

Initial Program Loading

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

Initial program loading is initiated manually by selecting an input device with the load-unit-address switches and then pressing the load key. Pressing the load key causes a system-clear or an initial-program-reset operation to be performed on the CPU, as determined by the setting of the enable-system-clear key. Subsequently, a read operation is initiated from the selected input device.

The read operation is performed as if a START I/O instruction were executed that specified the device addressed by the load-unit-address switches and used a channel address word (CAW) containing a protection key of zero and a channel command word (CCW) address of 0. The address set up on the load-unit-address switches provides the 12 low-order bits of the I/O address; zeros are implied for the high-order address bits. Although the location of the first CCW to be executed is specified as 0, the first CCW actually executed is an implied CCW, containing, in effect, a read command with the modifier bits set to zero, a data address of 0, a byte count of 24, the chain-command flag on, the suppress-incorrect-length-indication flag on, the chain-data flag off, the skip flag off, and the program-controlled-interruption (PCI) flag off. The CCW fetched, as a result of command chaining, from location 8 or 16, as well as any subsequent CCW in the IPL sequence, is interpreted the same as a CCW in any I/O operation, with the exception that the PCI flag is 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 locations 0-7. When this PSW specifies the BC mode, the I/O address that was used for the IPL operation is stored at locations 2 and 3; when the EC mode is specified, the I/O address is stored at locations 186-187, and zeros are stored at location 185. The load indicator is turned off, and CPU operation proceeds under the control of the new PSW.

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, and the load indicator remains on. This occurs when the device designated by the address set up on the load-unit-address switches 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 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 locations 0-7 are unpredictable, but the contents of other main-storage locations remain unchanged. When less than eight bytes are read into the doubleword at location 0, the PSW fetched from location 0 at the conclusion of the IPL operation is unpredictable.

Programming Notes

The information read into 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 main 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 main-storage 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 read into locations 0-7 by a subsequent CCW.

The IPL I/O operation implicitly specifies the use of the first 24 bytes of main storage. Since the remainder of the IPL program may be placed in any part of storage, it is possible to preserve such areas of storage as the PSW and logout areas, which may be helpful in recovery.

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 manual indicator is off, and the wait indicator is on.

Dynamic Address Translation

Contents

Logical Storage Addressing . . . 58
Control . . . 58
PSW . . . 58
Control Register 0 . . . 58
Control Register 1 . . . 59
Translation Tables . . . 59
Segment-Table Entries . . . 59
Page-Table Entries . . . 60
Translation . . . 60
Types of Translation . . . 60
Translation Process . . . 60
Inspection of Control Register 0 . . . 61
Segment Table Lookup . . . 61
Page Table Lookup . . . 61
Formation of the Real Address . . . 62
Addresses Translated . . . 62
Interlocks Between Logical and Real Storage References . . . 63
Table Manipulation . . . 64
Translation-Lookaside Buffer . . . 64
States of Translation-Table Entries . . . 65
Use of the Translation-Lookaside Buffer . . . 65
Modification of Translation Tables . . . 66
Reference and Change Recording . . . 67
Address-Translation Exceptions . . . 68
Summary of Dynamic Address Translation Formats . . . 68

Dynamic address translation provides the ability to interrupt the execution of a program at an arbitrary moment, record it and its data on an external medium, such as a direct-access storage device, and at a later time return the program and the data to different main-storage locations for resumption of execution. The transfer of the program and its data between main and external storage may be performed piecemeal, and the return of the information to main storage may take place in response to an attempt by the CPU to access it at the time it is needed for execution. These functions may be performed without change or inspection of the program and its data, do not require any explicit programming convention for the relocated program, and do not disturb the execution of the program except for the time delay involved.

Address translation is achieved by treating the addresses supplied by and available to the program as *logical addresses*. These logical addresses are translated by means of translation tables to *real addresses* when storage is addressed. The translation occurs in blocks of addresses, called *pages*.

With appropriate support by an operating system, the dynamic-address-translation facility may be used to provide to a user a system wherein his main storage appears to be larger than the installed main storage. This apparent main storage is referred to as *virtual storage*, and the logical addresses used to designate locations in the virtual storage are referred to as *virtual addresses*. The virtual storage of a user may far exceed the size of the real main storage of the installation and normally is maintained on an external storage medium. Only the most recently referred-to pages of the virtual storage are assigned to occupy blocks of real main storage. As the user refers to pages of his virtual storage that do not appear in real main storage, they are brought in to replace pages in real main storage that are less likely to be needed. The swapping of pages of storage is performed by the operating system without the user's knowledge.

In the process of replacing blocks of main storage by new information from an external medium, it must be determined which block to replace and whether the block being replaced should be recorded and preserved on the external medium. To aid in this decision process, the key in storage is extended with a *reference* bit and a *change* bit.

Dynamic address translation may be specified for instruction and data addresses generated by the central processing unit (CPU), but is not available for the addressing of data and of control words in I/O operations. To facilitate I/O operations in a virtual-storage environment, the indirect-data-addressing facility is provided in the channel.

The address-translation facility requires that the CPU be equipped with the extended-control facility, as address translation is under control of bit 5 of the extended-control (EC) PSW.

The address-translation facility includes the instructions LOAD REAL ADDRESS, RESET REFERENCE BIT, and PURGE TLB. It makes use of control register 1 and bits 8-12 in control register 0.

Logical Storage Addressing

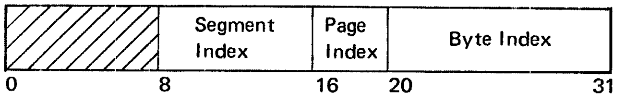
Address translation is achieved by treating the addresses supplied by the program as *logical addresses.* When the dynamic-address-translation facility is active, a logical address is translated during a storage reference into the corresponding *real address*, which designates a location in real storage. When the dynamic-address-translation facility is not installed or translation is not specified, a real address is identical to the corresponding logical address.

In the process of translation, two types of units of information are recognized--segments and pages. A *segment* is a block of sequential logical addresses spanning 65,536 (64K) or 1,048,576 (1M) bytes and beginning at an address that is a multiple of its size. The size of the segment is controlled by bits 11 and 12 of control register 0. A *page* is a block of contiguous storage containing 2,048 (2K) or 4,096 (4K) bytes and beginning at an address that is a multiple of its size. The size of the page is controlled by bits 8 and 9 of control register 0.

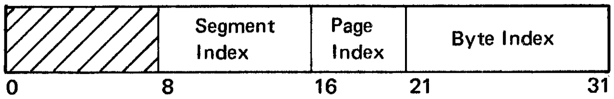
The logical address, accordingly, is divided into a segment-index field, a page-index field, and a byte-index field. The size of these fields depends on the segment and page size.

The segment index starts with bit 8 of the logical address and extends through bit 15 for a 64K-byte segment size and through bit 11 for a 1M-byte segment size. The page index starts with the bit following the segment index and extends through bit 19 for a 4K-byte page size and through bit 20 for a 2K-byte page size. The byte index comprises the remaining 11 or 12 low-order bits of the logical address. The formats of the logical address are as follows:

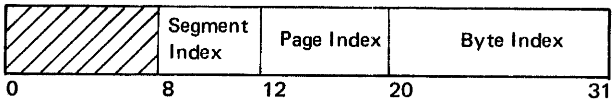
For 64K-byte segments and 4K-byte pages:



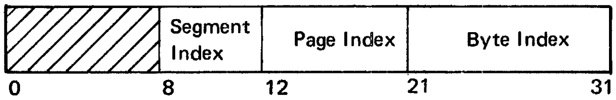
For 64K-byte segments and 2K-byte pages:



For 1M-byte segments and 4K-byte pages:



For 1M-byte segments and 2K-byte pages:



Logical addresses are translated into real addresses by means of two translation tables, a segment table and a page table, which reflect the current assignment of real storage. The assignment of real storage occurs in units of pages, the real locations being assigned contiguously within a page. The pages need not be adjacent in real storage even though assigned to a set of sequential logical addresses.

Control

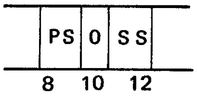
Address translation is controlled by the translation-mode bit in the PSW and by a set of bits in control registers 0 and 1. Additional controls are located in the translation tables.

PSW

When the dynamic-address-translation facility is installed, the CPU can operate either in the translation mode or without address translation. The mode of operation is controlled by bit 5 of the extended-control PSW, the translation-mode bit. When this bit is one, translation is specified; when this bit is zero, no implicit dynamic address translation takes place, and logical addresses are used as real addresses.

Control Register 0

Four bits are provided in control register 0 for the control of page size and segment size, as follows:



The bits are defined as follows:

Page Size (PS): Bits 8 and 9 of control register 0 control the size of pages, using the following code:

Bits 8 and 9 of Control Register 0	Page Size (Bytes)
01	2,048 (2K)
10	4,096 (4K)

When bit positions 8 and 9 contain a binary code other than 01 or 10, a translation-specification exception is recognized as part of the execution of an instruction using address translation, and the operation is suppressed. These bits are initialized to zeros.

Segment Size (SS): Bits 11 and 12 of control register 0 control the size of segments, using the following code:

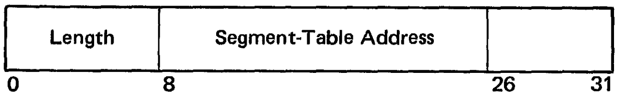
Bits 11 and 12 of Control Register 0	Segment Size (Bytes)
00	65,536 (64K)
10	1,048,576 (1M)

When bit position 12 contains a one, a translation-specification exception is recognized as part of the execution of an instruction using address translation, and the operation is suppressed. These bits are initialized to zeros.

Bit 10 of control register 0 must be zero when an instruction is executed that uses address translation; otherwise, a translation-specification exception is recognized as part of the execution of the instruction, and the operation is suppressed. The bit is not checked for zero when address translation is not installed.

Control Register 1

Bits 0-25 of control register 1 designate the beginning and length of the segment table:



The fields in the register are allocated as follows:

Segment-Table Length: Bits 0-7 of control register 1 designate the length of the segment table in units of 64 bytes, thus making the length of the segment table variable in multiples of 16 entries. The length of the segment table, in units of 64 bytes, is equal to one more than the value in bit positions 0-7. The contents of the length field are used to establish whether the entry designated by the segment-index portion of the logical address falls within the segment table.

Segment-Table Address: Bits 8-25 of control register 1, with six low-order zeros appended, form a 24-bit real address that designates the beginning of the segment table.

Programming Note

The validity of the information loaded into a control register, including that pertaining to dynamic address translation, is not checked at the time the register is loaded. This information is checked and the program exception, if any, is indicated at the time the information is used.

The information pertaining to dynamic address translation is considered to be used when an instruction is executed in the translation mode or when LOAD REAL ADDRESS is executed. The information is not considered to be used when the PSW specifies translation, but an I/O, external, restart, or machine-check interruption occurs before an instruction is executed, including the case when the PSW specifies the wait state.

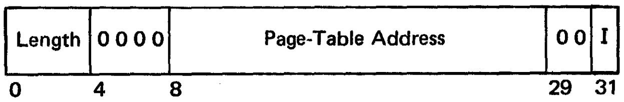
Translation Tables

Two types of translation tables are used for the translation process—a segment table and a page table. These tables reside in main storage.

Segment-Table Entries

The entry fetched from the segment table designates the length, availability, and origin of the corresponding page table.

An entry in the segment table has the following format:



The fields in the segment-table entry are allocated as follows:

Page-Table Length: Bits 0-3 designate the length of the page table in increments that are equal to a sixteenth of the maximum size of the table, the maximum size depending on the size of segments and pages. The length of the page table, in units one-sixteenth of the maximum size, is equal to one more than the value in bit positions 0-3. The length field is compared against the high-order four bits of the page-index portion of the logical address to determine whether the page index designates an entry within the page table.

Page-Table Address: Bits 8-28, with three low-order zeros appended, form a 24-bit real address that designates the beginning of the page table.

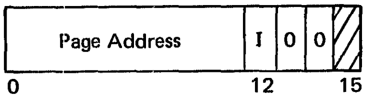
Segment-Invalid Bit: Bit 31 controls whether the segment associated with the segment-table entry is available. When bit position 31 contains a zero, address translation proceeds using the designated page table. When the bit is a one, a segment-translation exception is recognized, and the unit of operation is nullified.

The handling of bit positions 4-7 and 29-30 of the segment-table entry depends on the model. Normally a translation-specification exception is recognized and the unit of operation is suppressed when these bits are not zeros; however, on some models the contents of these bit positions may be ignored.

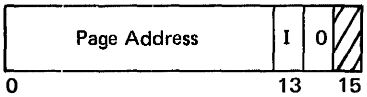
Page-Table Entries

The entry fetched from the page table indicates the availability of the page and contains the high-order bits of the real address. The format of the page-table entry depends on page size, as follows:

Page-table entry with 4K-byte pages:



Page-table entry with 2K-byte pages:



The fields in the page-table entry are allocated as follows:

Page Address: Bits 0-11 or bits 0-12, depending on the page size, provide the leftmost 12 or 13 bits of a 24-bit real storage address. When the page address and the contents of the byte-index field of the logical address are concatenated, with the page address forming the high-order part, the real storage address is obtained.

Page-Invalid Bit: Bit 12 or 13, depending on the page size, controls whether the page associated with the page-table entry is available. When the bit is zero, address translation proceeds using the table entry. When the bit is one, a page-translation exception is recognized, and the unit of operation is nullified.

Except for the rightmost bit position of the entry, the bit positions to the right of the page-invalid bit must contain zeros; otherwise, a translation-specification exception is recognized as part of the execution of an instruction using that entry for address translation, and the unit of operation is suppressed.

Programming Notes

A segment-table or page-table length code in excess of the maximum usable length code is valid. For example, the length code is valid even if the end of the table falls outside the available main storage or if part of the table is not addressable by the logical address.

The low-order bit position of a page-table entry is unassigned and is not checked for zero; thus, it is available for programming use.

Translation

Types of Translation

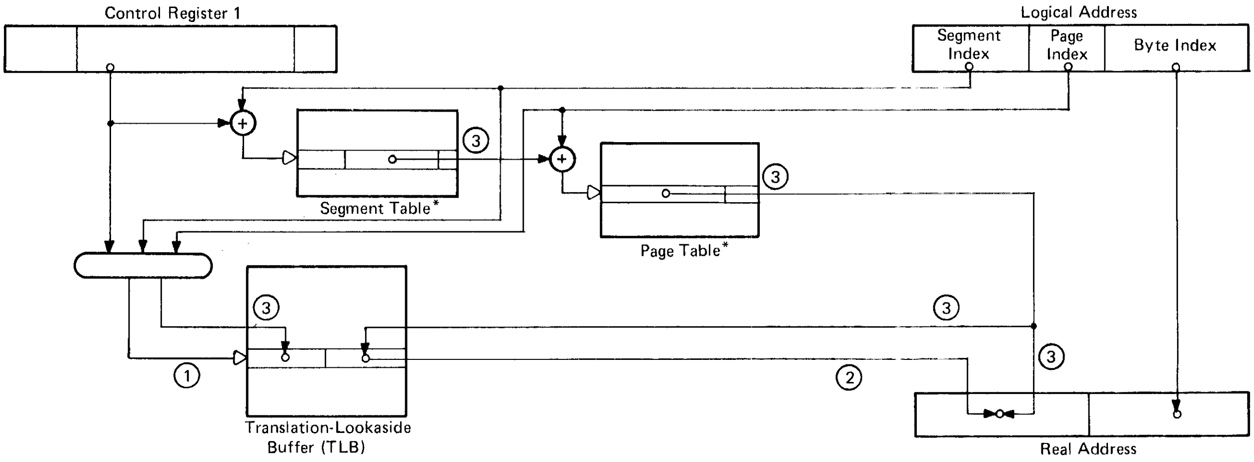
Two types of translation of main-storage addresses are distinguished--implicit and explicit. An explicit translation is one that is invoked for the translation of the operand address of LOAD REAL ADDRESS. The procedure invoked for the translation of all instruction addresses and of addresses of main-storage operands for all other instructions is referred to as implicit translation.

Translation Process

Translation is performed by means of a segment table and a page table, both of which reside in main storage. It is controlled by the translation-mode bit in the PSW and by a set of bits in control registers 0 and 1.

The segment-index portion of the logical address is used to select an entry from the segment table, the starting address and length of which are specified by the contents of control register 1. This entry designates the page table to be used. The page-index portion of the logical address is used to select an entry from the page table. This entry, the format of which depends on the size of the page, contains the high-order bits of the real address that corresponds to the logical address. The byte-index field of the logical address is used unchanged for the low-order bit positions of the real address.

In order to avoid the delay associated with references to translation tables in main storage, the information fetched from the tables normally is placed also in a special buffer, the *translation-lookaside buffer* (TLB), and subsequent translations involving the same table entries may be performed using the information recorded in the TLB.



* In Main Storage

① Information, which may include portions of the logical address and the segment-table address, is used to search the TLB.

② If match exists, address from TLB is used in forming the real address.

③ If no match exists, table entries in main storage are fetched to translate the address. Resulting value, in conjunction with search information, is used to form an entry in the TLB.

Translation Process

The translation process, including the effect of the TLB, is shown graphically in the figure "Translation Process."

Inspection of Control Register 0

The interpretation of the logical address for translation purposes is controlled by the segment size and page size, which are specified by the contents of bit positions 8-12 of control register 0. If bit positions 8-9 or 11-12 contain an invalid code or if bit 10 is one, a translation-specification exception is recognized, and the operation is suppressed.

Segment Table Lookup

The segment-index portion of the logical address is used to select a segment-table entry that designates the page table to be used in arriving at the real address. The address of the segment-table entry is obtained by appending six low-order zeros to the contents of bit positions 8-25 of control register 1 and adding the segment index to this value, with the low-order bit position of the segment index aligned with bit position 29 of the segment-table address.

As part of the segment-table lookup process, the segment index is compared against the segment-table length, bits 0-7 of control register 1, to establish whether the addressed entry is within the table. With 1M-byte segments, entries for all addressable segments are contained in a table of minimum length (length code of 0). With 64K-byte segments, four high-order zeros are appended to the contents of bit positions 8-11 of the logical address, and this extended value is compared against the eight-bit segment-table length. If the value in the segment-table-length field is less than the value in the corresponding bit positions of the logical address, a segment-translation exception is recognized, and the unit of operation is nullified.

If the storage address generated for fetching the segment-table entry refers to a location outside the main storage of the installed system, an addressing exception is recognized, and the unit of operation is suppressed.

Bit 31 of the entry fetched from the segment table specifies whether the corresponding segment is available. This bit is inspected, and, if it is one, a segment-translation exception is recognized, with the unit of operation nullified. Handling of bit positions 4-7 and 29-30 of the segment-table entry depends on the model: normally a translation-specification exception is indicated and the unit of operation is suppressed when they do not contain zeros; however, on some models they may be ignored.

When no exceptions are recognized in the process of segment-table lookup, the entry fetched from the segment table designates the length and beginning of the corresponding page table.

Page Table Lookup

The page-index portion of the logical address, in conjunction with the page-table address derived from the segment-table entry, is used to select an entry from the page table. The page-table-entry address is obtained by appending three low-order zeros to the contents of bit positions 8-28 of the segment-table entry and adding the page index to this value. The addition is performed with the low-order bit of the page-index aligned with bit 30 of the page-table address.

As part of the page-table lookup process, the four high-order bits of the page index are compared against the page-table length, bits 0-3 of the segment-table entry, to establish whether the addressed entry is within the table. If the value in the page-table-length field is less than the value in the four high-order bit positions of the page-index field, a page-translation exception is recognized, and the unit of operation is nullified.

If the storage address generated for fetching the page-table entry refers to a location outside the main storage of the installed system, an addressing exception is recognized, and the unit of operation is suppressed.

The entry fetched from the page table indicates the availability of the page and contains the high-order bits of the real address. The page-invalid bit is inspected to establish whether the corresponding page is available. If this bit is one, a page-translation exception is recognized, and the unit of operation is nullified. If bit positions 13-14 for 4K-byte pages or bit position 14 for 2K-byte pages contains one, a translation-specification exception is recognized, and the unit of operation is suppressed.

Formation of the Real Address

When no exceptions in the translation process are encountered, the real page address obtained from the page-table entry and the byte-index portion of the logical address are concatenated, with the page address forming the high-order part. The result forms the real storage address.

Whenever access to main storage is made during the address translation process for the purpose of fetching an entry from a segment or page table, storage protection is ignored; that is, the reference is made as if the storage location containing the translation-table entry were not protected against fetching.

Programming Note

When more than one exception is encountered in the process of address translation, only the exception with the highest priority is indicated with the program interruption. The priority in which exceptions are recognized is listed in the table "Priorities of Access Exceptions" in the chapter "Interruptions."

Addresses Translated

All main storage addresses that are explicitly specified by the program and are used by the CPU to refer to main storage for an instruction or an operand are logical addresses and are subject to dynamic address translation. Analogously, the corresponding addresses indicated to the program on an interruption or as the result of executing an instruction are logical, as are the addresses in control registers 10 and 11 designating the starting and ending locations for program-event recording (PER).

Translation is not applied to addresses explicitly designating keys in storage (operand addresses in SET STORAGE KEY, INSERT STORAGE KEY, and RESET REFERENCE BIT) and to quantities that are formed as storage addresses from the values designated in the B and D fields of an instruction but that are not used to address main storage. The latter include operand addresses in LOAD ADDRESS, MONITOR CALL, and the shifting and I/O instructions. Similarly, translation is not applied to the addresses implicitly used by the CPU or channel for such sequences as interruptions, updating the interval timer at location 80, address translation, and logout, including the machine-check-extended-logout address in control register 15. However, when the program explicitly designates these locations as the source of an operand or instruction, the addresses are subject to translation.

Dynamic address translation is not applied to the addresses used by channels to transfer data, channel-command words, or indirect-data-address words. Similarly, dynamic address translation is not applied to the I/O-extended-logout address at location 172.

The handling of storage addresses associated with DIAGNOSE is model-dependent.

The processing of addresses, including dynamic address translation and prefixing, is summarized in the charts "Types of Addresses" and "Handling of

Absolute, real, and logical addresses are distinguished on the basis of the transformations that are applied to the address during a storage access.

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

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

When a logical address is used for an access to main storage, it is translated, by means of dynamic address translation, to a real address and subsequently is converted, by means of prefixing, to an absolute address.

Types of Addresses

Addresses." Prefixing, when provided, is applied after the address has been translated by means of the dynamic-address-translation facility. For a description of prefixing, see "Prefixing" in the chapter "Multiprocessing."

Interlocks Between Logical and Real Storage References

When dynamic address translation is not invoked, the results stored by one instruction appear to that CPU to be completed before execution of the next instruction, including the instruction fetch, is begun. When an instruction has two main-storage operands, the handling of overlapped main-storage operands is included as part of the instruction definition.

When dynamic address translation is invoked and translation tables are constructed such that a location in real storage is designated by one and only one logical address, overlapping operands and changes to subsequent instructions are handled in the same way as when the references are made by real addresses.

With dynamic address translation, a location in real main storage may have multiple logical addresses. That is, the translation tables may be set up in such a way that more than one logical page address (segment-index and page-index portion of logical address) is translated to the same real page address. Only when the tables are set up in this way and when more than one logical page address is used to refer to a real location as a source of an instruction or operand, the following exceptions to the normal interlocks occur:

1. When an instruction changes the contents of a storage location from which a subsequent instruction has been prefetched and when different logical page addresses are used to designate that location for storing the result and fetching the instruction, the use of the common real location is not necessarily recognized. An instruction may be prefetched using a translated logical address only when the associated dynamic-address-translation table entries are attached and valid. Instructions which are prefetched may be interpreted for execution only for the same logical address for which the instruction was prefetched. All copies of prefetched instructions are discarded when the CPU enters or leaves translation mode, when changes are made to the translation parameters in control registers 0 and 1 while the CPU is in the translation mode, when a serializing operation is performed, and when the CPU enters the operating state.
2. When both operands in a unit of operation include the same real storage location (the operands overlap in real storage) and the common location is designated in the two operands by

Logical Addresses Explicitly Designated by the Program:

- Instruction address in PSW
- Branch addresses
- Addresses of operands in main storage
- Operand address in LOAD REAL ADDRESS
- PER starting address in control register 10 and PER ending address in control register 11

Real Addresses Explicitly Designated by the Program:

- Operand addresses in SET STORAGE KEY, INSERT STORAGE KEY, and RESET REFERENCE BIT
- MCEL address in control register 15
- Segment-table address in control register 1
- Page-table address in segment-table entry
- Page address in page-table entry

Absolute Addresses Explicitly Designated by the Program:

- Prefix value
- CCW address in CAW
- Data address in CCW
- CCW address in a CCW specifying transfer in channel
- Data address in indirect-data-address words
- IOEL address at real location 172

Addresses Not Used to Address Storage:

- Operand addresses specifying the amount of shift in the shift insrtuctions
- Operand address in LOAD ADDRESS
- Operand address in MONITOR CALL
- Second-operand address in SIGNAL PROCESSOR
- I/O addresses in I/O instructions

Real Addresses Used Implicitly:

- Addresses of PSWs used during interruption
- Address used by CPU to update interval timer at real location 80
- Address of CAW, CSW, and other locations used during an I/O interruption or during execution of an I/O instruction, including STORE CHANNEL ID

Absolute Addresses Used Implicitly:

- Addresses used for the store-status function

Logical Addresses Provided to the Program:

- Address stored in instruction-address field of old PSW on interruption
- Address stored by BRANCH AND LINK
- Address stored in register 1 by TRANSLATE AND TEST and EDIT AND MARK
- Address stored at real location 144 on a program interruption for page-translation or segment-translation exception
- Address stored at real location 152 on a program interruption for PER

Real Addresses Provided to the Program:

- The translated address generated by LOAD REAL ADDRESS
- Address of segment-table entry or page-table entry provided by LOAD REAL ADDRESS

Absolute Addresses Provided to the Program:

- Failing-storage address at real location 248
- CCW address in CSW

Handling of Addresses

different logical page addresses; the use of the common real location is not necessarily recognized.

When the use of a common real storage location is not recognized, storing into the location does not necessarily appear to be completed by the time the instruction or operand is fetched from the location. In the case of unrecognized operand overlap, the portion of the instruction definition pertaining to overlap does not necessarily apply.

Any change to the key in storage appears to be completed before the following reference to the associated storage block is made, regardless of whether the reference to the storage location is made by a logical or real address. Analogously, any prior references to the storage block appear completed when the key for that block is changed or inspected.

Since the interlocks discussed in this section pertain to references made by the same CPU, a common real location implies also a common absolute location. This is true because, for any one CPU, a one-to-one correspondence exists between real and absolute addresses, and a change in the prefix value, changing this mapping, causes serialization.

The interlocks between storage references are summarized in the table "Summary of Interlocks Between Storage References."

Table Manipulation

Translation-Lookaside Buffer

To enhance performance, the dynamic-address-translation mechanism normally is implemented such that some of the information specified in the segment and page tables is maintained in a special buffer, referred to as the translation-lookaside buffer (TLB). The CPU necessarily refers to a table entry in main storage only for the initial access to that entry. This information subsequently may be maintained in the TLB, and all subsequent translations involving translation-table entries from the same real storage location may be performed using the information recorded in the TLB. The presence of the TLB affects the translation process to the extent that a modification of the contents of a table entry in main storage does not necessarily have an immediate effect, if any, on the translation.

The size and the structure of the TLB depends on the model. For instance, the TLB may be implemented such as to contain only a few entries pertaining to the currently designated segment table, each entry consisting of the high-order portions of a logical address and its corresponding real address; or it may contain arrays of values where the real page address is selected on the basis of the current segment-table starting address, page-size designation, segment-size designation, and the high-order bits of the logical address. Entries within the TLB are not explicitly addressable by the program.

The following sections describe the conditions under which information may be placed in the TLB and information from the TLB may be used for address translation, and describe how changes to the translation tables affect the translation process. Information is not necessarily retained in the TLB under all conditions for which such retention is permissible. Furthermore, information in the TLB may

Interlocks between two references by a single CPU to a location in real storage when the same real location is designated by different addresses.

	Is it necessarily recognized that reference is made to the same real location ?		
Addresses used to designate a location in real storage	References within Same Instruction	References by Two Instructions	
	Operand-Operand	Operand-Operand	Operand-Instruction
Real X and Real X	Yes	Yes	Yes
Real X and Logical A	–	Yes	Yes
Logical A and Logical A	Yes	Yes	Yes
Logical A and Logical B	No	Yes	No*

Explanation:

– Not applicable.

* Reference to the same real location is recognized when a serialization function occurs between the two references.

Real X A real address designating location X in real storage.

Logical A A logical address A designating location X in real storage.

Logical B A logical address B designating location X in real storage.

Summary of Interlocks Between Storage References be purged under conditions additional to those for which purging is mandatory.

States of Translation-Table Entries

The effects of any manipulation of the contents of a table entry by the program and the recording of its contents in the TLB depend on whether the entry is valid, on whether the entry is attached, and on whether the entry is active.

The *valid* state denotes that the segment or page associated with the table entry is available. An entry is valid when the segment-invalid bit or page-invalid bit in the entry is zero. A segment-translation or page-translation exception is recognized when an attempt is made to use an invalid table entry for translation.

The *attached* state denotes that the CPU can attempt to use the table entry for implicit address translation and hence depends on the state of the CPU as specified by the PSW, control register 1, and bit positions 8-12 of control register 0.

A segment-table entry is attached to a CPU when all of the following three conditions are met:

- The current PSW specifies the translation mode.
- The entry is within the segment table designated by control register 1.
- With the segment size currently specified in control register 0, the entry can be designated by a logical address.

The PSW is considered to specify the translation mode when bit 5 is one and the EC mode is specified, regardless of whether the contents of any other PSW fields are due to cause an exception to be recognized.

A page-table entry is attached to a CPU when it is within the page table designated by the page-table address and page-table length either in an attached and valid segment-table entry or in a TLB copy of an attached segment-table entry and by the page-size specification in control register 0.

The *active* state denotes that the table entry may remain recorded in the TLB.

A table entry becomes active when it is made both valid and attached or after the TLB is purged with the table entry both valid and attached. A table entry ceases being active when the TLB is purged. Although all entries become inactive during a purge of the TLB, entries that are both valid and attached become active at the completion of the purge.

Programming Notes

The segment size controls how many segment-table entries can be referred to for translation. Both the page size and segment size control selection of page-table entries and hence may affect whether or not an entry is attached.

Although a table entry becomes active when it is made both valid and attached, it need not remain valid and attached to remain active. For example, an attached table entry remains active when the I bit is set to one, and a valid table entry remains active when it is made unattached.

Use of the Translation-Lookaside Buffer

A segment-table entry or a page-table entry may be placed in the TLB only when the entry is attached and the I bit in the entry is zero. The entry may be placed in the TLB as soon as it becomes attached and valid.

Information from a TLB copy of a segment-table entry may be used for implicit address translation only when the TLB entry was formed using information that was fetched from storage as an attached and valid segment-table entry, only when that real storage location is selected as a segment-table entry during the translation process, and only when the table entry is attached at the time of the selection.

Information from a TLB copy of a page-table entry may be used for implicit address translation only when the TLB entry was formed using information that was fetched from storage as an attached and valid page-table entry, only when that real storage location is selected as a page-table entry during the translation process and the table entry is attached at the time of selection, and only when the page size at the time of forming the TLB copy was the same as the current page size.

The operand address of LOAD REAL ADDRESS is translated without the use of the TLB contents. Translation in this case is performed by the use of the designated tables in main storage.

All information in the TLB is necessarily cleared only by execution of PURGE TLB, SET PREFIX, or CPU reset.

Programming Notes

No entries can be placed in the TLB in the BC mode or when translation is not specified, because the table entries at this time are not attached. In particular, translation of the operand address of LOAD REAL ADDRESS, with translation suppressed, does not cause entries to be placed in the TLB.

Conversely, when translation is specified, information may be loaded into the TLB from all translation-table entries that could be used for address translation, given the current designation of page size, segment size, segment-table address, and segment-table length. The loading of the TLB does not depend on whether the entry is used for translation as part of the execution of the current instruction, and such loading can occur when the wait state is specified. Similarly, information from a segment-table or page-table entry having a format error may be recorded in the TLB.

More than one copy of a table entry may exist in the TLB. For example, some implementations may cause a copy of a valid table entry to be placed in the TLB for each segment-table address by which the entry becomes attached, and some implementations may cause a valid page-table entry to be placed in the TLB for each attached and valid segment-table entry by which the page-table entry is designated.

Modification of Translation Tables

When an inactive, attached, and invalid table entry is made valid, the change takes effect immediately. Similarly, when an inactive, unattached, and valid page-table entry is made attached by making an inactive, attached, and invalid segment-table entry valid, the change takes effect immediately.

However, since the exceptions associated with dynamic address translation may be established by a pretest for operand accessibility that is performed as part of the initiation of the execution of the instruction, a segment-translation or page-translation exception may be indicated on the basis of the state of the table entry at the start of the execution of an instruction. Consequently, a segment-translation or page-translation exception may be indicated when a table entry is invalid at the start of execution even if the instruction would have validated the table entry it uses and the table entry would have appeared valid if the instruction were considered to process the operands one byte at a time. See the section "Recognition of Access Exceptions" in the chapter "Interruptions" for the recognition of dynamic-address-translation exceptions for the interruptible instructions.

A change to an active table entry may take effect for implicit translation any time from the instant of the change through the completion of the following purging of the TLB.

When an active table entry is changed, either to another value suitable for translation, or to a value that prohibits its use for translation, any subsequent attempt to use the entry for implicit address translation before the TLB is purged may yield unpredictable results. The use of the new value may begin between instructions or during execution of an instruction, including the instruction that caused the change. Moreover, until the TLB is purged, the TLB may contain both the old and the new values, and it is unpredictable whether the old or new value is selected for a particular access. If the use of the new value of the entry causes an exception, the exception may or may not cause an interruption to occur, and if an interruption does occur, the instruction execution may be terminated even though the exception would normally cause suppression or nullification.

Manipulation of attached table entries may cause spurious table-entry values to be recorded in a TLB. For example, if changes are made piecemeal, modification of a valid attached entry may cause a partially updated entry to be recorded, and, if an intermediate value is introduced in the process of the change, a supposedly invalid entry may temporarily appear valid and may be recorded in the TLB. Such an intermediate value may be introduced if the change is made by an I/O operation that is retried, or if an intermediate value is introduced during the execution of a single instruction.

When LOAD CONTROL changes the segment size, page size, segment-table address, or segment-table length, the values of these fields at the start of the operation are in effect for the duration of the operation.

The relation between the states of table entries and their use is summarized in the table "Use of Translation Tables."

Programming Notes

When an instruction, such as MOVE (MVC), changes an attached table entry, including a change that makes the entry invalid, and subsequently uses the entry for implicit translation, a changed entry is being used without a prior purging of the TLB, and the associated unpredictability of result values and of exception indication applies.

All modifications to translation tables by the program should consider the effect of the TLB on the use of the tables in main storage and the possible effects of intermediate result values and of piecemeal changes. The following rules are recommended for changing translation tables. If these rules are observed, translation is performed as if the table entries from main storage were always used for the translation process.

1. An entry must not be changed, other than changing the low-order bit of a page-table entry, while it is being used by any CPU.
2. When any change is made to a table entry, other than a change to the low-order bit of a page-table entry, each CPU in which the entry is active must issue PURGE TLB after the change occurs and prior to the use of the entry for implicit address translation by that CPU.

State of Table Entry			Can Copy of Table Entry Be in TLB ?	Can Table Entry Be Fetched for Translation ?	Can Table Entry Be Used for Translation ?	Can TLB Copy Be Fetched for Implicit Translation ?
Active	Attached	Valid	Yes	Yes	Yes	Yes
Active	Attached	Invalid	Yes[1]	Yes	No	Yes
Active	Unattached	Valid	Yes[1]	No	No	No
Active	Unattached	Invalid	Yes[1]	No	No	No
Inactive	Attached	Valid	*	*	*	*
Inactive	Attached	Invalid	No	Yes	No	No
Inactive	Unattached	Valid	No	No	No	No
Inactive	Unattached	Invalid	No	No	No	No

Explanation:

[1] The TLB may contain a copy of a previously attached and valid entry.

* This state cannot exist. An attached and valid table entry is active.

Use of Translation Tables

3. When any change is made to an invalid entry in such a way as to cause intermediate valid values to appear in the entry, each CPU to which the entry is attached must issue PURGE TLB after the change occurs and prior to the use of the entry for implicit address translation by that CPU.

Note that when an invalid page-table entry is made valid without introducing intermediate valid values, the TLB need not be purged in a CPU in which the entry previously was inactive. Similarly, when an invalid segment-table entry is made valid without introducing intermediate valid values, the TLB need not be purged in a CPU in which the segment-table entry and all page-table entries attached by it previously were inactive.

Execution of the PURGE TLB instruction may have an adverse effect on the performance of some models. Use of this instruction should, therefore, be minimized in conformity with the above rules.

Reference and Change Recording

Reference recording provides information for use in selecting storage blocks for page replacement. Change recording provides information as to which pages have to be saved when they are replaced by new pages. Both reference and change recording are associated with the dynamic-address-translation facility.

When the dynamic-address-translation facility is installed, the key in storage is extended with two additional bits. Bit 5, the *reference bit*, normally is set to one each time a location in the corresponding storage block is referred to either for storing or fetching of information. Bit 6, the *change bit*, is set to one each time information is stored in the corresponding storage block. The recording of references and changes is not contingent on whether the CPU is in the extended-control or basic-control mode or whether address translation is specified.

Reference and change recording takes place for any main-storage access and applies to accesses made by a CPU, as well as to those due to I/O operations. Hence, references to a main-storage location associated with interruptions and I/O instructions, such as occur to the CAW, CSW, or PSW locations, are included. A translation-table lookup in the process of address translation is considered a reference, provided the table in main storage has actually been referred to. It is unpredictable whether updating of the interval timer causes change and reference bits for location 80 to be turned on. References to the operand locations of SET STORAGE KEY, INSERT STORAGE KEY, and RESET REFERENCE BIT do not cause reference or change to be recorded.

The change bit is not turned on 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, addressing, segment-translation, or page-translation exception, or an I/O reference to an invalid or protected location does not cause the change bit to be turned on.

The record of references provided by the reference bit is substantially accurate. The reference bit may be turned on 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 turned on. Under certain unusual conditions, a reference bit that is on may be turned off by other than explicit program action.

Reference and change recording operates on 2,048-byte blocks regardless of the page size invoked. With a 4,096-byte page size, two keys are associated with a page.

Programming Note

The accuracy of reference recording is such as to allow for effective operation of paging algorithms.

The reference bit cannot be used to establish the usage of pages containing translation tables since, after the initial reference, the tables may be used by means of references to the translation-lookaside buffer, without any fetching of the table entries from main storage.

Address-Translation Exceptions

When the dynamic-address-translation facility is installed, three additional program-exception conditions are introduced: segment-translation exception, page-translation exception, and translation-specification exception. The CPU cannot be disabled for the translation exceptions.

The presence of the dynamic-address-translation facility also introduces new conditions that are recognized as addressing exceptions. When address translation is invoked, an addressing exception is recognized when an attempt is made to use a segment-table entry or a page-table entry that is designated at a location outside the available main storage of the installed system. The unit of operation is suppressed.

The handling of all exceptions associated with dynamic address translation is summarized in the table "Handling of Access Exceptions" in the chapter "Interruptions."

Summary of Dynamic Address Translation Formats

The first table summarizes the possible combinations of the page-address and byte-index fields in the formation of a real storage address.

The eight-bit length field in control register 1 provides for a maximum length code of 255 and permits designating a segment table of 16,384 bytes, or 4,096 entries, which is more than can be referred to for translation purposes by the logical address. With 1M-byte segments, only 16 segments can be addressed, requiring a segment table of 64 bytes. A table of 64 bytes is specified by a length code of 0 and is the smallest table that can be specified. With 64K-byte segments, up to 256 segments can be addressed, requiring at the most a segment table of 1,024 bytes and a length code of 15. These relations are summarized in the second table.

The third table lists the maximum sizes of the page table and the increments in which the size of the page table can be controlled.

	Real Storage Address			
	Page Address		Byte Index	
Size of Page (Bytes)	Bit Positions in Page-Table Entry	No. of Bits	Bit Positions in Logical Address	No. of Bits
2K	0-12	13	21-31	11
4K	0-11	12	20-31	12

			Maximum Segment Table		
Size of Segment (Bytes)	Segment Index Field Size (Bits)	Number of Addressable Segments	Size (Bytes)	Usable Length Code	Table Increment (Bytes)
64K	8	256	1,024	15	64
1M	4	16	64	0	64

Size of				Maximum Page Table		
Segment (Bytes)	Page (Bytes)	Page Index Field Size (Bits)	Number of Pages in Segment	Size (Bytes)	Usable Length Code	Table Increment (Bytes)
64K	2K	5	32	64	15	4
64K	4K	4	16	32	15	2
1M	2K	9	512	1,024	15	64
1M	4K	8	256	512	15	32

Summary of DAT Formats

Interruptions

Contents

Interruption Action . . . 70
Source Identification . . . 70
Enabling and Disabling . . . 70
Instruction-Length Code . . . 71
Zero ILC . . . 71
ILC on Instruction Fetch Exceptions . . . 73
Point of Interruption . . . 74
Instruction Execution . . . 74
Types of Ending . . . 74
Execution of Interruptible Instructions . . . 74
Machine-Check Interruption . . . 75
Program Interruption . . . 75
Program Interruption Conditions . . . 75
Operation Exception . . . 76
Privileged-Operation Exception . . . 76
Execute Exception . . . 76
Protection Exception . . . 76
Addressing Exception . . . 76
Specification Exception . . . 77
Data Exception . . . 78
Fixed-Point-Overflow Exception . . . 78
Fixed-Point-Divide Exception . . . 78
Decimal-Overflow Exception . . . 78
Decimal-Divide Exception . . . 78
Exponent-Overflow Exception . . . 78
Exponent-Underflow Exception . . . 79
Significance Exception . . . 79
Floating-Point-Divide Exception . . . 79
Segment-Translation Exception . . . 79
Page-Translation Exception . . . 79
Translation-Specification Exception . . . 79
Special-Operation Exception . . . 80
Monitor Event . . . 80
Program Event . . . 80
Recognition of Access Exceptions . . . 80
Handling of Multiple Program-Interruption Conditions . . . 83
Supervisor-Call Interruption . . . 84
External Interruption . . . 84
Interval Timer . . . 86
Interrupt Key . . . 86
External Signal . . . 86
Malfunction Alert . . . 86
Emergency Signal . . . 87
External Call . . . 87
Time-of-Day Clock Sync Check . . . 87
Clock Comparator . . . 87
CPU Timer . . . 88
Input/Output Interruption . . . 88
Restart . . . 88
Priority of Interruptions . . . 89
Assigned Main-Storage Locations . . . 90
Real Main Storage . . . 90
Absolute Main Storage . . . 91

The interruption system permits the CPU to change its state as a result of conditions external to the system, within the system, 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: input-output, external, program, supervisor call, machine check, 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 execution of the instruction causing the interruption normally is resumed, 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 either stopped or operating.

The details of source identification, location determination, and instruction execution are explained in later sections and are summarized in the table "Interruption Action."

Programming Note

See the section "Program Status Word" in the chapter "System Control" for details as to when the new PSW introduced by an interruption is checked for format errors.

Source Identification

The six classes of interruptions (I/O, external, program, supervisor call, machine check, 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 main storage during the interruption. For I/O, external, supervisor-call, and program interruptions, the interruption code comprises 16 bits and is placed in the old PSW when the old PSW specifies the BC mode and in separate main-storage locations when the EC mode is specified.

For I/O interruptions, additional information is provided by the contents of the channel status word (CSW) stored at location 64, and further information may be provided by the limited channel logout stored at location 176 and by the I/O extended logout.

For program interruptions, additional information may be provided in the form of the translation-exception address, monitor-class number and monitor code, and PER code and PER address stored at locations 144-159.

For machine-check interruptions, the interruption code comprises 64 bits and is placed in main storage at location 232. Additional information for identifying the cause of the interruption and for recovering the state of the CPU may be provided by the contents of the logout and save areas.

The assignment and format of the permanently allocated storage locations is shown in the table "Permanently Assigned Storage Locations" at the end of this chapter.

Enabling and Disabling

The CPU may be enabled or disabled for all I/O, external, and machine-check interruptions and for some program interruptions. When the CPU is enabled for a class of interruptions, these interruptions can take place. When the CPU is disabled, the conditions that cause I/O interruptions remain pending, and the disallowed program-interruption conditions are ignored, except that some causes are indicated also by the setting of the condition code. External and machine-check conditions, depending on the type, are ignored or remain pending.

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

Whether the CPU is enabled or disabled for a particular type of interruption is controlled by mask bits in the current PSW and in control registers. The setting of the mask bits may disallow all interruptions within the class or may selectively allow interruptions for particular causes. This control is provided by assigning a mask bit in the PSW to a particular cause, such as in the case of the four maskable program interruption conditions, or by providing a hierarchy of masks, where a mask in the PSW controls all interruptions within a type, and masks in control registers 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 mask bits in the hierarchy are ones.

Programming Note

Mask bits in the PSW provide a means of disabling 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.

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.

In an old PSW specifying the BC mode, the instruction-length code is stored in bit positions 32 and 33. It is meaningful, however, only after a program or supervisor-call interruption. For I/O, external, machine-check, and restart interruptions, the code 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 store-status function and when the PSW is displayed.

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

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 subject 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			
Decimal	Binary	Instruction Bits 0-1	Instruction Length
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. Instruction-length code 0 occurs only in the following cases:

1. 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. In the case of LOAD PSW, the address of the instruction has been replaced by the new PSW. When the invalid PSW is introduced by an interruption, the format error cannot be attributed to an instruction.
2. On some models, when an addressing exception (excluding those detected during implicit references to dynamic-address-translation-table entries) or a protection exception is recognized during a store-type reference. In these cases the interruption due to the exception is delayed, the length of time or number of instructions of the delay being unpredictable. Neither the location of the instruction causing the exception nor the length of the last-executed instruction is made available to the program.

When the new PSW introduced by LOAD PSW or a supervisor-call interruption has a format error,

			PSW Mask Bits		Mask Bits in Control Registers			
Source Identification	Interruption Code		BC	EC	Register	Bit	ILC Set	Execution of Instruction Identified by Old PSW
Machine check (old PSW 48, new PSW 112)								
Exigent condition	mmmmmmmm	mmmmmmmm[1]	13	13			x	terminated or nullified[7]
Repressible cond.	mmmmmmmm	mmmmmmmm[1]	13	13	14	4-7	x	unaffected[7]
Supervisor call (old PSW 32, new PSW 96)								
Instruction bits	00000000	rrrrrrrr					1,2	completed
Program (old PSW 40, new PSW 104)								
Operation	00000000	n0000001					1,2,3	suppressed
Privileged oper.	00000000	n0000010					1,2	suppressed
Execute	00000000	n0000011					2	suppressed
Protection	00000000	n0000100					0,1,2,3	suppressed or terminated
Addressing	00000000	n0000101					0,1,2,3	suppressed or terminated
Specification	00000000	n0000110					0,1,2,3	suppressed or completed
Data	00000000	n0000111					2,3	suppressed or terminated
Fixed-pt. overflow	00000000	n0001000	36	20			1,2	completed
Fixed-point divide	00000000	n0001001					1,2	suppressed or completed
Decimal overflow	00000000	n0001010	37	21			2,3	completed
Decimal divide	00000000	n0001011					2,3	suppressed
Exponent overflow	00000000	n0001100					1,2	completed
Exponent underflow	00000000	n0001101	38	22			1,2	completed
Significance	00000000	n0001110	39	23			1,2	completed
Floating-pt. divide	00000000	n0001111					1,2	suppressed
Segment transl.	00000000	n0010000					1,2,3	nullified
Page translation	00000000	n0010001					1,2,3	nullified
Translation spec	00000000	n0010010					1,2,3	suppressed
Special operation	00000000	n0010011			0	1	2	suppressed
Monitor event	00000000	n1000000			8	16+	2	completed
Program event	00000000	1e0eeeee[2]	*	1	9	0-3	0,1,2,3	completed[3]
External (old PSW 24, new PSW 88)								
Interval timer	00000000	1nnnnnnn	7	7	0	24	x	unaffected
Interrupt key	00000000	n1nnnnnn	7	7	0	25	x	unaffected
External signal 2	00000000	nn1nnnnn	7	7	0	26	x	unaffected
External signal 3	00000000	nnn1nnnn	7	7	0	26	x	unaffected
External signal 4	00000000	nnnn1nnn	7	7	0	26	x	unaffected
External signal 5	00000000	nnnnn1nn	7	7	0	26	x	unaffected
External signal 6	00000000	nnnnnn1n	7	7	0	26	x	unaffected
External signal 7	00000000	nnnnnnn1	7	7	0	26	x	unaffected
Malfunction alert	00010010	00000000	7	7	0	16	x	unaffected
Emergency signal	00010010	00000001	7	7	0	17	x	unaffected
External call	00010010	00000010	7	7	0	18	x	unaffected
TOD clock sync chk	00010000	00000011	7	7	0	19	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[4]	0	6	2	0[5]	x	unaffected
Channel 1	00000001	dddddddd[4]	1	6	2	1[5]	x	unaffected
Channel 2	00000010	dddddddd[4]	2	6	2	2[5]	x	unaffected
Channel 3	00000011	dddddddd[4]	3	6	2	3[5]	x	unaffected
Channel 4	00000100	dddddddd[4]	4	6	2	4[5]	x	unaffected
Channel 5	00000101	dddddddd[4]	5	6	2	5[5]	x	unaffected
Channels 6 & on	cccccccc	dddddddd[4]	6	6	2	6+	x	unaffected
Restart (old PSW 8, new PSW 0)								
Restart key	00000000	00000000[6]					x	unaffected

Explanation:

1 A machine-check interruption code of 64 bits is stored at locations 232-239.

2 When the interruption code indicates a program event, an ILC of zero may be stored only when the code formed by bits 12-15 of the interruption code has a nonzero value.

3 The unit of operation is completed, unless a program exception concurrently indicated has caused the unit of operation to be nullified, suppressed, or terminated.

4 In the EC mode, the I/O address is stored at locations 186-187.

5 For channels 0-5, channel masks in control register 2 have no effect in BC mode.

6 Bits 16-31 in the old PSW in BC mode are set to zeros. No interruption code is provided in EC mode.

7 For any machine-check interruption condition, either exigent or repressible, the effect of this condition is identified by the validity bits in the machine-check interruption code. The instruction has been nullified or unaffected only if the associated validity bits are set to ones.

+ Plus the following bits in the control register.

* In BC mode, program-event recording is disabled.

c Channel address bits.

d Device address bits.

e A possible nonzero code indicating another program interruption condition.

m Bits of model-dependent code.

n Possible bit-significant indication of other concurrent interruption conditions.

r Bits of the I field of SUPERVISOR CALL.

x Unpredictable in BC mode; not stored in EC mode.

Interruption Action (Continued)

other than an odd instruction address, and, concurrently, the LOAD PSW or SUPERVISOR CALL instruction causes a program event, the ILC is 0, as called for in the specification exception.

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 a protection, addressing, or translation-specification exception is encountered on fetching an instruction, the instruction-length code is 1, 2, or 3, indicating the number of halfwords by which the instruction address has been incremented. When the instruction address in the old PSW is reduced by the number of halfword locations indicated by the instruction-length code, the address originally appearing in the PSW is obtained. It is unpredictable whether the code is 1, 2, or 3.
2. When a segment-translation or page-translation exception is recognized on the access to an instruction, the ILC is 1, 2, or 3, with the indication being unpredictable. 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.

When any exceptions are encountered on fetching the subject instruction of EXECUTE, the ILC is 2.

Programming Notes

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:

1. The interruption is caused by a segment-translation or page-translation exception.
2. An interruption for a program event occurs before the completion of the execution of an interruptible instruction.
3. The interruption is caused by a program event due to a branch instruction, LOAD PSW, or SUPERVISOR CALL.
4. 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).
5. The interruption is caused by a specification exception because of an odd instruction address.
6. The interruption is caused by a specification or access exception encountered in fetching an instruction, and changes have been made or may have been made to the parameters that control the relation between the logical and real instruction address (turning the translation mode on or off without introducing an entire new PSW, changing the translation-control

parameters in control registers 0 and 1, introducing invalid values in bit positions 0-7 of an EC PSW).

For situations 1 and 2, the 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 3, 4, and 5, the instruction address in the program old PSW has been replaced and cannot be calculated using the one appearing in the PSW.

For situation 6, the logical instruction address in the PSW has not been replaced, but the corresponding real address after the change is different.

When bit 8 (program event) in the interruption code is on, the PER address at locations 153-155 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 ILC can be derived from the operation code of the instruction identified by the old PSW.

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 an operation.

For the two instructions MOVE LONG and COMPARE LOGICAL LONG, referred to as interruptible instructions, an interruption is permitted after a partial execution of the instruction. 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 model and on the particular condition that causes the execution of the instruction to be interrupted.

Whenever discussion in this publication pertains to points of interruptibility that include those occurring within the execution of an interruptible instruction, the term "unit of operation" is used. This use of the term considers that the entire execution of the noninterruptible instruction consists, in effect, of one unit of operation.

Programming Note

Any interruption, other than supervisor call and some program interruptions, can occur after a partial execution of an interruptible instruction. In particular, interruptions for I/O, external, and machine-check conditions and for program access exceptions can occur between units of operation.

Instruction Execution

Types of Ending

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

When the execution of an instruction is completed, results are provided 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.

When the execution of an instruction is suppressed, the instruction is 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 is the same as suppression, except that when an interruption occurs after the execution of the instruction has been nullified, the instruction address in the old PSW designates the instruction whose execution was nullified instead of the next sequential instruction.

When the execution of an instruction is terminated, the contents of any fields due to be changed by the instruction are 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 system 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.

Execution of Interruptible Instructions

The execution of an interruptible instruction is completed when all units of operation associated with that instruction are completed. When an interruption occurs after completion, nullification, or suppression of a unit of operation, all 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.

Machine-Check Interruption

The machine-check interruption provides 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 interruption code and the instruction-length code in the old PSW are unpredictable.

The cause and severity of the malfunction are identified by a 64-bit machine-check code stored at location 232. Further information identifying the cause of the interruption and the location of the fault may be stored at locations 216-511 and in the area starting with the location designated by the contents of control register 15.

Interruption action and the storing of the associated information are under the control of PSW bit 13 and bits in control register 14. See the chapter "Machine-Check Handling" for more detailed information.

Program Interruption

Exceptions resulting from execution of the program, including the improper specification or use of instructions and data, or the detection of a program or monitor event cause 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 BC mode, the interruption code and the instruction-length code are placed in the old PSW; when it 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 zero, and zeros are stored at location 140. For some causes additional information identifying the reason for the interruption is stored in main-storage locations 144-159.

Except for the program-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 program-event condition is indicated by setting bit 8 of the interruption code to one, with bits 0-7 set to zeros. A program-event condition can be indicated concurrently with another program interruption condition, in which case bit 8 is one and the coded value 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 control interruptions due to monitor events, and, in the EC mode, masks are provided for controlling interruptions due to program events. When the mask bit is zero, the condition is ignored; the condition does not remain pending.

Programming Note

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 "Priority of Interruptions" for a description of how such strings are terminated.

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.

Operation Exception

An operation exception is recognized when the CPU encounters an instruction with an invalid operation code. The operation code may not be assigned, or the instruction with that operation code may not be available on the CPU. For the purpose of recognizing an operation exception, the first eight bits of an instruction, or, when the first eight bits have the hexadecimal value B2, the first 16 bits form the operation code.

The operation is suppressed.

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

Programming Note

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 the seven I/O instructions are shown as 9C00, 9C01, 9D00, 9D01, 9E00, 9E01, and 9F00.

Some models may offer instructions not listed in this manual, such as those provided for emulation or as part of special or custom features. Consequently, all unlisted operation codes do not necessarily cause an operation exception to be recognized. Furthermore, as part of the specified operation, these instructions may cause modes of operation to be set up or otherwise alter the system so as to affect the execution of subsequent instructions. In order to avoid the possibility of accidentally causing such operation, instructions with an unlisted operation code should be issued only when the specific function associated with the operation code is desired.

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 software uses where indication of invalid operation is required. It is improbable that these operation codes will ever be assigned to an instruction implemented in the CPU.

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.

Execute Exception

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

The operation is suppressed.

The instruction-length code is 2.

Protection Exception

A protection exception is recognized when the CPU causes a reference to a main-storage location that is protected against the type of reference, and the key in storage associated with the location does not match the protection key in the PSW.

The execution of the instruction is suppressed when the location of the instruction, including the location of the subject 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. See the following table for a summary of the action taken on a protection exception.

On fetching, the protected information is not loaded into an addressable register or moved to another storage location. When 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. However, for a store-protected operand location, the instruction-length code on some models may be 0.

When the location of any part of the instruction is protected against fetching, the instruction-length code is 1, 2, or 3, indicating the number of halfwords by which the instruction address has been incremented. It is unpredictable whether the code is 1, 2, or 3.

Addressing Exception

An addressing exception is recognized when the CPU causes a reference to a main-storage location that is not available to the CPU. A main-storage location is not available to the CPU when the location is not provided, when the storage unit is not configured to the CPU, or when power is off in the storage unit. An address designating an unavailable storage location is referred to as invalid.

The execution of the instruction is suppressed when the address of the instruction, including the location of the subject instruction of EXECUTE, is invalid. Similarly, the unit of operation is suppressed when the exception is encountered during an implicit reference to a dynamic-address-translation (DAT) table entry Except for some specific instructions whose execution is suppressed, the operation is terminated for an operand address that can be translated but designates an unavailable location. See the following table for a summary of the action taken on an addressing exception.

Data in storage remains unchanged unless the location is available to the CPU. When part of an operand location is available to the CPU and part is not, storing may be performed in the available part.

For an invalid operand address or an invalid address of a DAT table entry associated with an operand reference, the instruction-length code is 1, 2, or 3, designating the length of the instruction that caused the reference. However, when the exception is due to an attempt to store and the address can be translated but designates an unavailable operand location, the code on some models may be 0.

When any part of the location of an instruction is unavailable or the address of a DAT table entry associated with an instruction fetch is invalid, the instruction-length code is 1, 2, or 3, indicating the number of halfword locations by which the instruction address has been incremented. It is unpredictable whether the code is 1, 2, or 3.

Specification Exception

A specification exception is recognized for the following causes:

1. An instruction address does not designate a location on an even-byte boundary.
2. An operand address does not designate an integral boundary in an instruction requiring such integral boundary designation.
3. The block 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 first-operand field is shorter than or equal to the second-operand field in decimal multiplication or division.
8. Bit positions 8-11 of MONITOR CALL do not contain zeros.
9. The EC mode is specified (PSW bit 12 is one) in a CPU that does not have the EC facility installed.
10. A one is introduced into an unassigned bit position of the EC-mode PSW (bit positions 0, 2-4, 16-17, 24-39).

The execution of the instruction identified by the old PSW is suppressed. However, for causes 9 and 10, the operation that introduces the new PSW is completed, but 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 number of halfword locations by which the instruction address has been incremented. It is unpredictable whether the code is 1, 2, or 3.

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

When the exception is recognized because of causes 9 and 10 and the invalid bit value has been introduced by LOAD PSW or an interruption, the ILC is 0. When the exception due to cause 10 is

	Action On		
Exception	DAT Table Entry Fetch	Instruction Fetch	Operand Reference
Protection Exception	– –	Suppress	Terminate[1], but suppress LPSW, SSM, STNSM, STOSM, SCKC, SPT, SPX
Addressing Exception	Suppress	Suppress	Terminate[1], but suppress LPSW, SSM, STNSM, STOSM, SCKC, SPT, SPX

Explanation:

– – Not applicable.

[1] For termination, changes may occur only to result fields. In this context, "result field" includes condition code, registers, and storage locations, if any, which are designated to be changed by the instruction. However, no change is made to a storage location or a key in storage when the reference causes an access exception. Therefore, if an instruction is due to change only the contents of a field in main storage, and every byte of that field would cause an access exception, the operation is suppressed.

Summary of Action for Protection and Addressing Exceptions introduced by SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the ILC is 2.

See "Program Status Word" in the chapter "System Control" for a discussion of when the exceptions associated with the PSW are recognized.

Data Exception

A data exception is recognized when:

1. The sign or digit codes of operands in the decimal-feature 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.

Except for EDIT and EDIT AND MARK, the operation is suppressed when a sign code is invalid, regardless of whether any other condition causing the exception exists; otherwise, the operation is terminated. However, 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, and the operation is terminated on a data exception.

The instruction-length code is 2 or 3.

Programming Note

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:

- The invalid sign is not located in the numerical portion of the result field.
- 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.

Fixed-Point-Overflow Exception

A fixed-point-overflow exception is recognized when a carry occurs out of the high-order bit position in fixed-point arithmetic operations, or high-order significant bits are lost during the algebraic left-shift operations.

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

The operation is completed by setting condition code 3 but otherwise ignoring the information placed outside the register.

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.

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 in the BC mode by PSW bit 37, and in the EC mode by PSW bit 21.

The operation is completed by setting condition code 3 but otherwise ignoring the overflow information.

The instruction-length code is 2 or 3.

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 operation is suppressed.

The instruction-length code is 2 or 3.

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 BC mode by PSW bit 38, and in the EC mode by PSW bit 22.

The operation is completed. The setting of 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.

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 BC mode by PSW bit 39, and in the EC mode by PSW bit 23.

The operation is completed. The significance mask affects also 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.

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.

Segment-Translation Exception

A segment-translation exception is recognized when:

1. The segment-table entry is outside the segment table.
2. The segment-invalid bit has the value 1.

The exception is recognized as part of the execution of the instruction that needs the segment-table entry in the translation of either the instruction or operand address, except for the operand address in LOAD REAL ADDRESS, in which case the condition is indicated by the setting of the condition code.

The unit of operation is nullified.

The segment and page portion of the logical address causing the exception is placed in main storage at locations 145-147, and zeros are placed at location 144. When 2,048-byte pages are used, the low-order 11 bits of the address are unpredictable; when 4,096-byte pages are used, the low-order 12 bits of the address 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 indication being unpredictable.

Page-Translation Exception

A page-translation exception is recognized when:

1. The page-table entry is outside the page table.
2. The page-invalid bit has the value 1.

The exception is recognized as part of the execution of the instruction that needs the page-table entry in the translation of either the instruction or operand address, except for the operand address in LOAD REAL ADDRESS, in which case the condition is indicated by the setting of the condition code.

The unit of operation is nullified.

The segment and page portion of the logical address causing the exception is placed in main storage at locations 145-147, and zeros are placed at location 144. When 2,048-byte pages are used, the low-order 11 bits of the address are unpredictable; when 4,096-byte pages are used, the low-order 12 bits of the address 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 indication being unpredictable.

Translation-Specification Exception

A translation-specification exception is recognized when:

1. Bit positions 8 and 9 of control register 0 contain values 00 or 11.
2. Bit position 10 of control register 0 contains a one.
3. Bit positions 11 and 12 of control register 0 contain values 01 or 11.
4. Bit positions 4-7 or 29-30 in a valid segment-table entry do not contain zeros (on some models these bit positions are not checked for zeros).
5. Depending on the page size, the one or two bit positions next to the low-order bit in a valid page-table entry do not contain zeros.

The exception is recognized only as part of the execution of an instruction using address translation, that is, when an instruction is executed with bit 5 of the EC-mode PSW one or when LOAD REAL ADDRESS is executed. Causes 1-3 are recognized on any translation attempt; causes 4 and 5 are recognized only for table entries that are actually used.

The unit of operation is suppressed.

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, indicating the number of halfword locations by which the instruction address has been updated. It is unpredictable whether the code is 1, 2, or 3.

Programming Note

When a translation-specification exception is recognized in the process of translating an instruction address, the operation is suppressed. In this case, the instruction-length code (ILC) is needed to derive the address of the instruction, as the instruction address in the old PSW has been incremented by the amount specified by the ILC. In the case of segment-translation and page-translation exceptions, the operation is nullified, the instruction address in the old PSW identifies the instruction, and the ILC is redundant.

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.

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 placed in main storage at locations 148-149 and 156-159. See "Monitoring" in the chapter "System Control" for a detailed description of the interruption condition.

The instruction-length code is 2.

Program Event

A program event is recognized when program-event recording 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 placed in main storage at locations 150-155. See "Program-Event Recording" in the chapter "System 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 because of a protection addressing or specification condition that is concurrently indicated.

Recognition of Access Exceptions

The protection, addressing, segment-translation, page-translation, and translation-specification exceptions are collectively referred to as *access* exceptions. The table "Handling of Access Exceptions" summarizes the conditions that can cause these exceptions and the action taken when they are encountered.

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

	Implicit Translation			Explicit Translation (Operand of LRA)	
	Indication				
Condition	Instruction	Operand	Action	Indication	Action
Control register contents[1]					
Invalid page size (CR 0 bits 8 and 9)	TS	**	suppress	TS	suppress
One in bit position 10 of control register 0	TS	**	suppress	TS	suppress
Invalid segment size (CR0 bits 11 and 12)	TS	**	suppress	TS	suppress
Segment table entry					
Segment table length violation	ST	ST	nullify	cc3	complete
Entry protected against fetching or storing	—	—	—	—	—
Invalid address of entry	A	A	suppress	A	suppress
I bit on	ST	ST	nullify	cc1	complete
One in an unassigned bit position[2]	TS	TS	suppress	TS	suppress
Page table entry					
Page table length violation	PT	PT	nullify	cc3	complete
Entry protected for fetching or storing	—	—	—	—	—
Invalid address of entry	A	A	suppress	A	suppress
I bit on	PT	PT	nullify	cc2	complete
One in an unassigned bit position[2]	TS	TS	suppress	TS	suppress
Access for instruction or data					
Location protected	P	P	*	—	—
Invalid address	A	A	*	—	—

Explanation:

TS	Translation-specification exception.
ST	Segment-translation exception.
PT	Page-translation exception.
A	Addressing exception.
P	Protection exception.
cc1	Condition code 1 set.
cc2	Condition code 2 set.
cc3	Condition code 3 set.
—	The condition does not apply.

* Action depends on the type of reference.

** The condition cannot occur because it is recognized as part of the translation of the instruction address.

[1] A translation-specification exception for an invalid code in control register 0 bit positions 8-12 is recognized as part of the execution of the instruction using address translation.

[2] A translation-specification exception for a format error in a table entry is recognized only when the execution of an instruction requires the entry for the translation of an address.

Handling of Access Exceptions

cases is summarized in the table "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 (TRT) 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 (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 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.

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:	
TM (zero mask)	Yes
CLC, CL	Unpredictable
CLM (nonzero mask)	Unpredictable
CLCL	Unpredictable*
TRT (first operand)	No
TR, TRT (second operand)	No
ED, EDMK (second operand)	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, no exceptions are indicated other than those for the current page and the following page of each operand.

Access exceptions include the following:

protection
addressing
segment translation
page translation
translation specification

Recognition of Access Exceptions

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.

The execution of the interruptible instructions COMPARE LOGICAL LONG and MOVE LONG is initiated only when no access exceptions associated with references to dynamic-address-translation tables for the initial page of each operand exist, and the initiation may additionally be contingent on the absence of exceptions associated with table references for the following page of each operand. After the execution of the instruction has been initiated, an access exception associated with a reference to a translation table may be indicated as early as when execution has progressed to the point where the last accessible page of the operand causing the exception is being processed.

The extent of the operands that is actually used in the operation may be established in a pretest for operand accessibility that is performed before the execution of the instruction is started.

In the case of TRANSLATE (TR), EDIT (ED), and EDIT AND MARK (EDMK), the initiation of the execution is contingent only on the absence of exceptions associated with dynamic-address-translation table entries 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, or by another CPU, after the initial pretest but before completion of execution, such that an additional second-operand page is needed and translation of the address of the additional page causes an access exception to be recognized, results are unpredictable. Furthermore, it is unpredictable whether an interruption for the access exception occurs. In the case of ED and EDMK, this situation can occur also because of overlapping operands.

This case is an exception to the general rule that the operation is nullified on segment-translation and page-translation exceptions and is suppressed on a translation-specification exception and on an addressing exception caused by an invalid address of a table entry. When, in this case, an interruption for a segment-translation or page-translation occurs, the instruction address in the old PSW points to the instruction causing the exception even though partial results have been stored.

Programming Notes

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 store protection when applicable, are recognized even if the operation could be completed without the use of the inaccessible part of the operand:

- Fetching the operand of TEST UNDER MASK with a zero mask.
- Fetching parts of operands of algebraic compare instructions (C and CH).
- Fetching parts of operands of floating-point instructions.
- 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.
- Storing the pattern character in an edit operation when the pattern character remains unchanged.
- Storing during SHIFT AND ROUND DECIMAL when no shifting or rounding takes place.
- Storing during move operations when the first- and second-operand locations coincide.
- 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.
- 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.

Handling of Multiple Program-Interruption Conditions

Except for program 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 or a protection 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 table "Priorities of Access Exceptions" lists the priorities of access exceptions for a single access. The table "Priorities of Program Interruption Conditions" lists the priorities of all program-interruption conditions other than program 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." Thus, the first table specifies which of several exceptions that are encountered in the access of a particular portion of an instruction, or in any particular access associated with an operand, has highest priority, and the latter table specifies the priority of this condition in relation to other conditions detected in the operation.

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

1. Translation-specification exception due to invalid page size or segment size designation or due to a one in bit position 10 of control register 0.
2. Segment-translation exception due to segment-table entry being outside table.
3. Addressing exception due to segment-table entry being outside main storage of installation.
4. Segment-translation exception due to I bit having the value one.
5. Translation-specification exception due to invalid ones in segment-table entry.
6. Page-translation exception due to page-table entry being outside table.
7. Addressing exception due to page-table entry being outside main storage of installation.
8. Page-translation exception due to I bit having the value one.
9. Translation-specification exception due to invalid ones in page-table entry.
10. Addressing exception due to instruction or operand location outside main storage of installation.
11. Protection exception due to attempt to access a protected instruction or operand location.

Explanation:
The access exceptions are listed in the order of descending priorities.

Priorities of Access Exceptions

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.

Supervisor-Call Interruption

The supervisor-call interruption occurs as a result of the execution of the instruction SUPERVISOR CALL. 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 BC mode, the interruption code and instruction-length code appear in the old PSW; 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 zero, and zeros are stored at location 136.

Programming Note

The name "supervisor call" indicates that one of the major purposes of the interruption is the switching from problem to supervisor state. This major purpose does not preclude the use of this interruption for other types of status switching.

The interruption code may be used to convey a message from the calling program to the supervisor.

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 system.

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 BC mode, the interruption code is placed in bit positions 16-31 of the old PSW, and the instruction-length code is unpredictable. When the old PSW specifies the EC mode, the interruption code is placed at locations 134-135.

Additionally, in both the BC and EC modes, for some conditions a 16-bit processor address is associ-

1.A Delayed addressing exception due to an attempted store by a previous instruction (zero ILC).

1.B Delayed protection exception due to an attempted store by a previous instruction (zero ILC).

2. Specification exception due to any PSW error of the type that causes an immediate interruption.[1]

3. Specification exception due to an odd instruction address in the PSW.

4. Access exceptions for first instruction halfword.[2]

5.A Access exception for second instruction halfword.[2]

5.B Access exception for third instruction halfword.

5.C.1 Operation exception.

5.C.2 Privileged-operation exception.

5.C.3 Execute exception.

5.C.4 Special-operation exception.

5.D Specification exception, due to conditions other than those included in 2 and 3 above, for an instruction that is not installed but has an operation code assigned.

6.A Specification exception due to conditions other than those included in 2, 3, and 5.D above.[2]

6.B-.G* Access exceptions for any particular access to an operand in main storage.[3]

6.H Data exception.[4]

6.I Decimal-divide exception.[4]

7.-14. Fixed-point divide, floating-point divide, and conditions, other than program events, which result in completion. These conditions are mutually exclusive.

Explanation:

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

* As in instruction fetching, separate accesses may occur for each portion of an operand. Each of these accesses is of equal priority, and in effect a different letter is assigned to each. There is a maximum of six different operand access exceptions corresponding to fetch accesses to two operands, each of which crosses a protection or page boundary, and store accesses to one operand which crosses a boundary. Access exceptions for INSERT STORAGE KEY, SET STORAGE KEY, RESET REFERENCE BIT, and LOAD REAL ADDRESS are also included in 6.B.

[1] PSW errors which cause an immediate interruption may be introduced by a new PSW loaded as a result of an interruption or by the instructions LPSW, SSM, and STOSM. The priority shown in the chart is that for the case of an error introduced by an interruption and may also be considered as the priority for the case of an 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.

[2] In the case of an EXECUTE instruction, both EXECUTE and the subject instruction of the EXECUTE must be accessed and interpreted. In this case, the priorities shown are for the subject instruction. The priority of exceptions associated with the EXECUTE can be considered as being prefixed with a "3.", thus occurring between priorities 3 and 4, and numbered as follows: 3.4, 3.5.A, and 3.6.A.

[3] 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 the byte-oriented operand feature applied.

[4] The exception can be indicated only if the sign, digit, or digits responsible for the exception were fetched without encountering an access exception.

Priorities of Program Interruption Conditions

ated with the source of the interruption and is stored at locations 132-133. When the processor address is stored, bit 6 of the interruption code is set to one. When bit 6 is zero and the old PSW specifies the BC mode, the contents of locations 132-133 remain unchanged. When bit 6 is zero and the old PSW specifies the EC mode, zeros are stored at locations 132-133.

An external interruption for a particular source can occur only when the CPU is enabled for interruption by that source. 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 is assigned a submask bit, 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 BC or EC 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 highest priority is assigned to the set of conditions that includes the interval timer, interrupt key, and external signals 2 through 7. Within this set, all pending requests for which the CPU is enabled are indicated concurrently in the interruption code. Next in priority are interruption requests for the following sources, the sources being listed in descending order of priority:

Malfunction alert
Emergency signal
External call
Time-of-day clock sync check
Clock comparator
CPU timer

When more than one emergency-signal or malfunction-alert request exists at a time, the request associated with the smallest processor address is honored first. Only one occurrence each of these conditions can be indicated at a time in the external-interruption code.

Interval Timer

An interruption request for the interval timer is generated when the value of the interval timer is decremented from a positive number, including 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 CPU reset.

The condition is indicated by setting bit 8 in the interruption code to one and by setting bits 0-7 to zero. Bits 9-15 are zero unless set to one for another condition that is concurrently indicated. In the EC mode, zeros are stored at locations 132-133.

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

Interrupt Key

An interruption request for the interrupt key is generated when the interrupt key on the operator section of the system control panel is activated. The request is preserved and remains pending in the CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

The condition is indicated by setting bit 9 in the interruption code to one and by setting bits 0-7 to zero. Bits 8 and 10-15 are zero unless set to one for another condition that is concurrently indicated. In the EC mode, zeros are stored at locations 132-133.

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

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 CPU reset.

External signals 2 through 7 are indicated by setting to one interruption code bits 10-15, respectively. Bits 0-7 are set to zero, and any other bits in the low-order byte are made zero unless set to one for another condition that is concurrently indicated. In the EC mode, zeros are stored at locations 132-133.

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.

The facility to accept external signals is part of the direct-control feature. On some models, it is also available as a separate feature.

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.

Malfunction Alert

An interruption request for malfunction alert is generated when another CPU that is configured to the CPU enters the check-stop state or loses power. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Facilities are provided for holding a separate malfunction-alert request pending in the receiving CPU for each other configured CPU. Configuring a CPU out of the system does not generate a malfunction-alert condition.

The condition is indicated by an external-interruption code of 1200 (hex). The processor address of the CPU that generated the condition is stored at locations 132-133.

The subclass mask bit is located in bit position 16 of control register 0. This bit is initialized to zero.

Emergency Signal

An interruption request for emergency signal is generated when the CPU accepts the emergency-signal order specified by a SIGNAL PROCESSOR instruction addressing this CPU. The instruction may have been executed by this CPU or by another CPU configured to this CPU. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Facilities are provided for holding a separate emergency-signal request pending in the receiving CPU for each configured CPU, including the receiving CPU itself.

The condition is indicated by an external-interruption code of 1201 (hex). The processor address of the CPU that issued the SIGNAL PROCESSOR instruction is stored at locations 132-133.

The subclass mask bit is located in bit position 17 of control register 0. This bit is initialized to zero.

External Call

An interruption request for external call is generated when the CPU accepts the external-call order specified by a SIGNAL PROCESSOR instruction addressing this CPU. The instruction may have been executed by this CPU or by another CPU configured to this CPU. The request is preserved and remains pending in the receiving CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

Only one external-call request, along with the processor address, may be held pending in a CPU at a time.

The condition is indicated by an external-interruption code of 1202 (hex). The processor address of the CPU that issued the SIGNAL PROCESSOR instruction is stored at locations 132-133.

The subclass mask bit is located in bit position 18 of control register 0. This bit is initialized to zero.

Time-of-Day Clock Sync Check

The time-of-day (TOD) clock sync check condition indicates that more than one TOD clock exists in the configuration, and that the low-order 32 bits of the clocks are not running in synchronism.

An interruption request for TOD clock sync check exists when the clock accessed by this CPU is running, the clock accessed by any other CPU configured to this CPU is running, and bits 32-63 of the two clocks do not match. When a clock enters the running state, or a running clock is added to the configuration, a delay of up to 1.048576 seconds (2^{20} microseconds) may occur before the mismatch condition is recognized.

When only two clocks are in the configuration and either or both of the clocks are in the error, stopped, or not-operational state, it is unpredictable whether a TOD clock sync check condition is recognized, and, if it is recognized, it may continue to persist up to 1.048576 seconds after both clocks have been running with low-order bits matching. However, in this case, the condition does not persist if the two CPUs are configured apart.

When more than one CPU shares a TOD clock, only the CPU with the smallest processor address among those sharing the clock indicates a sync-check condition associated with that clock.

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 1003 (hex). In the EC mode, zeros are stored at locations 132-133.

The subclass mask bit is located in bit position 19 of control register 0. This bit is initialized to zero.

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 running, 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 binary unsigned quantities.
2. The clock comparator is installed, and the time-of-day clock is in the error state or not operational.

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). In the EC mode, zeros are stored at locations 132-133.

The submask bit is located 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). In the EC mode, zeros are stored at locations 132-133.

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

Input/Output Interruption

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

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 and in the form of an I/O extended logout starting at the location designated by the contents of locations 173-175.

When the old PSW specifies the BC mode, the interruption code in PSW bit positions 16-31 identifies the channel and device causing the interruption: the channel address appears in the high-order eight bit positions and the device address in the low-order eight. The instruction-length code is unpredictable. When the old PSW specifies the EC mode, the device address is placed at location 187, the channel address at location 186, and zeros are stored at location 185.

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

Channel mask bits are located in control register 2 starting at bit position 0 and extending 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. The state of channel mask bits for unavailable channels is unpredictable.

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 is one and the corresponding channel mask is one. Interruptions from channels 0-5 are controlled by channel masks 0-5 in the PSW; an interruption can occur only when the mask 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.

When the current PSW specifies the EC mode, each channel is controlled by the I/O mask bit 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 CPU becomes enabled for a pending I/O-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

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. 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 the chapter on I/O operations.

Restart

The restart interruption provides a means for the operator or another CPU 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 main-storage location 8 and a new PSW to be fetched from location 0. In the BC mode, the instruction-length code in the PSW is unpredictable, and zeros are stored in the interruption-code field. In the EC mode, the instruction-length and interruption codes are not stored.

If the CPU is in the operating state, the exchange of the PSWs occurs at the completion of the current unit of operation and after all pending interruption conditions for which the CPU is enabled have been taken. In this case, it depends on the model if the CPU temporarily enters the stopped state as part of the execution of the restart operation. 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 on the system console. In a multiprocessing system, the operation can also be initiated at the addressed CPU by issuing SIGNAL PROCESSOR, specifying the restart order.

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 program-addressable registers, including the control registers, but causes the attached 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 installed. 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 the following order of priority (the conditions are listed in descending order of priorities):

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. This 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 unacceptable instruction address (the instruction address is odd 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 in an EC-mode PSW, or if it specifies the EC mode in a CPU that does not have the EC facility installed, or if it specifies any other facility that is not installed on the CPU, another program interruption occurs. This condition is of higher priority than restart, I/O, external, or repressible machine-check conditions, or the stop function, and CPU reset has to be used to break the loop.

Interruption loops of other interruption classes can also exist if the new PSW is enabled for the same interruption. These include machine-check interruptions and external interruptions due to channel-available or PCI conditions. Interruption loops involving more than one interruption class can also exist. For example, assume that the CPU timer is negative and the CPU-timer subclass mask is one. If the external new PSW has an exception which is recognized as part of early recognition, and the program new PSW is enabled for external interruptions, then a series of interruptions occur, alternating between external and program. Even more complex loops are possible. So long as more interruptions must be serviced, the loop cannot be broken by employing the stop function; CPU reset is required.

Similarly, CPU reset has to be invoked to terminate the condition that exists when an interruption is attempted with a prefix value designating a main-storage location that is not available to the CPU.

On some models, when an excessive number of consecutive interruptions is detected which cannot be broken by means of the stop function, the CPU enters a special state that can be exited only by use of CPU reset.

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.

Assigned Main-Storage Locations

Real Main Storage

The chart "Assigned Locations in Real Main Storage" shows the format and extent of the assigned locations in real main storage. In a multiprocessing system, real storage addresses are transformed to absolute addresses by means of prefixing. The locations are used as follows. Unless specifically noted, the usage applies to both the BC and EC modes.

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

8-15 *Restart Old PSW*: The current PSW is stored as the old PSW at locations 8-15 during the restart interruption.

24-63 *Interruption Old PSWs*: The current PSW is stored as the old PSW at locations 24-31, 32-39, 40-47, 48-55, and 56-63 during the external, supervisor-call, program, machine-check, and input/output interruptions, respectively.

64-71 *CSW*: The channel status word (CSW) is stored at locations 64-71 during an I/O interruption. It, or portions 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 timer is updated whenever the CPU is in the operating state. Depending on the resolution of the timer, the low-order locations may not be updated.

88-127 *Interruption New PSWs*: The new PSW is fetched from locations 88-95, 96-103, 104-111, 112-119, and 120-127 during the external, supervisor-call, program, machine-check, and input/output interruptions, respectively.

132-133 *Processor Address*: During an external interruption due to malfunction alert, emergency signal, or external call, the processor address associated with the source of the interruption is stored at locations 132-133. For all other external interruption conditions, zeros are stored at locations 132-133 when the old PSW specified EC mode, and the field remains unchanged when the old PSW specified the BC mode.

134-135 *External-Interruption Code*: During an external interruption in the EC mode, the interruption code is stored at locations 134-135.

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 *Translation-Exception Address*: During a program interruption due to a segment-translation exception or a page-translation exception, the translation-exception address is stored at locations 145-147, and zeros are stored at location 144. This field can be stored only when the old program PSW specifies the EC mode.

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. This field can be stored in either the BC or EC modes.

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. This field can be stored in either the BC or EC mode.

168-171 *Channel ID*: The four-byte channel-identification information is stored at locations 168-171 during the execution of STORE CHANNEL ID.

172-175 *IOEL Address*: The I/O-extended-logout address is fetched from locations 172-175 during the I/O-extended-logout operation.

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. It may be stored in either the BC or EC mode.

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.

216-511 *Machine-Check Interruption Code, Save Area, and Logout*: Information may be stored at locations 216-239 and 248-511 during a machine-check interruption, and information may be stored at locations 256-351 during an I/O interruption. Additionally, the contents of locations 256-351 may be changed at any time, subject to the asynchronous-fixed-logout-control bit in control register 14.

Absolute Main Storage

The chart "Assigned Locations in Absolute Main Storage" shows the format and extent of the assigned locations in absolute main storage. The locations are as follows, and the usage applies to both the BC and EC modes.

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 second 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 the third CCW of an IPL CCW chain after completion of the IPL initial read operation.

216-511 *Store-Status Save Area*: Information is stored at locations 216-231, 256-271, and 352-511 during the execution of the store-status operation.

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	Processor Address	External-Interruption Code		
88	136	0000000000000	ILC	0	Superv.-Call-Irptn. Code
8C	140	0000000000000	ILC	0	Program-Interruption Code
90	144	00000000	Translation-Exception Address		
94	148	00000000	Monitor Cl.#	PER C.	000000000000
98	152	00000000	PER Address		
9C	156	00000000	Monitor Code		
A0	160				
A4	164				
A8	168	Channel ID			
AC	172		IOEL Address		
B0	176	Limited Channel Logout			
B4	180				
B8	184		00000000	I/O Address	

Hex	Dec		
BC	188		
C0	192		
C4	196		
C8	200		
CC	204		
D0	208		
D4	212		
D8	216	Machine-Check CPU-Timer Save Area	
DC	220		
E0	224	Machine-Check 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	Region Code	
100	256	Fixed Logout Area	
104	260		
108	264		
10C	268		
154	340		
158	344		
15C	348		
160	352	Machine-Check Floating-Point Register Save Area	
164	356		
168	360		
16C	364		
170	368		
174	372		
178	376		
17C	380		
180	384	Machine-Check General-Register Save Area	
184	388		
188	392		
18C	396		
1B4	436		
1B8	440		
1BC	444		
1C0	448	Machine-Check Control-Register Save Area	
1C4	452		
1C8	456		
1CC	460		
1F4	500		
1F8	504		
1FC	508		

Assigned Locations in Real Main Storage

Hex	Dec	
0	0	Initial Program Loading PSW
4	4	
8	8	Initial Program Loading CCW1
C	12	
10	16	Initial Program Loading CCW2
14	20	
18	24	
1C	28	
20	32	
24	36	
28	40	
2C	44	
30	48	
34	52	
38	56	
3C	60	
40	64	
44	68	
48	72	
4C	76	
50	80	
54	84	
58	88	
5C	92	
60	96	
64	100	
68	104	
6C	108	
70	112	
74	116	
78	120	
7C	124	
80	128	
84	132	
88	136	
8C	140	
90	144	
94	148	
98	152	
9C	156	
A0	160	
A4	164	
A8	168	
AC	172	
B0	176	
B4	180	
B8	184	
BC	188	
C0	192	
C4	196	
C8	200	
CC	204	
D0	208	
D4	212	
D8	216	Store-Status CPU Timer Save Area
DC	220	
E0	224	Store-Status Clock-Comparator Save Area
E4	228	
E8	232	
EC	236	
F0	240	
F4	244	
F8	248	
FC	252	
100	256	Store-Status PSW Save Area
104	260	
108	264	Store-Status Prefix Save Area
10C	268	Store-Status Model-Dependent Feature Area
110	272	
		≈
158	344	
15C	348	
160	352	Store-Status Floating-Point Register Save Area
164	356	
168	360	
16C	364	
170	368	
174	372	
178	376	
17C	380	
180	384	Store-Status General-Register Save Area
184	388	
188	392	
18C	396	
		≈
1B4	436	
1B8	440	
1BC	444	
1C0	448	Store-Status Control-Register Save Area
1C4	452	
1C8	456	
1CC	460	
		≈
1F4	500	
1F8	504	
1FC	508	

Assigned Locations in Absolute Main Storage

Multiprocessing

Contents

Shared Main Storage 95
Prefixing 95
CPU Signaling and Response 97
Orders 97
Conditions Determining Response 98
Conditions Precluding Interpretation of the Order Code 98
Status Bits 99
TOD Clock Synchronization 101
CPU Address Identification 101

The multiprocessing feature provides for the interconnection of CPUs, via a common main storage, in order to enhance system availability and to share data and resources. The multiprocessing feature includes the following facilities:

- Shared main storage
- Prefixing
- CPU signaling and response
- TOD-clock synchronization

Associated with these facilities are four extensions to external interruption (external call, emergency signal, TOD clock sync check, and malfunction alert), which are described in the chapter "Interruptions"; control-register positions for the TOD-clock-sync-control bit and for the masks for the four external-interruption conditions, which are listed in "Control Registers" in the chapter "System Control"; and the instructions SET PREFIX, SIGNAL PROCESSOR, STORE CPU ADDRESS, and STORE PREFIX, which are described in the chapter "System-Control Instructions."

When the CPU is equipped with the multiprocessing feature, certain additional functions are provided as part of the system console. These functions pertain to the following controls, which are described in the chapter "System Console": configuration controls, enable-system-clear key, load key, system-reset key, and TOD-clock key.

Channels in a multiprocessing system are associated with a particular CPU. Only one CPU can initiate I/O operations at a channel, and all interruption conditions are directed to that CPU.

Shared Main Storage

The shared-main-storage facility permits more than one CPU to have access to common main-storage locations. All CPUs having access to a common main-storage location have access to the entire 2,048-byte block containing that location and to the associated key in storage. All CPUs refer to a shared-main-storage location using the same absolute address.

Prefixing

When the multiprocessing feature is installed in a CPU, most addresses associated with storage references by the CPU are processed by a mechanism called "prefixing." All addresses subject to this processing are referred to as "real" addresses. Storage addresses which are not subject to this processing, and all addresses that have been processed, whether or not they are changed, are referred to as "absolute" addresses.

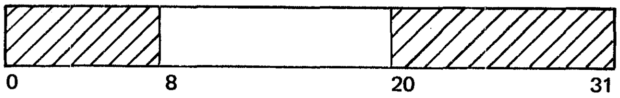
As a result of the processing to form the absolute address, real addresses 0-4095 are interchanged with the 4,096 addresses of the block that begins at the address identified in the prefix register. All other real addresses remain unchanged.

The real addresses 0-4095 include the addresses of the assigned storage locations that are implicitly generated by the CPU and channels, and include the addresses that can be specified by the program without the use of a base address or an index. Prefixing provides the ability to reassign this block of real locations for each CPU to a different block in absolute main storage, thus permitting more than one CPU sharing main storage to operate concurrently with a minimum of interference, especially in the processing of interruptions.

Because the prefixing mechanism *interchanges* the real addresses, each CPU can access all of absolute main storage, including the first 4,096 bytes and the assigned locations for another CPU.

The relationship between real and absolute addresses is graphically depicted in the figure "Relationship Between Real and Absolute Addresses."

The prefix is a 12-bit quantity located in the prefix register. The register has the following format:



The contents of the register can be set and inspected by the privileged instructions SET PREFIX and STORE PREFIX, respectively. On setting, bits corresponding to bit positions 0-7 and 20-31 of the prefix register are ignored. On storing, zeros are provided for these bit positions. The prefix register is initialized to zero.

Prefixing is applied to all references to main storage and to keys in storage, except for references by a CPU to the permanently assigned storage locations during performance of the store-status function, and except for references by a channel to extended-logout locations, to I/O data, to indirect-data-address words, and to CCWs. When dynamic address translation is specified, prefixing is applied after the address has been translated by the dynamic-address-translation mechanism. When installed, prefixing is always active and is not subject to any mode control.

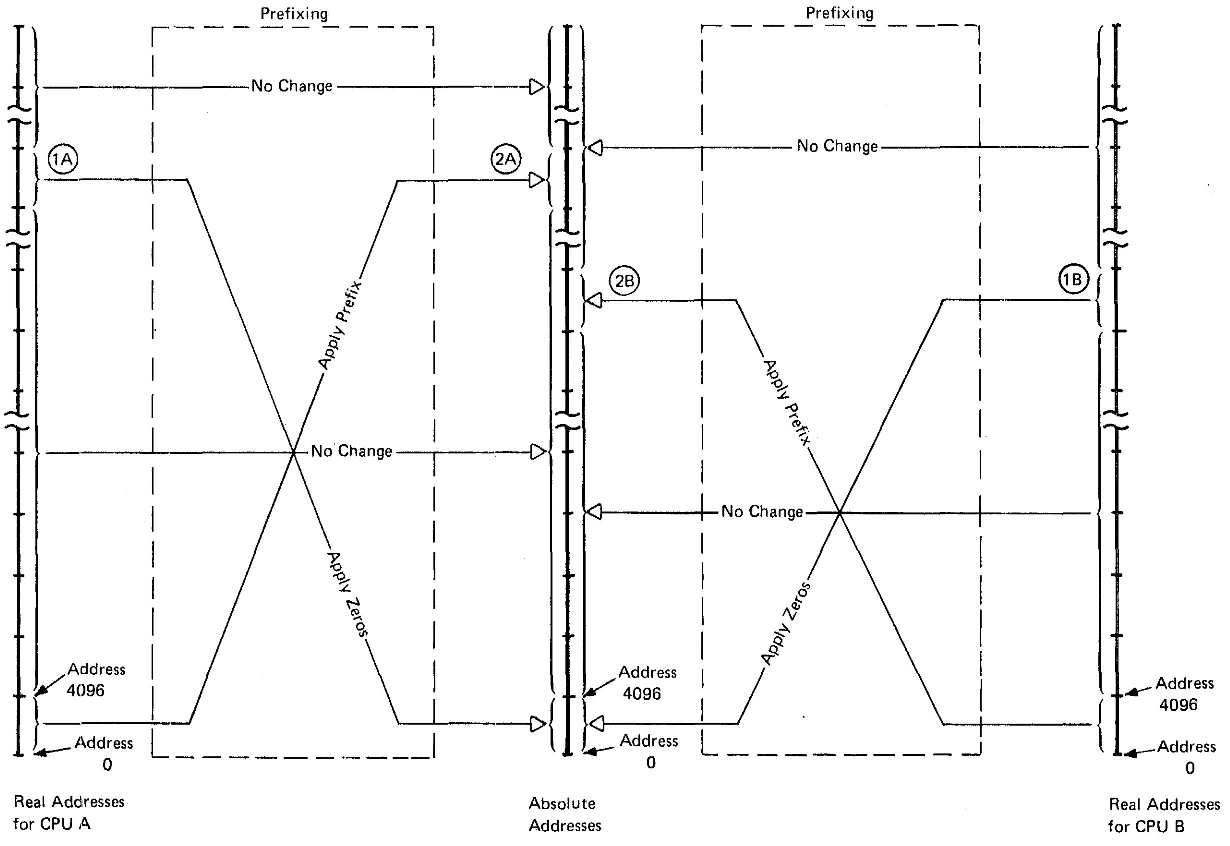
When prefixing is applied, the storage address is translated as follows:

1. Bits 8-19 of the storage address, if all zeros, are replaced with bits 8-19 of the prefix.
2. Bits 8-19 of the storage address, if equal to bits 8-19 of the prefix, are replaced with all zeros.
3. Bits 8-19 of the storage address, if not all zeros and not equal to bits 8-19 of the prefix, remain unchanged.

In all cases, bits 20-31 of the storage address remain unchanged.

Only the address presented to storage is translated by prefixing. The contents of the source of the address remain unchanged.

The distinction between real and absolute addresses is made even when prefixing is not installed



① Real addresses in which the high-order 12 bits are equal to the prefix for this CPU (A or B).

② Absolute addresses of the block that contains, for this CPU (A or B), the assigned locations in real storage.

Relationship Between Real and Absolute Addresses or when the prefix register contains all zeros. In both of these cases, a real address and its corresponding absolute address are identical.

CPU Signaling and Response

The CPU-signaling-and-response facility provides for communications among CPUs by means of the SIGNAL PROCESSOR instruction. It provides for transmitting and receiving the signal, decoding a set of assigned order codes, performing the specified operation, and responding to the signaling CPU.

If a CPU has the CPU-signaling-and-response facility installed, it can address the SIGNAL PROCESSOR instruction to itself. All orders are executed as defined.

Orders

Twelve orders are provided for communications among CPUs in a multiprocessing system. The orders are specified in bit positions 24-31 of the second-operand address of SIGNAL PROCESSOR and are encoded as follows:

Code	Order
00	Invalid and Unassigned
01	Sense
02	External Call
03	Emergency Signal
04	Start
05	Stop
06	Restart
07	Initial Program Reset
08	Program Reset
09	Stop and Store Status
0A	Initial Microprogram Load
0B	Initial CPU Reset
0C	CPU Reset
0D-FF	Invalid and Unassigned

The orders are defined as follows:

Sense: The addressed CPU presents its status to the issuing CPU (see "Status Bits" in this chapter for a definition of the bits). No other action is caused at the addressed CPU. The status, if not all zeros, is stored in the general register designated by the R_1 field, and condition code 1 is set; if all status bits are zero, condition code 0 is set.

External Call: An "external call" external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of the SIGNAL PROCESSOR instruction. The associated interruption occurs when the CPU is interruptible for that condition and does not necessarily occur during the execution of the SIGNAL PROCESSOR instruction. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. Only one external-call condition can be kept pending in a CPU at a time.

Emergency Signal: An "emergency-signal" external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of the SIGNAL PROCESSOR instruction. The associated interruption occurs when the CPU is interruptible for that condition and does not necessarily occur during the execution of the SIGNAL PROCESSOR instruction. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. At any one time the receiving CPU can keep pending one emergency-signal condition for each CPU of the multiprocessing system, including the receiving CPU itself.

Start: The addressed CPU is placed in the operating state (see "Stopped and Operating States" in the chapter "System Control"). The order is effective only when the addressed CPU is in the stopped state, and the effect is unpredictable when the stopped state has been entered by reset. The CPU does not necessarily enter the operating state during the execution of the SIGNAL PROCESSOR instruction.

Stop: The addressed CPU performs the stop function (see "Stopped and Operating States" in the chapter "System Control"). The CPU does not necessarily enter the stopped state during the execution of the SIGNAL PROCESSOR instruction. No action is caused at the addressed CPU if that CPU is in the stopped state when the order code is accepted.

Restart: The addressed CPU performs the restart function (see "Restart" in the chapter "Interruptions"). The CPU does not necessarily perform the function during the execution of the SIGNAL PROCESSOR instruction.

Initial Program Reset: The addressed CPU performs initial program reset (see "Resets" in the chapter "System Control"). The execution of the reset does not affect other CPUs and does not affect channels not configured to the CPU being reset. The reset operation is not necessarily completed during the execution of the SIGNAL PROCESSOR instruction.

Program Reset: The addressed CPU performs program reset (see "Resets" in the chapter "System Control"). The execution of the reset does not affect other CPUs and does not affect channels not configured to the CPU being reset. The reset operation is not necessarily completed during the execution of the SIGNAL PROCESSOR instruction.

Stop and Store Status: The addressed CPU performs the stop function, followed by the store-status function (see "Stopped and Operating States" and "Store Status" in the chapter "System Control"). The CPU does not necessarily complete the operation, or even enter the stopped state, during the execution of the SIGNAL PROCESSOR instruction.

Initial Microprogram Load (IMPL): The addressed CPU performs initial program reset and then initiates the initial-microprogram-load function. The latter function is the same as that which is performed as part of manual initial microprogram loading. If the initial-microprogram-load function is not provided on the addressed CPU, the order code is treated as unassigned and invalid. The operation is not necessarily completed during the execution of the SIGNAL PROCESSOR instruction.

Initial CPU Reset: The addressed CPU performs initial CPU reset (see "Resets" in the chapter "System Control"). The execution of the reset does not affect other CPUs and does not cause any channels, including those configured to the addressed CPU, to be reset. The reset operation is not necessarily completed during the execution of the SIGNAL PROCESSOR instruction.

CPU Reset: The addressed CPU performs CPU reset (see "Resets" in the chapter "System Control"). The execution of the reset does not affect other CPUs and does not cause any channels, including those configured to the addressed CPU, to be reset. The reset operation is not necessarily completed during the execution of the SIGNAL PROCESSOR instruction.

Conditions Determining Response

Conditions Precluding Interpretation of the Order Code

The following situations determine the initiation of the order. The sequence in which the situations are listed is the order of priority for indicating concurrently existing situations:

1. The access path to the addressed CPU is busy because a concurrently issued SIGNAL PROCESSOR instruction is using the CPU-signaling-and-response facility. The concurrently issued instruction may or may not have been issued by or to the addressed CPU and may or may not have been issued to this CPU. The order is rejected. Condition code 2 is set.
2. The addressed CPU is not operational, that is, the addressed CPU is not installed, is not configured to the issuing CPU, or is in certain customer-engineer test modes, or does not have its power on. The order is rejected. Condition code 3 is set.
3. One of the following conditions exists at the addressed CPU:

   a. A previously issued start, stop, restart, or stop-and-store-status order has been accepted by the addressed CPU, and execution of the order has not yet been completed.

   b. A manual start, stop, restart, or store-status function has been initiated at the addressed CPU, and the operation has not yet been completed.

   c. A manual initial-program-load function has been initiated at the addressed CPU, and the reset portion, but not the program load portion, of the operation has been completed.

   If the currently specified order is a sense, external call, emergency signal, start, stop, restart, or stop and store status, the order is rejected, and condition code 2 is set. If the currently specified order is an IMPL, one of the reset orders, or an unassigned or not-implemented order, the order code is interpreted as described in the section "Status Bits."
4. One of the following conditions exists at the addressed CPU:

   a. A previously issued initial-program-reset, program-reset, IMPL, initial-CPU-reset, or CPU-reset order has been accepted by the addressed CPU, and execution of the order has not yet been completed.

   b. A manual reset or IMPL function has been initiated at the addressed CPU, and the function has not yet been completed. The term "manual reset function" includes the reset portion of IPL.

   If the currently specified order is a sense, external call, emergency signal, start, stop, restart, or stop and store status, the order is rejected, and condition code 2 is set. If the currently specified order is an IMPL, one of the reset orders, or an unassigned or not-implemented order, either the order is rejected and condition code 2 is set or the order code is interpreted as described under the heading "Status Bits."

When any of the conditions described in 3 and 4 exists, the addressed CPU is referred to as "busy." Busy is not indicated if the addressed CPU is in the check-stop state or when the operator-intervening condition exists. A CPU-busy condition is normally of short duration; however, the conditions described in item 3 may last indefinitely because of an unending series of interruptions or because of an invalid address in the prefix register. In this situation, however, the CPU does not appear busy to any of the reset orders or to IMPL.

Status Bits

Eight status conditions are defined whereby the issuing and addressed CPUs can indicate their response to the designated order. The status conditions and their bit positions in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction are as follows:

Bit Position	Status Condition
0	Equipment check
1-23	Unassigned; zeros stored
24	External-call pending
25	Stopped
26	Operator intervening
27	Check stop
28	Not ready
29	Unassigned; zero stored
30	Invalid order
31	Receiver check

The status condition assigned to bit position 0 is generated by the CPU executing the SIGNAL PROCESSOR instruction. The status conditions assigned to bit positions 24-31 are generated by the addressed CPU.

When the access path to the addressed CPU is not busy and the addressed CPU is operational and does not indicate busy to the currently specified order, the addressed CPU presents its status to the issuing CPU. These status bits are of two types:

- Status bits 24-28 indicate the presence of the corresponding conditions in the addressed CPU at the time the order code is received. Except in response to the sense order, each condition is indicated only when the condition precludes the successful execution of the designated order. In the case of sense, all existing status conditions are indicated; the operator-intervening and not-ready conditions each are indicated if these conditions preclude the execution of any installed order.
- Status bits 30 and 31 indicate that the corresponding conditions were detected by the addressed CPU during reception of the order.

If the presented status is all zeros, the addressed CPU has accepted the order, and condition code 0 is set at the issuing CPU; if the presented status is not all zeros, the addressed CPU has rejected the order, the presented status is stored at the issuing CPU in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction, zeros are stored in bit positions 0-23 of the register, and condition code 1 is set.

When the equipment-check condition exists, bit 0 of the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction is set to one, bits 1-23 are set to zeros, and the contents of bit positions 24-31 are unpredictable. In this case, condition code 1 is set independently of whether the access path to the addressed CPU is busy and independently of whether the addressed CPU is not operational, is busy, or has presented zero status.

The status conditions are defined as follows:

Equipment Check: This condition exists when the CPU executing the instruction detects equipment malfunctioning that has affected only the execution of this instruction and the associated order. The order code may or may not have been transmitted, and may or may not have been accepted, and the status bits provided by the addressed processor may be in error.

External Call Pending: This condition exists when an external-call interruption condition is pending in the addressed CPU because of a previously issued SIGNAL PROCESSOR instruction. The condition exists from the time an external-call order is accepted until the resultant external interruption has been completed. The condition may be due to the issuing CPU or another CPU. The condition, when present, is indicated only in response to sense and to external call.

Stopped: This condition exists when the addressed CPU is in the stopped state. The condition, when present, is indicated only in response to sense.

Operator Intervening: This condition exists when the addressed CPU is executing certain operations initiated from the console or the remote operator control panel. The particular manually initiated operations that cause this condition to be present depend on the model and on the order specified. This condition, when present, can be indicated in response to all orders. Operator intervening is indicated in response to sense if the condition is present and precludes the acceptance of any of the installed orders. The condition may also be indicated in response to unassigned or uninstalled orders.

Check Stop: This condition exists when the addressed CPU is in the check-stop state. The condition, when present, is indicated only in response to sense, external call, emergency signal, start, stop, restart, and stop and store status. The condition may also be indicated in response to unassigned or uninstalled orders.

Not Ready: This condition exists when the addressed CPU uses reloadable control storage to perform an order and the required microprogram is not loaded. The not-ready condition may be indicated in response to all orders except IMPL.

Invalid Order: This condition exists during the communications associated with the execution of SIGNAL PROCESSOR when the addressed CPU decodes an unassigned or uninstalled order code.

Receiver Check: This condition exists when the addressed CPU detects malfunctioning of equipment during the communications associated with the execution of SIGNAL PROCESSOR. When this condition is indicated, the order has not been initiated and, since the malfunction may have affected the generation of the remaining receiver status bits, these bits are not necessarily valid. A machine-check condition may or may not have been generated at the addressed CPU.

The following chart summarizes which status conditions are presented to the issuing CPU in response to each order code.

If the presented status bits are all zeros, the order has been accepted, and the issuing processor sets condition code 0. If one or more ones are presented, the order has been rejected, and the issuing processor stores the status in the general register specified by the R_1 field of the SIGP instruction and sets condition code 1.

Programming Notes

A CPU can obtain the following functions by addressing SIGNAL PROCESSOR to itself:

1. *Sense* indicates whether an external-call condition is pending.

	External Call Pending	Stopped	Operator Intervening #	Check Stop	Not Ready	Invalid Order	Receiver Check ≠
Sense	X	X	X	X	X	0	X
External Call	X	0	X	X	X	0	X
Emergency Signal	0	0	X	X	X	0	X
Start	0	0	X	X	X	0	X
Stop	0	0	X	X	X	0	X
Restart	0	0	X	X	X	0	X
Initial Program Reset	0	0	X	0	X	0	X
Program Reset	0	0	X	0	X	0	X
Stop and Store Status	0	0	X	X	X	0	X
IMPL*	0	0	X	0	0	0	X
Initial CPU Reset*	0	0	X	0	X	0	X
CPU Reset*	0	0	X	0	X	0	X
Unassigned Order	0	0	X	0/X	X	1	X

Explanation:

0 A zero is presented in this bit position regardless of the current state of this condition.

1 A one is presented in this bit position.

X A zero or a one is presented in this bit position, reflecting the current state of the corresponding condition.

0/X Either a zero or the current state of the corresponding condition is indicated.

# The current state of the operator-intervening condition may depend on the order code that is being interpreted.

≠ If a one is presented in the receiver-check bit position, the values presented in the other bit positions are not necessarily valid.

* If the order code is implemented, use the line entry for the order code; if the order code is not implemented, use the line entry labeled "Unassigned Order."

2. *External call* and *emergency signal* cause the corresponding interruption conditions to be generated. *External call* can be rejected because of a previously generated external-call condition.
3. *Start* sets condition code 0 and has no other effect.
4. *Stop* causes the CPU to set condition code 0, take pending interruptions for which it is enabled, and enter the stopped state.
5. *Restart* provides a means to store the current PSW.

Two CPUs can simultaneously execute SIGNAL PROCESSOR instructions, with each CPU addressing the other. When this occurs, one CPU, but not both, can find the access path busy because of the transmission of the order code or status bits associated with the SIGNAL PROCESSOR instruction that is being executed by the other CPU. Alternatively, both CPUs can find the access path available and transmit the order codes to each other. In particular, two CPUs can simultaneously stop, restart, or reset each other.

TOD Clock Synchronization

In an installation with more than one CPU, depending on the model, each CPU may have a separate time-of-day clock, or more than one CPU may share a clock. In all cases, each CPU accesses a single clock. A configuration change does not affect the value in any of the clocks or which clock a CPU accesses.

When more than one time-of-day clock exists in a configured system, the stepping rates are synchronized such that all time-of-day clocks in the configuration are incremented at the exact same rate. The CPU timer in each CPU is also decremented at this same rate.

The TOD-clock-synchronization facility provides functions that make it possible to write a single model-independent supervisor clock-synchronization program which can handle systems with a single time-of-day clock or with multiple clocks. Additionally, the facility, in conjunction with the supervisor clock-synchronization program, provides, in effect, only one clock in a multiprocessing system, so that, to all programs storing the clock, it appears that all CPUs read the same clock.

The synchronization is provided by a mechanism which causes a stopped clock to start incrementing in response to a signal from another clock and which checks whether the low-order 32 bits of all clocks in the configuration are stepped at the same time. Lack of synchronization is signaled by an external interruption indicating the TOD-clock-sync-check condition. The synchronization is under control of the TOD-clock-sync-control bit in control register 0, bit position 2. See "Time-of-Day Clock Sync Check" in the chapter "Interruptions" and "Time-of-Day Clock" in the chapter "System Control."

Programming Note

A stopped clock, with the TOD-clock-sync-control bit set to one, starts when bits 32-63 of any running clock in the configuration are incremented to zero. This permits the program to synchronize all clocks to any particular clock without requiring special operator action to select a "master clock" as the source of the clock synchronization pulses. The supervisor clock-synchronization program must check for synchronization of high-order bits and assist in synchronizing clocks by communicating the high-order bit values and setting them in the clocks to be synchronized.

CPU Address Identification

Each CPU in a multiprocessor installation is assigned a unique address. The CPU is designated by specifying this address in the processor address field of a SIGNAL PROCESSOR instruction. The CPU signaling a malfunction alert, emergency signal, or external call is identified by storing this address in the processor-address field with the interruption. The CPU address is assigned during system installation and is not changed as a result of configuration changes. The program can determine the address of the CPU by means of the instruction STORE CPU ADDRESS.

System-Control Instructions

Contents

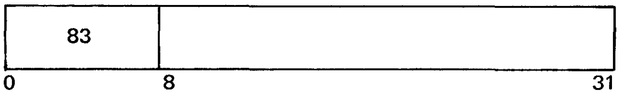
DIAGNOSE 103
INSERT PSW KEY 104
INSERT STORAGE KEY 105
LOAD CONTROL 105
LOAD PSW 105
LOAD REAL ADDRESS 106
PURGE TLB 107
READ DIRECT 107
RESET REFERENCE BIT 107
SET CLOCK 108
SET CLOCK COMPARATOR 108
SET CPU TIMER 109
SET PREFIX 109
SET PSW KEY FROM ADDRESS 109
SET STORAGE KEY 110
SET SYSTEM MASK 110
SIGNAL PROCESSOR 110
STORE CLOCK COMPARATOR 111
STORE CONTROL 111
STORE CPU ADDRESS 112
STORE CPU ID 112
STORE CPU TIMER 113
STORE PREFIX. 113
STORE THEN AND SYSTEM MASK 113
STORE THEN OR SYSTEM MASK 114
WRITE DIRECT 114

The system-control instructions include all privileged instructions that are described in this manual except the input/output instructions, which are described in the chapter "Input/Output Operations."

The system-control 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.

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/370 assembly language are shown with each instruction. For LOAD PSW, for example, LPSW is the mnemonic and $D_2(B_2)$ the operand designation.

Diagnose



The CPU performs built-in diagnostic functions, or other model-dependent functions. The purpose of the diagnostic functions is to verify proper functioning of CPU equipment and to locate faulty components. Other model-dependent functions may include disabling of failing buffers, reconfiguration of storage and channels, and modification of control storage.

Bits 8-31 may be used as in the SI or RS formats, or in some other way, to specify the particular diagnostic function. The use depends on the model.

The execution of the instruction may affect the state of the CPU and the contents of a register or storage location, as well as the progress of an I/O operation. Some diagnostic functions cause the test light of the system console to be turned on.

Condition Code: The code is unpredictable.

Program Exceptions:

Privileged Operation

Depending on the model, other exceptions may be recognized.

Programming Notes

Since the instruction is not intended for problem-program or supervisor-program use, DIAGNOSE has no mnemonic.

DIAGNOSE, unlike other instructions, does not

Name	Mnemonic	Characteristics									Code
DIAGNOSE					M	DM					83
INSERT PSW KEY	IPK	S		PK	M				R		B20B
INSERT STORAGE KEY	ISK	RR			M	A1	SP		R		09
LOAD CONTROL	LCTL	RS			M	A	SP				B7
LOAD PSW	LPSW	S	L		M	A	SP	$			82
LOAD REAL ADDRESS	LRA	RX	C	TR	M	A2			R		B1
PURGE TLB	PTLB	S		TR	M			$			B20D
READ DIRECT	RDD	SI		DC	M	A		$		ST	85
RESET REFERENCE BIT	RRB	S	C	TR	M	A1					B213
SET CLOCK	SCK	S	C		M	A	SP				B204
SET CLOCK COMPARATOR	SCKC	S		CK	M	A	SP				B206
SET CPU TIMER	SPT	S		CK	M	A	SP				B208
SET PREFIX	SPX	S		MP	M	A	SP	$			B210
SET PSW KEY FROM ADDRESS	SPKA	S		PK	M						B20A
SET STORAGE KEY	SSK	RR			M	A1	SP				08
SET SYSTEM MASK	SSM	S			M	A			SO		80
SIGNAL PROCESSOR	SIGP	RS	C	MP	M			$	R		AE
STORE CLOCK COMPARATOR	STCKC	S		CK	M	A	SP			ST	B207
STORE CONTROL	STCTL	RS			M	A	SP			ST	B6
STORE CPU ADDRESS	STAP	S		MP	M	A	SP			ST	B212
STORE CPU ID	STIDP	S			M	A	SP			ST	B202
STORE CPU TIMER	STPT	S		CK	M	A	SP			ST	B209
STORE PREFIX	STPX	S		MP	M	A	SP			ST	B211
STORE THEN AND SYSTEM MASK	STNSM	SI		TR	M	A				ST	AC
STORE THEN OR SYSTEM MASK	STOSM	SI		TR	M	A				ST	AD
WRITE DIRECT	WRD	SI		DC	M	A		$			84

Explanation:

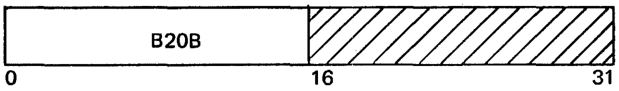
A	Access exceptions
A1	Addressing exceptions only
A2	Addressing and translation-specification exceptions only
C	Condition code is set
CK	CPU-timer and clock-comparator feature
DC	Direct-control feature
DM	Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code.
L	New condition code loaded
M	Privileged-operation exception
MP	Multiprocessing feature
PK	PSW-key-handling feature
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
ST	PER storage-alteration event
TR	Translation feature
$	Causes serialization

System-Control-Instruction Summary

follow the rule that programming errors are distinguished from equipment errors. Improper use of DIAGNOSE may result in false machine-check indications or may cause actual machine malfunctions to be ignored. It may also alter other aspects of system operation, including instruction execution and channel operation, to an extent that the operation does not comply with that specified in this manual. As a result of the improper use of DIAGNOSE, the system may be left in such a condition that the power-on reset or initial-microprogram-loading function must be performed.

Insert PSW Key

IPK [S]



The four-bit protection key of the current PSW is inserted into bit positions 24-27 of general register 2. Bits 0-23 of general register 2 remain unchanged, and bits 28-31 are set to zeros. Bits 16-31 of the instruction are ignored.

Condition Code: The code remains unchanged.

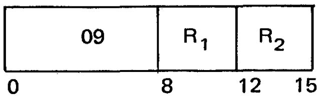
Program Exceptions:

Operation (if the PSW-key-handling feature is not installed)

Privileged operation

Insert Storage Key

ISK R_1,R_2 [RR]



The key in storage associated with the block that is addressed by the contents of the general register designated by the R_2 field is inserted in the general register designated by the R_1 field.

Bits 8-20 of the register designated by the R_2 field designate a block of 2,048 bytes in real main storage. 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 address designating the storage block, being a real address, is not subject to dynamic address translation. Hence, the reference to the key cannot cause segment-translation, page-translation, and translation-specification exceptions to be recognized, and an addressing exception can be caused only by an invalid storage-block address (as contrasted to an invalid address of a table entry). The reference to the key is not subject to a protection exception.

The execution of the instruction depends on the mode of operation. When the PSW specifies the extended-control mode, the complete seven-bit key is inserted into bit positions 24-30 of the register designated by the R_1 field, with bit 31 set to zero. When the PSW specifies the basic-control mode, bits 0-4 of the key are placed in bit positions 24-28 of the register, with bits 29-31 of the register set to zeros. The contents of bit positions 0-23 of the register remain unchanged.

Condition Code: The code remains unchanged.

Program Exceptions:

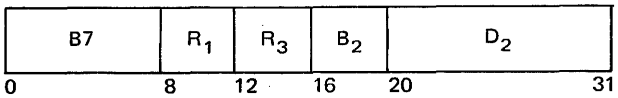
Privileged operation

Access (addressing for operand access only, operand 2)

Specification

Load Control

LCTL $R_1,R_3,D_2(B_2)$ [RS]



The set of control registers starting with the control register designated by the R_1 field and ending with the control register designated by the R_3 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_1 field and continuing up to and including the control register designated by the R_3 field, with control register 0 following control register 15. The second operand remains unchanged.

An attempt is made to fetch the operand from main storage for each of the designated control registers, regardless of whether the facility requiring the presence of the control register is installed. Whenever the storage reference causes an access exception, the exception is indicated.

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:

Privileged operation

Access (fetch, operand 2)

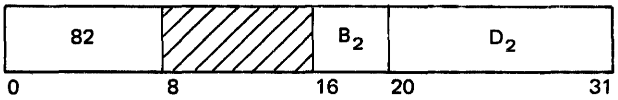
Specification

Programming Note

To ensure that presently written programs run when new facilities using additional control register positions are installed, only zeros should be loaded in unassigned control register positions.

Load PSW

LPSW $D_2(B_2)$ [S]



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 basic-control (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. CPU operation is delayed until all previous accesses by this CPU to main 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.

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 value specifies the EC mode and the EC facility is not installed, or when the value specifies the EC mode and the contents of bit positions 0, 2-4, 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 set as specified in the new PSW loaded.

Program Exceptions:

Privileged operation
Access (fetch, operand 2)
Specification

Load Real Address

LRA $R_1,D_2(X_2,B_2)$ [RX]

B1	R_1	X_2	B_2	D_2	
0	8	12	16	20	31

The real address corresponding to the second-operand address is inserted in the general register designated by the R_1 field. The remaining high-order bits of the register are set to zeros.

The logical address specified by the X_2, B_2, and D_2 fields is translated by means ofthe dynamic-address-translation facility, regardless of whether translation is specified in the PSW, and regardless of whether the PSW specifies the BC or EC mode. The translation is performed using the current contents of control registers 0 and 1, but without the use of the translation-lookaside buffer (TLB). The resultant 24-bit real address is inserted in bit positions 8-31 of the general register designated by the R_1 field, and bits 0-7 of the register are set to zeros. The translated address is not inspected for resolution, protection, or validity.

Condition code 0 is set when translation can be completed, that is, when the entry in each table lies within the specified table length and its I bit is zero.

When the I bit in the segment-table entry is one, condition code 1 is set, and the real address of the segment-table entry is placed in the register designated by the R_1 field. When the I bit in the page-table entry is one, condition code 2 is set, and the real address of the page-table entry is placed in the register designated by the R_1 field. When either the segment-table entry or the page-table entry is outside the table, condition code 3 is set, and the register designated by the R_1 field contains the real address of the entry that would have been referred to if the length violation did not occur. In all these cases, the 24-bit address is placed in bit positions 8-31 of the register, and the leftmost eight bits of the register are set to zeros.

An addressing exception is recognized when the address of the segment-table entry or page-table entry designates a location outside the available main storage of the installed system. A translation-specification exception is recognized when bits 8-12 of control register 0 contain an invalid code, or the segment-table entry or page-table entry has a format error. For all these cases, the operation is suppressed.

Resulting Condition Code:

0 Translation available
1 Segment-table entry invalid (I bit is one)
2 Page-table entry invalid (I bit is one)
3 Segment- or page-table length violation

Program Exceptions:

Operation (if the translation feature is not installed)

Privileged operation

Access (addressing for table-entry access and translation specification only, operand 2)

Purge TLB

PTLB [S]

B20D	▨
0	16 31

All information in the translation-lookaside buffer (TLB) of this CPU is made invalid. No change is made to the contents of addressable storage or registers.

The TLB appears cleared of its original contents for all following instructions. When the CPU does not have a TLB, the instruction is equivalent to a no-operation. The invalidation is not signaled to any other CPU.

A serialization function is performed. CPU operation is delayed until all previous accesses by this CPU to main storage have been completed, as observed by channels and other CPUs. No subsequent instructions, their operands, or dynamic-address-translation entries are fetched by this CPU until the execution of this instruction is complete.

Bits 16-31 of the instruction are ignored.

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the translation feature is not installed)
Privileged operation

Read Direct

RDD $D_1(B_1),I_2$ [SI]

85	I_2	B_1	D_1
0	8	16	20 31

The contents of the I_2 field are made available as signal-out timing signals. A direct-in data byte is accepted from an external device in the absence of a hold signal and is placed in the location designated by the operand address.

The contents of the I_2 field are made available on a set of eight signal-out lines as 0.5-microsecond to 1.0-microsecond timing signals. These signal-out lines are also used in WRITE DIRECT. On a ninth line (read out) a 0.5-microsecond to 1.0-microsecond timing signal is made available coincident with these timing signals. The read-out line is distinct from the write-out line in WRITE DIRECT. No checking bits are made available with the eight instruction bits.

Eight data bits are accepted from a set of eight direct-in lines when the hold signal on the hold-in line is absent. The hold signal is sampled after the read-out signal has been completed and should be absent for at least 0.5 microsecond. No checking bits are accepted with data signals, but a checking-block code is generated as the data is placed in storage. When the hold signal is not removed, the CPU does not complete the instruction.

A serialization function is performed before the signals are made available and again after the first-operand byte is placed in storage. CPU operation is delayed until all previous accesses by this CPU to main storage have been completed, as observed by channels and other CPUs, and then the signal-out timing signals are presented. No subsequent instructions or their operands are accessed by this CPU until the first operand byte has been placed in main storage, as observed by channels and other CPUs.

An excessively long instruction execution may result in incomplete updating of the interval timer.

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the direct-control feature is not installed)
Privileged operation
Access (store, operand 1)

Reset Reference Bit

RRB $D_2(B_2)$ [S]

B213	B_2	D_2
0	16	20 31

The reference bit is set to zero in the key in storage associated with the block that is designated by the second-operand address.

Bits 8-20 of the second-operand address designate a block of 2,048 bytes in real main storage. Bits 0-7 and 21-31 of the address are ignored.

The address designating the storage block, being a real address, is not subject to dynamic address translation. Hence, the reference to the key cannot cause segment-translation, page-translation, and translation-specification exceptions to be recognized, and an addressing exception can be caused only by an invalid storage-block address (as contrasted to an invalid address of a table entry). The reference to the key is not subject to a protection exception.

The value of the remaining bits of the key, including the change bit, is not affected.

The condition code is set to reflect the state of the reference and change bits before the reference bit is set to zero.

Resulting Condition Code:

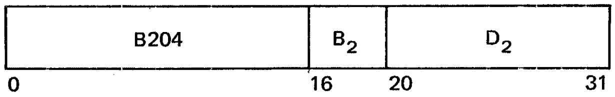
0 Reference bit zero, change bit zero
1 Reference bit zero, change bit one
2 Reference bit one, change bit zero
3 Reference bit one, change bit one

Program Exceptions:

Operation (if the translation feature is not installed)
Privileged operation
Access (addressing for operand access only, operand 2)

Set Clock

SCK $D_2(B_2)$ [S]



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 is placed in the stopped state.

The operand designated by the instruction is considered an unsigned, 64-bit, fixed-point number. This operand replaces the contents of the clock, as determined by the clock's resolution. Only those bits of the operand are set in the clock that correspond to the bit positions to be updated by the clock; the contents of the remaining rightmost bit positions are not preserved in the clock and are ignored.

After the clock value is set, the clock is placed in the stopped state. The clock leaves the stopped state to enter the set state and resume counting under control of the time-of-day clock synchronization control bit (control register 0, bit 2). When the bit is zero or the clock-synchronization facility is not installed, the clock enters the set state at the completion of the instruction. When the bit is one, the clock remains in the stopped state either until the bit is set to zero or until any other running time-of-day clock in the configured system is incremented to a value of all zeros in bit positions 32-63.

The value of the clock is changed, and the clock is placed in the stopped state only if the TOD-clock switch on the system console is in the enable-set position. If the switch is in the secure position, 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, regardless of the setting of the TOD-clock switch, the value and the state of the clock are not changed, 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 switch.

Resulting Condition Code:

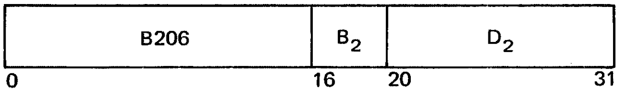
0 Clock value set
1 Clock value secure
2 -
3 Clock not operational

Program Exceptions:

Privileged operation
Access (fetch, operand 2)
Specification

Set Clock Comparator

SCKC $D_2(B_2)$ [S]



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 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:

Operation (if the clock comparator is not installed)
Privileged operation
Access (fetch, operand 2)
Specification

Set CPU Timer

SPT $D_2(B_2)$ [S]

B208	B_2	D_2
0	16	20 31

The current value of the CPU timer is replaced by the contents of the doubleword designated by the second-operand address.

Only those bits of the operand are set in the CPU timer that correspond to the bit positions to be updated; the contents of the remaining rightmost bit positions 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:

Operation (if the CPU timer is not installed)
Privileged operation
Access (fetch, operand 2)
Specification

Set Prefix

SPX $D_2(B_2)$ [S]

B210	B_2	D_2
0	16	20 31

The contents of the prefix register are replaced by the contents of bit positions 8-19 of the word at the location designated by the second-operand address. All information in the translation-lookaside buffer (TLB) of this CPU is made invalid.

If the operation is completed, the new prefix is used for any interruptions following the execution of the instruction and for the execution of subsequent instructions. The contents of bit positions 0-7 and 20-31 of the operand are ignored.

The TLB, if the CPU has one, appears cleared of its original contents for all following instructions.

A serialization function is performed. CPU operation is delayed until all previous accesses by this CPU to main storage have been completed, as observed by channels and other CPUs. No subsequent instructions, operands, or dynamic-address-translation entries are fetched by this CPU until the execution of this instruction is completed.

The operand must be designated on a word 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:

Operation (if the multiprocessing feature is not installed)
Privileged operation
Access (fetch, operand 2)
Specification

Set PSW Key From Address

SPKA $D_2(B_2)$ [S]

B20A	B_2	D_2
0	16	20 31

The four-bit protection key of the current PSW is replaced by bits 24-27 of the operand address.

The second-operand address is not used to address data; instead, bits 24-27 of the address form the new key. Bits 8-23 and 28-31 of the second-operand address are ignored.

Resulting Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the PSW-key-handling feature is not installed)
Privileged operation

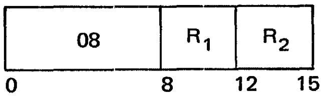
Programming Notes

The format of the SPKA instruction permits the program to set the protection key either from the general register designated by the B_2 field or from the D_2 field in the instruction itself.

When a problem program requests the supervisor program to access a location specified by the problem program, the SPKA instruction can be used by the supervisor program to verify that the problem program is authorized to make this access, provided the supervisor program is not protected against fetching. The supervisor program can perform the verification by replacing the PSW key of the supervisor program with the problem-program key before making the access and subsequently restoring the supervisor-program PSW key to its original value. Caution must be observed, however, in handling any resulting protection exceptions since such exceptions may cause the operation to be terminated and, on some models, the resulting interruption may be delayed and indicated with an instruction-length code of zero.

Set Storage Key

SSK R_1,R_2 [RR]



The key in storage associated with the block that is addressed by the contents of the register designated by the R_2 field is replaced by the contents of the register designated by the R_1 field.

Bits 8-20 of the register designated by the R_2 field designate a block of 2,048 bytes in real main storage. 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 address designating the storage block, being a real address, is not subject to dynamic address translation. Hence, the reference to the key cannot cause segment-translation, page-translation, and translation-specification exceptions to be recognized, and an addressing exception can be caused only by an invalid storage-block address (as contrasted to an invalid address of a table entry). The reference to the key is not subject to a protection exception.

The seven-bit key is obtained from bit positions 24-30 of the register designated by the R_1 field. The contents of bit positions 0-23 and 31 of the register are ignored. When dynamic address translation is not installed, bits 29 and 30 are ignored.

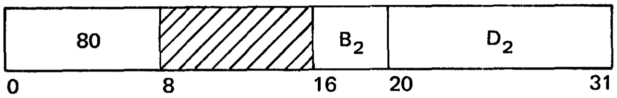
Condition Code: The code remains unchanged.

Program Exceptions:

Privileged operation
Access (addressing for operand access only, operand 2)
Specification

Set System Mask

SSM $D_2(B_2)$ [S]



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 facility is installed, the execution of the instruction is subject to the SSM-suppression bit, bit 1 of control register 0. When the bit is zero, the instruction is executed normally. When the bit 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 2-4 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.

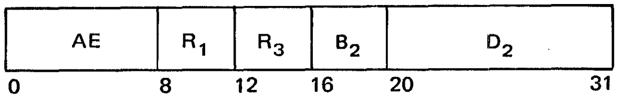
Condition Code: The code remains unchanged.

Program Exceptions:

Privileged operation
Access (fetch, operand 2)
Specification
Special operation

Signal Processor

SIGP R_1,R_3,$D_2(B_2)$ [RS]



An eight-bit order code is transmitted to the CPU designated by the processor address contained in the third operand. The result is indicated by the condition code and may be detailed by status assembled in the first-operand location.

The second-operand address is not used to address data; instead, bits 24-31 of the address contain the eight-bit order code. Bits 8-23 of the second-operand address are ignored. The order code specifies the function to be performed by the addressed

CPU. The assignment and definition of order codes appears in the chapter "Multiprocessing."

The 16-bit binary number contained in bit positions 16-31 of the general register designated by the R_3 field forms the processor address. The high-order 16 bits of the register are ignored.

A serialization function is performed. CPU operation is delayed until all previous accesses by this CPU to main storage have been completed, as observed by channels and other CPUs, and then the signaling occurs. No subsequent instructions or their operands are accessed by this CPU until the execution of the instruction is completed.

When the order code is accepted and no nonzero status is returned, condition code 0 is set. When status information is generated by this CPU or returned by the addressed CPU, the status is placed in the general register designated by the R_1 field, and condition code 1 is set.

When the access path to the addressed CPU is busy or the addressed CPU is operational and in a state where it cannot respond to the order code, condition code 2 is set.

When the addressed CPU is not operational (that is, it is not provided, or it is not configured to this CPU, or it is in certain customer-engineer test modes, or its power is off), condition code 3 is set.

A more detailed discussion of the condition-code settings for SIGNAL PROCESSOR is contained in the chapter "Multiprocessing."

Resulting Condition Code:

0 Order code accepted
1 Status stored
2 Busy
3 Not operational

Program Exceptions:

Operation (if the multiprocessing feature is not installed)
Privileged operation

Programming Notes

The execution time on the issuing CPU for SIGNAL PROCESSOR may vary depending on the model, the order code, and the state of the addressed CPU. In some cases, the execution time may be several seconds.

To ensure that presently written programs will be executed properly when new facilities using additional bits are installed, only zeros should appear in the unused bit positions of the second-operand address and in bit positions 0-15 of the register designated by the R_3 field.

Store Clock Comparator

STCKC $D_2(B_2)$ [S]

B207	B_2	D_2
0	16	20 31

The current value of the clock comparator is stored at the doubleword designated by the second-operand address.

Zeros are provided for the rightmost bit positions that are not used for comparison 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:

Operation (if the clock comparator is not installed)
Privileged operation
Access (store, operand 2)
Specification

Store Control

STCTL $R_1,R_3,D_2(B_2)$ [RS]

B6	R_1	R_3	B_2	D_2
0	8	12	16	20 31

The set of control registers starting with the control register designated by the R_1 field and ending with the control register designated by the R_3 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_1 field and continuing up to and including the control register designated by the R_3 field, with control register 0 following control register 15. The contents of the control registers remain unchanged.

An attempt is made to store each of the designated control registers, regardless of whether the facility requiring the presence of the control register is installed. Whenever the storage reference causes an access exception, the exception is indicated. The information provided for control register positions not associated with an installed facility 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:

Privileged operation

Access (store, operand 2)

Specification

Programming Note
Although on some CPUs STORE CONTROL may provide zeros in the bit positions corresponding to the unassigned register positions, the program should not depend on such zeros.

Store CPU Address

STAP $D_2(B_2)$ [S]

B212	B_2	D_2
0	16	20 31

The processor address by which this CPU is identified in a multiprocessing system is stored at the halfword location designated by the second-operand address.

The operand must be designated on a halfword 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:

Operation (if the multiprocessing feature is no:
installed)
Privileged operation
Access (store; operand 2)
Specification

Store CPU ID

STIDP $D_2(B_2)$ [S]

B202	B_2	D_2
0	16	20 31

Information identifying the CPU is stored at the doubleword location designated by the second-operand address.

The format of the information is as follows:

Version Code	CPU Identification Number
0	8 31

Model Number	Maximum MCEL Length
32	48 63

The version-code field, bit positions 0-7, contains model-dependent information, not otherwise easily obtained, that is normally of importance only in model-dependent recovery or diagnostic programs.

Bit positions 8-31 contain the CPU identification number, consisting of six digits: a high-order zero digit and five digits selected from the physical serial number stamped on the CPU, or six digits selected from the serial number. The contents of the CPU identification-number field, in conjunction with the model number, permit unique identification of the CPU.

Bit positions 32-47 contain the model number, consisting of four digits: a high-order zero digit and the three digits of the model number, such as 0145 or 0168.

Bit positions 48-63 contain a 16-bit binary value indicating the length in bytes of the longest machine-check extended logout (MCEL) that can be stored by the CPU.

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:
Privileged operation
Access (store, operand 2)
Specification

Programming Notes
The program should allow for the possibility that the CPU identification number may contain the digits A-F as well as the digits 0-9.

The principal uses of the information stored by the instruction STORE CPU ID are the following:

1. The CPU identification number, combined with the model number, provides a unique CPU identification that can be used in associating results with an individual system, particularly in regard to functional differences, performance differences, and error handling.

2. The model number, in conjunction with the version code, can be used by model-independent programs in determining which model-dependent recovery programs should be called.

3. The MCEL length can be used by model-independent programs to allocate main storage for the MCEL area.

Store CPU Timer

STPT $D_2(B_2)$ [S]

B209	B_2	D_2
0	16	20 31

The current value of the CPU timer is stored at the doubleword designated by the second-operand address.

Zeros are provided for the rightmost bit positions that are not updated by 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:

Operation (if the CPU timer is not installed)
Privileged operation
Access (store, operand 2)
Specification

Store Prefix

STPX $D_2(B_2)$ [S]

B211	B_2	D_2
0	16	20 31

The contents of the prefix register are stored at the word location designated by the second-operand address. Zeros are provided for bit positions 0-7 and 20-31.

The operand must be designated on a word 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:

Operation (if the multiprocessing feature is not installed)
Privileged operation
Access (store, operand 2)
Specification

Store Then AND System Mask

STNSM $D_1(B_1),I_2$ [SI]

AC	I_2	B_1	D_1
0	8	16	20 31

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 product (AND) of their original contents and the second operand.

The operation is suppressed on protection and addressing exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the translation feature is not installed)
Privileged operation

Programming Note
The STORE THEN AND SYSTEM MASK instruction permits the program to turn off selected bits in the system mask while retaining the original contents for later restoration. For example, in EC mode it

may be necessary that a program, which is not aware of the present status, disable program-event recording for a few instructions.

Store Then OR System Mask

STOSM $D_1(B_1),I_2$ [SI]

AD	I_2	B_1	D_1
0	8	16	20 31

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 sum (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 2-4 of the PSW are not all zeros. In this case, the instruction is completed, and the instruction-length code is set to 2.

The operation is suppressed on protection and addressing exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the translation feature is not installed)
Privileged operation
Access (store, operand 1)
Specification

Programming Note

The STORE THEN OR SYSTEM MASK instruction permits the program to turn on selected bits in the system mask while retaining the original contents for later restoration. For example, in EC mode the program may desire to enable the CPU for I/O interruptions and yet may not know the current status of the external-mask bit.

Write Direct

WRD $D_1(B_1),I_2$ [SI]

84	I_2	B_1	D_1
0	8	16	20 31

The byte at the location designated by the operand address is made available as a set of direct-out static signals. Eight instruction bits are made available as signal-out timing signals.

The eight data bits of the byte fetched from storage are presented on a set of eight direct-out lines as static signals. These signals remain until the next WRITE DIRECT is executed. No checking bits are presented with the eight data bits.

The contents of the I_2 field are made available simultaneously on a set of eight signal-out lines as 0.5-microsecond to 1.0-microsecond timing signals. On a ninth line (write out), a 0.5-microsecond to 1.0-microsecond timing signal is made available concurrently with these timing signals. The eight signal-out lines are also used in READ DIRECT. No checking bits are made available with the eight instruction bits.

A serialization function is performed before the operand is fetched and again after the signals have been presented. CPU operation is delayed until all previous accesses by this CPU to main storage have been completed, as observed by channels and other CPUs, and then the first operand byte is fetched and the signals made available. No subsequent instructions or their operands are fetched by this CPU until the signals have been made available.

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the direct-control feature is not installed)
Privileged operation
Access (fetch, operand 1)

General Instructions

Contents

Data Format 116
Number Representation 116
Instructions 117
ADD 117
ADD HALFWORD 117
ADD LOGICAL 120
AND 120
BRANCH AND LINK 121
BRANCH ON CONDITION 121
BRANCH ON COUNT 122
BRANCH ON INDEX HIGH 122
BRANCH ON INDEX LOW OR EQUAL 123
COMPARE 123
COMPARE AND SWAP 123
COMPARE DOUBLE AND SWAP 124
COMPARE HALFWORD 125
COMPARE LOGICAL 125
COMPARE LOGICAL CHARACTERS UNDER MASK 126
COMPARE LOGICAL LONG 126
CONVERT TO BINARY 127
CONVERT TO DECIMAL 128
DIVIDE 128
EXCLUSIVE OR 128
EXECUTE 129
INSERT CHARACTER 130
INSERT CHARACTERS UNDER MASK 130
LOAD 130
LOAD ADDRESS 131
LOAD AND TEST 131
LOAD COMPLEMENT 131
LOAD HALFWORD 131
LOAD MULTIPLE 132
LOAD NEGATIVE 132
LOAD POSITIVE 132
MONITOR CALL 132
MOVE 133
MOVE LONG 133
MOVE NUMERICS 135
MOVE WITH OFFSET 135
MOVE ZONES 136
MULTIPLY 136
MULTIPLY HALFWORD 136
OR 137
PACK 137
SET PROGRAM MASK 138
SHIFT LEFT DOUBLE 138
SHIFT LEFT DOUBLE LOGICAL 139
SHIFT LEFT SINGLE 139
SHIFT LEFT SINGLE LOGICAL 139
SHIFT RIGHT DOUBLE 140
SHIFT RIGHT DOUBLE LOGICAL 140
SHIFT RIGHT SINGLE 140
SHIFT RIGHT SINGLE LOGICAL 141
STORE 141
STORE CHARACTER 141
STORE CHARACTERS UNDER MASK 141
STORE CLOCK 141
STORE HALFWORD 142

Instructions (continued)

STORE MULTIPLE 142
SUBTRACT 143
SUBTRACT HALFWORD 143
SUBTRACT LOGICAL 143
SUPERVISOR CALL 144
TEST AND SET 144
TEST UNDER MASK 145
TRANSLATE 145
TRANSLATE AND TEST 145
UNPACK 146

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

Data Format

The general instructions treat data as being of four types: signed fixed-point numbers, unsigned fixed-point numbers, unstructured logical quantities, and decimal data. Data is treated as decimal by the conversion, packing, and unpacking instructions and is described in the chapter "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.

Number Representation

Fixed-point numbers are treated as signed or unsigned integers.

In an unsigned fixed-point number, all bits are used to express the absolute value of the number. When two unsigned fixed-point numbers are added, the shorter number is considered to be extended with high-order zeros.

For signed fixed-point numbers, the leftmost bit represents the sign, which is followed by the integer 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. The two's complement of a number is obtained by inverting each bit of the number 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 an operand must be extended with high-order bits, the expansion is achieved by setting the bits equal to the high-order bit of the operand.

The notation for signed fixed-point numbers 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 an all-one integer field with a sign bit of zero, whereas the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a sign bit of one.

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, a fixed-point overflow exception is recognized. An overflow does not result, however, when the maximum negative number is complemented and 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 fixed-point numbers in this publication, the expression "32-bit signed integer" denotes a 31-bit integer with a sign bit, and the expression "64-bit signed integer" denotes a 63-bit integer with 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 integer field changes the sign. However, in algebraic left-shifting the sign bit does not change even if significant high-order bits are shifted out.

Programming Note
The integer part of a signed fixed-point number may be considered to represent a positive value, with the sign representing a value of either zero or the maximum negative number.

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.

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designations for the IBM System/370 assembly language are shown with each instruction. For LOAD AND TEST, for example, LTR is the mnemonic and R_1, R_2 the operand designation.

Add

AR R_1,R_2 [RR]

1A	R_1	R_2
0	8	12 15

A R_1,D_2(X_2,B_2) [RX]

5A	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The second operand is added to the first operand, and the sum is placed in the first-operand location.

Addition is performed by adding all 32 bits of both operands. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The 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

Programming Note
In two's-complement notation a zero result is always positive.

Add Halfword

AH R_1,D_2(X_2,B_2) [RX]

4A	R_1	X_2	B_2	D_2
0	8	12	16	20 31

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 considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the addition by propagating the sign-bit value through the 16 high-order bit positions. The contents of the second operand in main storage remain unchanged.

Addition is performed by adding all 32 bits of both operands. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the sum is satisfactory; if they disagree, an overflow occurs. The sign bit is not changed after the overflow. A positive overflow yields a negative final sum, and a negative overflow results in a positive sum. The 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

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								$[1]	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	SW	A	SP				$		R	ST	BA
COMPARE DOUBLE AND SWAP	CDS	RS	C	SW	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									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
INSERT CHARACTER	IC	RX			A							R		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

General-Instruction Summary (Part 1 of 2)

Name	Mnemonic	Characteristics								Code
MOVE (immediate)	MVI	SI		A					ST	92
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
B	PER branch event
C	Condition code is set
D	Data exception
EX	Execute exception
IF	Fixed-point-overflow exception
II	Interruptible instruction
IK	Fixed-point-divide exception
L	New condition code loaded
MO	Monitor event
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
SP	Specification exception
SS	SS instruction format
ST	PER storage-alteration event
SW	Conditional-swapping feature
$	Causes serialization
$[1]	Causes serialization when the R_1 and R_2 fields contain all ones and zeros, respectively

General-Instruction Summary (Part 2 of 2)

Add Logical

ALR R_1,R_2 [RR]

1E	R_1	R_2

0 8 12 15

AL $R_1,D_2(X_2,B_2)$ [RX]

5E	R_1	X_2	B_2	D_2

0 8 12 16 20 31

The second operand is added to the first operand, and the sum is placed in the first-operand location. The occurrence of a carry out of the sign position is recorded in the condition code.

Logical addition is performed by adding all 32 bits of both operands without further change to the resulting sign bit. The instruction differs from ADD in the meaning of the condition code and in the absence of the interruption for overflow.

If a carry out of the sign position occurs, the leftmost bit of the condition code is made one. In the absence of a carry, the bit is made zero. When the sum is zero, the rightmost bit of the condition code is made zero. For a nonzero sum, the bit is made one.

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

NR R_1,R_2 [RR]

14	R_1	R_2

0 8 12 15

N $R_1,D_2(X_2,B_2)$ [RX]

54	R_1	X_2	B_2	D_2

0 8 12 16 20 31

NI $D_1(B_1),I_2$ [SI]

94	I_2	B_1	D_1

0 8 16 20 31

NC $D_1(L,B_1),D_2(B_2)$ [SS]

D4	L	B_1	D_1	B_2	D_2

0 8 16 20 32 36 47

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

Operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set to one if the corresponding bit positions in both operands contain a one; otherwise, the result bit is set to zero.

For NC, each operand field 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.

Resulting Condition Code:
0 Result is zero
1 Result not zero
2 -
3 -

Program Exceptions:

Access (fetch, operand 2, N and NC; fetch and store, operand 1, NI and NC)

Programming Note

The instruction AND may be used to set a bit to zero.

The execution of NI and NC consists in fetching a first-operand byte from main 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 shared location in main storage if the possibility exists that another CPU or a channel may also be updating the location. For NI, only one byte is stored.

Branch and Link

BALR R_1,R_2 [RR]

05	R_1	R_2
0	8	12 15

BAL $R_1,D_2(X_2,B_2)$ [RX]

45	R_1	X_2	B_2	D_2
0	8	12	16	20 31

Information from the current PSW, including the updated instruction address, is loaded as link information in the general register designated by R_1. 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, the contents of bit positions 8-31 of the general register designated by R_2 are used as the branch address. However, when the R_2 field contains zeros, the operation is performed without branching.

The branch address is computed before the link information is loaded. The link information, in both the BC and EC modes, consists of the instruction-length code, the condition code, the program mask bits, and the updated instruction address, arranged in the following format:

ILC	CC	Prog Mask	Instruction Address
0	2	4	8 31

The instruction-length code is 1 or 2.

Condition Code:
The code remains unchanged.

Program Exceptions:
None

Programming Notes
When the R_2 field in the RR format contains all zeros, the link information is loaded without branching. The format and the contents of the link information do not depend on whether the PSW specifies the BC or EC mode.

When BRANCH AND LINK is the subject instruction of EXECUTE, the instruction-length code is 2.

In both the BC and EC modes, the link information is in the format of the rightmost 32 bit positions of the BC-mode PSW.

Branch on Condition

BCR M_1,R_2 [RR]

07	M_1	R_2
0	8	12 15

BC $M_1,D_2(X_2,B_2)$ [RX]

47	M_1	X_2	B_2	D_2
0	8	12	16	20 31

The updated instruction address in the current PSW is replaced by the branch address if the state of the condition code is as specified by M_1; otherwise, normal instruction sequencing proceeds with the updated instruction address.

In the RX format the second-operand address is used as the branch address. In the RR format the contents of bit positions 8-31 of the general register specified by R_2 are used as the branch address. However, when the R_2 field contains zeros, the operation is performed without branching.

The M_1 field is used as a four-bit mask. The four bits of the mask correspond, left to right, with the four condition codes (0, 1, 2, and 3), as follows:

Instruction Bit	Mask Position Value	Condition Code
8	8	0
9	4	1
10	2	2
11	1	3

The branch is successful whenever the condition code has a corresponding mask bit of one.

When the M_1 and R_2 fields of BCR are 15 and 0, respectively, a serialization function is performed. CPU operation is delayed until all previous storage accesses by this CPU to main 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
When a branch is to be made on more than one condition code, 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 on condition codes 0 and 1.

When all four mask bits are zero or when the R_2 field in the RR format contains zero, the branch instruction is equivalent to a no-operation. When all four mask bits are ones, that is, the mask value is 15, the branch is unconditional unless the R_2 field in the RR format is zero.

Execution of BCR 15,0 may result in significant performance degradation, especially on larger models. 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_2, B_2, and D_2; in the RR format, the branch address is in the low-order 24 bits of the register specified by R_2. For operands, the address specified by X_2, B_2, and D_2 is the operand address, but the register specified by R_2 contains the operand itself.

Branch on Count

BCTR R_1,R_2 [RR]

06	R_1	R_2
0	8	12 15

BCT R_1,D_2(X_2,B_2) [RX]

46	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The contents of the general register specified by R_1 are algebraically reduced by one. 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_2 are used as the branch address. However, when the R_2 field contains zeros, the operation is performed without branching.

The branch address is computed before the counting operation. Counting does not change the condition code. The overflow occurring on transition from the maximum negative number to the maximum positive number is ignored. Otherwise, the subtraction proceeds as in fixed-point arithmetic, and all 32 bits of the general register participate in the operation.

Condition Code:
The code remains unchanged.

Program Exceptions:
None

Programming Notes

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^{31} results in the positive value of 2^{31}-1.

Counting is performed without branching when the R_2 field in the RR format contains zero.

Branch on Index High

BXH R_1,R_3,D_2(B_2) [RS]

86	R_1	R_3	B_2	D_2
0	8	12	16	20 31

An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first-operand location, regardless of whether the branch is taken. The second-operand address is used as the branch address.

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, instruction sequencing proceeds with the updated instruction address.

The first operand and the increment are in the registers specified by R_1 and R_3. The comparand register address is odd and is either one larger than R_3 or equal to R_3. The branch address is computed before the addition and comparison.

Overflow caused by the addition is ignored and does not affect the comparison. Otherwise, the addition and comparison proceed as in fixed-point arithmetic. All 32 bits of the general registers participate in the operations, and negative quantities are expressed in two's-complement notation. When the first operand and comparand locations coincide, the original register contents are used as the comparand.

Condition Code:
The code remains unchanged.

Program Exceptions:
None

Programming Note

The name "branch on index high" indicates that one of the major purposes of this instruction is the incrementing and testing of an index value. The increment may be algebraic and of any magnitude.

Branch on Index Low or Equal

BXLE $R_1,R_3,D_2(B_2)$ [RS]

87	R_1	R_3	B_2	D_2
0	8	12	16	20 31

An increment is added to the first operand, and the sum is compared algebraically with a comparand. Subsequently, the sum is placed in the first-operand location, regardless of whether the branch is taken. The second-operand address is used as the branch address.

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.

The first operand and the increment are in the registers specified by R_1 and R_3. The comparand register address is odd and is either one larger than R_3 or equal to R_3. The branch address is computed before the addition and comparison.

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.

Condition Code:
The code remains unchanged.

Program Exceptions:
None

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 determines the setting of the condition code.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

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

The first and second operands are compared. If they are equal, the third operand is stored in the second-operand location. If they are unequal, the second operand is loaded into the first-operand location.

The first and third operands are 32 bits in length, with each operand occupying a general register. The second operand is a word in main storage.

The result of the 32-bit comparison, either equal or unequal, is used to set the condition code. When the result of the comparison is unequal, 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 is permitted at the second-operand location 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.

A serialization function is performed before the operand is fetched, and, if condition code 0 is set, after the result is stored. CPU operation is delayed until all previous accesses by this CPU to main 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 main storage, as observed by channels and other CPUs.

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized, and the operation is suppressed.

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:

Operation (if the conditional-swapping feature is not installed)

Specification
Access (fetch and store, operand 2)

Programming Notes

The instruction COMPARE AND SWAP can be used by programs sharing common storage areas in either a multiprogramming or multiprocessing environment. The following are two examples:

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 COMPARE AND SWAP is executed with the R_1 field designating the register that contains the original value and the R_3 field designating the register that contains the updated value. If 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_1 field of the COMPARE AND SWAP 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.

COMPARE AND SWAP may 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 may 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 may use COMPARE AND SWAP to update the control word to indicate that the area is in use or to add messages to the list. The single program which has seized the area may also safely use COMPARE AND SWAP to remove messages from the list.

It should be noted that COMPARE AND SWAP does not interlock against storage accesses by channels. Therefore, the instruction should not be used to update a word, all or part of which is in an I/O input area, since the input data may be lost.

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 in the second-operand location. If they are unequal, the second operand is loaded into the first-operand location.

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 main storage.

The result of the 64-bit comparison, either equal or unequal, is used to set the condition code. When the result of the comparison is unequal, 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 is permitted at the second-operand location 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.

A serialization function is performed before the operand is fetched, and, if condition code 0 is set, after the result is stored. CPU operation is delayed until all previous accesses by this CPU to main 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 instruction is completed, including placing the result value, if any, in main storage, as observed by channels and other CPUs.

The R_1 and R_3 fields must each designate an even register, and the second operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized, and the operation is suppressed.

Resulting Condition Code:

0 First and second operands equal, second operand replaced by third operand
1 First and second operands not equal, first operand replaced by second operand
2 -
3 -

Program Exceptions:

Operation (if the conditional-swapping feature is not installed)
Specification
Access (fetch and store, operand 2)

Programming Note

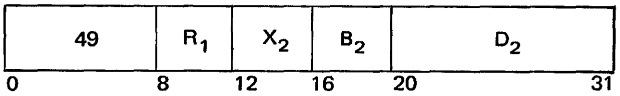
The instruction COMPARE DOUBLE AND SWAP may be used in a manner similar to that described in the programming notes for COMPARE AND SWAP.

In addition, it has another use. Consider a chained list, with a control word used to address the first message in the list, as described in the second example for COMPARE AND SWAP. If multiple programs are permitted to add and delete messages by using COMPARE AND SWAP, 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, is not aware that the list is changed. By increasing the size of the control word to a doubleword containing both the first message address and a word with a change number that is incremented for each modification of the list, and by using COMPARE DOUBLE AND SWAP to update both fields together, the possibility of the list being incorrectly updated is reduced to a negligible level. That is, an incorrect update can occur only if the first program is delayed while changes exactly equal in number to a multiple of 2^{32} take place *and* only if the last change places the original message address in the control word.

It should be noted that COMPARE DOUBLE AND SWAP does not interlock against storage accesses by channels. Therefore, the instruction should not be used to update a doubleword all or part of which is in an I/O input area, since the input data may be lost.

Compare Halfword

CH $R_1,D_2(X_2,B_2)$ [RX]



The first operand is compared with the second operand, and the result determines the setting of the condition code. The second operand is two bytes in length and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the comparison by propagating the sign-bit value through the 16 high-order bit positions.

Comparison is algebraic, treating both comparands as 32-bit signed integers. Operands in registers or storage are not changed.

Resulting Condition Code:

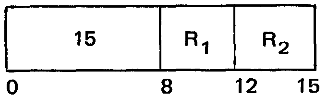
0 Operands are equal
1 First operand is low
2 First operand is high
3 -

Program Exceptions:

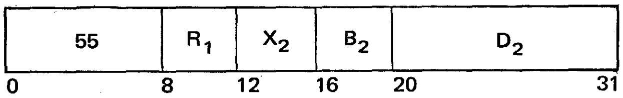
Access (fetch, operand 2)

Compare Logical

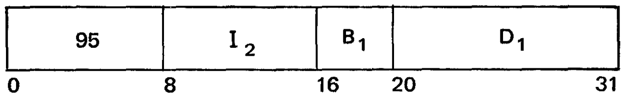
CLR R_1,R_2 [RR]



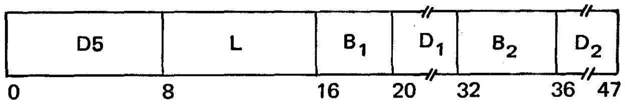
CL $R_1,D_2(X_2,B_2)$ [RX]



CLI $D_1(B_1),I_2$ [SI]



CLC $D_1(L,B_1),D_2(B_2)$ [SS]



The first operand is compared with the second operand, and the result is indicated in the condition code.

The comparison is performed with the operands considered as binary unsigned quantities, with all codes valid. The operation proceeds left to right and ends as soon as an inequality is found or an end of the fields is reached.

When part of an operand in CL or CLC is designated 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 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, CL and CLC; fetch, operand 1, CLI and CLC)

Programming Note

The COMPARE LOGICAL instruction treats all bits alike as part of an unsigned binary quantity. In variable-length operation, comparison may extend to field lengths of 256 bytes. The operation may be used to compare unsigned packed decimal fields or alphameric information in any code that has a collating sequence based on ascending or descending binary values. For example, EBCDIC has a collating sequence based on ascending binary values.

Compare Logical Characters Under Mask

CLM $R_1,M_3,D_2(B_2)$ [RS]

BD	R_1	M_3	B_2	D_2	
0	8	12	16	20	31

The second operand is compared with the first operand, under control of a mask, and the result is indicated in the condition code.

The contents of the M_3 field, bit positions 12-15, are used as a mask. The four bits of the mask, 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 is performed with the operands considered as binary unsigned quantities, with all codes valid. The operation proceeds left to right.

When the mask is not zero, exceptions associated with storage-operand access are recognized only for the number of bytes specified by the mask. However, when part of the designated storage operand is in an inaccessible location but the operation can be completed by using the accessible operand parts, it is unpredictable whether the exception for the inaccessible part is indicated. 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)

Compare Logical Long

CLCL R_1,R_2 [RR]

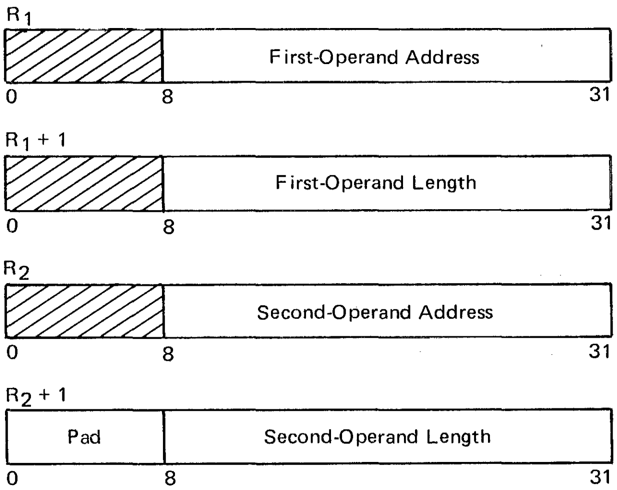
0F	R_1	R_2	
0	8	12	15

The first operand is compared with the second operand, and the result is indicated in the condition code. The shorter operand is considered extended with the padding character.

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 the general registers having addresses R_1+1 and R_2+1, respectively. Bit positions 0-7 of register R_2+1 contain the padding character. 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 is performed with the operands considered as binary unsigned quantities, with all codes valid. The comparison starts at the high-order end of both fields 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 the operands are not of the same length, the shorter operand is extended, for the purpose of comparison, with the padding character.

If both operands are of zero length, the operands are considered equal.

The execution of the instruction is interruptible. When an interruption occurs after a unit of operation other than the last one, 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 re-executed, resumes at the point of interruption. The high-order byte of registers R_1 and R_2 is set to zero; 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 count field pertaining to the shorter operand is zero, and its address is updated accordingly.

If the operation ends'because of a mismatch, the count and address fields at completion identify the byte of mismatch. The contents of bit positions 8-31 of registers R_1+1 and R_2+1 are decremented by the number of bytes that matched, unless the mismatch occurred with the padding character, in which case the count field for the shorter operand is set to zero. The contents of bit positions 8-31 of registers R_1 and R_2 are incremented by the amounts by which the corresponding count fields were reduced. If the two operands, including the padding character, if necessary, are equal, both count fields are made zero at completion, and the addresses are incremented by the corresponding count values. The contents of bit positions 0-7 of registers R_1 and R_2 are set to zero, including the case when one or both of the original count values are zero. The contents of bit positions 0-7 of registers R_1+1 and R_2+1 remain unchanged.

When part of an operand is designated in an inaccessible location, but the operation can be completed by using the available operand parts, it is unpredictable whether the access exception for the inaccessible part is recognized.

When the count field for an operand has the value zero, no access exceptions are recognized for that operand.

Resulting Condition Code:

0 Operands are equal, or both fields 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

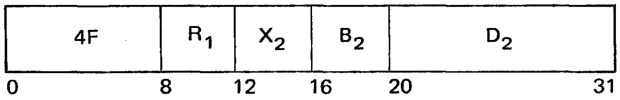
When the contents of the R_1 and R_2 fields are the same, condition code 0 is set, and protection and addressing exceptions are indicated when called for by the operand designation.

Special precautions should be taken when COMPARE LOGICAL LONG is made the subject of EXECUTE. See the programming notes under EXECUTE.

See also the programming notes under MOVE LONG.

Convert to Binary

CVB R_1,D_2(X_2,B_2) [RX]



The radix of the second operand is changed from decimal to binary, and the result is placed in the first-operand location. The number is treated as a right-aligned signed integer both before and after conversion.

The second operand has the packed decimal data format and is checked for valid sign and digit codes. Improper codes are a data exception and cause a program interruption. The decimal operand occupies eight bytes in storage. The low-order four bits of the field represent the sign. The remaining 60 bits contain 15 binary-coded-decimal digits in true notation. The packed decimal data format is described under "Decimal Instructions."

The result of the conversion is placed in the general register specified by R_1. The maximum number that can be converted and still be contained in a 32-bit register is 2,147,483,647; the minimum number is -2,147,483,648. For any decimal number outside this range, the operation is completed by placing the 32 low-order binary bits in the register; a fixed-point divide exception exists, and a program interruption follows. In the case of a negative second operand, the low-order part is in two's-complement notation.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)
Data
Fixed-Point Divide

Convert to Decimal

CVD R_1,D_2(X_2,B_2) [RX]

4E	R_1	X_2	B_2	D_2

0 8 12 16 20 31

The radix of the first operand is changed from binary to decimal, and the result is stored in the second-operand location. The number is treated as a right-aligned signed integer both before and after conversion.

The result is placed in the storage location designated by the second operand and has the packed decimal format, as described in "Decimal Instructions." The result occupies eight bytes in storage. The low-order four bits of the field represent the sign. A positive sign is encoded as 1100; a negative sign is encoded as 1101. The remaining 60 bits contain 15 binary-coded-decimal digits in true notation.

The number to be converted is obtained as a 32-bit signed integer from a general register. Since 15 decimal digits are available for the decimal equivalent of 31 bits, an overflow cannot occur.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

Divide

DR R_1,R_2 [RR]

1D	R_1	R_2

0 8 12 15

D R_1,D_2(X_2,B_2) [RX]

5D	R_1	X_2	B_2	D_2

0 8 12 16 20 31

The dividend (first operand) is divided by the divisor (second operand) and replaced by the remainder and the quotient.

The dividend is a 64-bit signed integer and occupies the even-odd pair of registers specified by the R_1 field of the instruction. A specification exception occurs when R_1 is odd. A 32-bit signed remainder and a 32-bit signed quotient replace the dividend in the even-numbered and odd-numbered registers, respectively. The divisor is a 32-bit signed integer.

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 relative magnitude of dividend and divisor is such that the quotient cannot be expressed by a 32-bit signed integer, a fixed-point divide exception is recognized (a program interruption occurs, no division takes place, and the dividend remains unchanged in the general registers).

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2 of D only)
Specification
Fixed-Point Divide

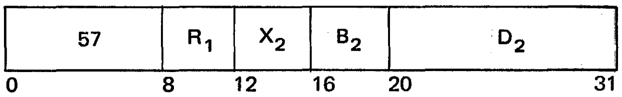
Exclusive OR

XR R_1,R_2 [RR]

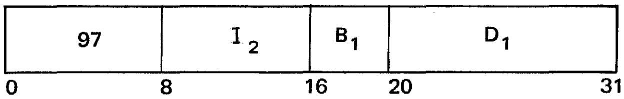
17	R_1	R_2

0 8 12 15

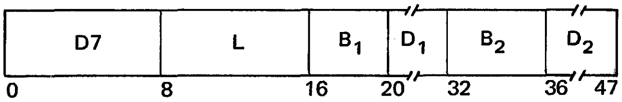
X $R_1,D_2(X_2,B_2)$ [RX]



XI $D_1(B_1),I_2$ [SI]



XC $D_1(L,B_1),D_2(B_2)$ [SS]



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

Operands are treated as unstructured logical quantities, and the connective EXCLUSIVE OR is applied 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 field 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.

Resulting Condition Code:

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

Program Exceptions:

Access (fetch, operand 2, X and XC; fetch and store, operand 1, XI and XC)

Programming Note

The instruction EXCLUSIVE OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.

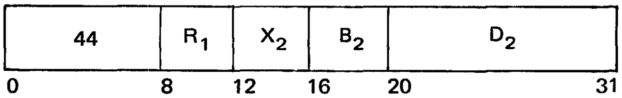
A field EXCLUSIVE-ORed with itself becomes all zeros.

The sequence A EXCLUSIVE-ORed B, B EXCLUSIVE-ORed A, A EXCLUSIVE-ORed B results in the exchange of the contents of A and B without the use of an auxiliary buffer area.

The execution of XI and XC consists in fetching a first-operand byte from main 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 shared location in main storage if the possibility exists that another CPU or a channel may also be updating the location. For XI, only one byte is stored.

Execute

EX $R_1,D_2(X_2,B_2)$ [RX]



The single instruction at the second-operand address is modified by the contents of the general register specified by R_1, and the resulting subject instruction is executed.

Bits 8-15 of the instruction designated by the branch address are ORed with bits 24-31 of the register specified by R_1, except when register 0 is specified, which indicates that no modification takes place. The subject instruction may be two, four, or six bytes in length. The ORing does not change either the contents of the register specified by R_1 or the instruction in storage, and it is effective only for the interpretation of the instruction to be executed.

The execution and exception handling of the subject instruction are exactly as if the subject 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 length code (2) of EXECUTE are used as part of the link information when the subject instruction is BRANCH AND LINK. When the subject instruction is a successful branching instruction, the updated instruction address of the current PSW is replaced by the branch address specified by the subject instruction.

When the subject instruction is in turn an EXECUTE, an execute exception is recognized, and the operation is suppressed. The effective address of EXECUTE must be even; otherwise, a specification exception is recognized.

Condition Code:

The code may be set by the subject instruction.

Program Exceptions:

Execute
Access (fetch, operand 2)
Specification

Programming Notes

The ORing of eight bits from the general register with the designated instruction permits indirect length, index, mask, immediate data, and arithmetic-register specification.

If the subject instruction is a successful branch, the length code still stands at 2.

An addressing or specification exception may be caused by EXECUTE or by the subject instruction.

When an interruptible instruction is made a subject of EXECUTE, the program normally should not designate any register updated by the interruptible instruction as either the R_1, X_2, or B_2 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 for a resumption of the execution.

Insert Character

IC R_1,D_2(X_2,B_2) [RX]

43	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The byte at the second-operand location is inserted into bit positions 24-31 of the general register designated by the R_1 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

ICM R_1,M_3,D_2(B_2) [RS]

BF	R_1	M_3	B_2	D_2
0	8	12	16	20 31

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_3 field, bit positions 12-15, are used as a mask. The four bits of the mask, 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 filled, in the order of ascending byte numbers, with bytes from the storage operand. Bytes are fetched from contiguous storage locations beginning at the second-operand address. 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 zero, the condition code is made 0. When all inserted bits are not zero, the code is set according to the leftmost bit of the storage operand: if this bit is one, the code is made 1 to indicate a negative algebraic value; if this bit is zero, the code is made 2, reflecting a positive algebraic value.

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 First bit of the inserted field is one
2 First bit of the inserted field is zero, and not all inserted bits are zeros
3 -

Program Exceptions:

Access (fetch, operand 2)

Programming Note

The condition code for INSERT CHARACTERS UNDER MASK is defined such that when the mask is 1111, the instruction causes the same condition code to be set as for LOAD AND TEST.

Load

LR R_1,R_2 [RR]

18	R_1	R_2
0	8	12 15

L R_1,D_2(X_2,B_2) [RX]

58	R_1	X_2	B_2	D_2
0	8	12	16	20 31

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)

Load Address

LA R_1,D_2(X_2,B_2) [RX]

41	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The address specified by the X_2, B_2, and D_2 fields is inserted in bit positions 8-31 of the general register specified by the R_1 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 Note
The same general register may be specified by the R_1, X_2, and B_2 instruction field, except that general register 0 can be specified only by the R_1 field. In this manner it is possible to increment the low-order 24 bits of a general register, other than 0, by the contents of the D_2 field of the instruction. The register to be incremented should be specified by R_1 and by either X_2 (with B_2 set to zero) or B_2 (with X_2 set to zero).

Load and Test

LTR R_1,R_2 [RR]

12	R_1	R_2
0	8	12 15

The second operand is placed unchanged in the first-operand location, and the sign and magnitude of the second operand determine 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_1 and R_2 fields designate the same register, the operation is equivalent to a test without data movement.

Load Complement

LCR R_1,R_2 [RR]

13	R_1	R_2
0	8	12 15

The two's complement of the second operand is placed in the first-operand location.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. 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 Note
Zero remains unchanged by complementation.

Load Halfword

LH R_1,D_2(X_2,B_2) [RX]

48	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The second operand is placed in the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits by propagating the sign-bit value through the 16 high-order bit positions. Expansion occurs after the operand is obtained from storage and before insertion in the register.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (fetch, operand 2)

Load Multiple

LM R_1,R_3,D_2(B_2) [RS]

98	R_1	R_3	B_2	D_2
0	8	12	16	20 31

The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is loaded from the locations designated by the second-operand address.

The storage area from which the contents of the general registers are obtained starts at the location designated by the second-operand address and continues through as many locations as needed. The general registers are loaded in the ascending order of their addresses, starting with the register 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 (fetch, operand 2)

Programming Note
All combinations of register addresses specified by R_1 and R_3 are valid. When the register addresses are equal, only one word is transmitted. When the address specified by R_3 is less than the address specified by R_1, the register addresses wrap around from 15 to 0.

Load Negative

LNR R_1,R_2 [RR]

11	R_1	R_2
0	8	12 15

The two's complement of the absolute value of the second operand is placed in the first-operand location.

The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged with positive sign.

Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 -
3 -

Program Exceptions:
None.

Load Positive

LPR R_1,R_2 [RR]

10	R_1	R_2
0	8	12 15

The absolute value of the second operand is placed in the first-operand location.

The operation includes complementation of negative numbers; positive numbers remain unchanged.

An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged. The 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

Monitor Call

MC D_1(B_1),I_2 [SI]

AF	I_2	B_1	D_1
0	8	16	20 31

A program interruption is caused if the appropriate monitor-mask bit in control register 8 is one.

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 program interruption for monitoring occurs. The contents of the I_2 field are stored at location 149 of main storage, with zeros stored at location 148. Bit 9 of the program interruption code is set to one.

The address specified by the B_1 and D_1 fields forms the monitor code, which is placed at locations

157-159. Address computation follows the rules of address arithmetic. The address is not inspected for access exceptions. Zeros are stored at location 156.

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, and the operation is suppressed.

Condition Code:
The code remains unchanged.

Program Exceptions:
Specification
Monitoring

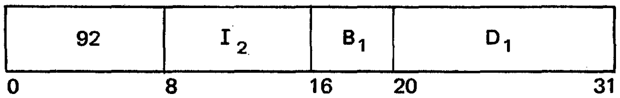
Programming Notes
The monitoring function is useful in performing various measurement functions; specifically, by implanting MONITOR CALL instructions within the code, tracing information can be generated indicating which programs were executed, counting information can be generated indicating how often particular programs are used, and timing information can be generated indicating how long a particular program required for execution.

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_1 field, each monitoring interruption can be identified by a 24-bit code.

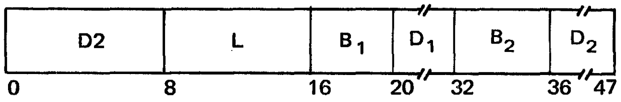
The monitor masks provide a means of disallowing all interruptions due to MONITOR CALL or allowing monitoring for all or selected classes.

Move

MVI $D_1(B_1),I_2$ [SI]



MVC $D_1(L,B_1),D_2(B_2)$ [SS]



The second operand is placed in the first-operand location.

For MVC, each operand field 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.

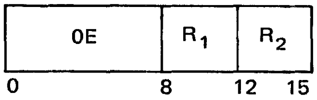
Program Exceptions:

Access (fetch, operand 2 of MVC; store, operand 1, MVI and MVC)

Programming Note
It is possible to propagate one character through an entire field by having the first-operand field start one character to the right of the second-operand field.

Move Long

MVCL R_1,R_2 [RR]

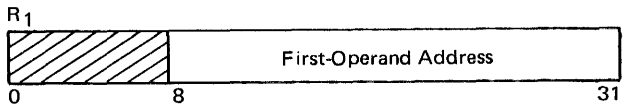


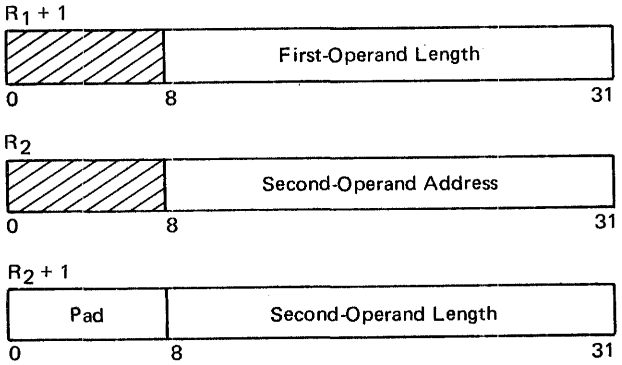
The second operand is placed in the first-operand location, provided overlapping of operand locations does not affect the final contents of the first-operand location. The remaining low-order byte positions, if any, of the first-operand location are filled with the padding character.

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 having addresses R_1+1 and R_2+1, respectively. Bit positions 0-7 of register R_2+1 contain the padding character. 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 movement starts at the high-order 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_1+1 have been moved into the first-operand location. If the second operand is shorter than the first operand, the remaining low-order bytes of the first operand are filled with the padding character.

As part of the execution of the instruction, the values of the two count fields are compared for the setting of the condition code, and a check is made for destructive overlap of the operands. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it, assuming the inspection for overlap is performed by the use of logical operand addresses. When the operands overlap destructively, no movement takes place, and condition code 3 is set.

Depending on whether the second operand wraps around from location 16,777,215 to location 0, movement takes place in the following cases:

1. When the second operand does not wrap around, movement is performed when the high-order byte of the first operand coincides with or is to the left of the high-order byte of the second operand, *or* if the high-order byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.
2. When the second operand wraps around, movement is performed when the high-order byte of the first operand coincides with or is to the left of the high-order byte of the second operand, *and* if the high-order 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 counts.

When the count specified by bit positions 8-31 of register R_1+1 is zero, no movement takes place, and the condition code is set to 0 or 1 to indicate the relative values of the counts.

The execution of the instruction is interruptible. When an interruption occurs, the contents of registers R_1+1 and R_2+1 are decremented by the number of bytes moved, and the contents of register R_1 and R_2 are incremented by the same number, so that the instruction, when re-executed, resumes at the point of interruption. The high-order byte of registers R_1 and R_2 is set to zero; 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 count field in register R_2+1 is zero, 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 main 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 count in register R_1+1 is zero, and the address in register R_1 is incremented by the original value of the count in register R_1+1. The count 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 contents of bit positions 0-7 of registers R_1 and R_2 are set to zero, including the case when one or both of the original count values are zero 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 the count specified by bit positions 8-31 of register R_1+1 is zero, or condition code 3 is set, no exceptions associated with operand access are recognized. When the count specified by bit positions 8-31 of register R_2+1 is zero, no access exceptions for the second-operand location are recognized. Similarly, when the second operand is larger 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.

Resulting Condition Code:

0 First-operand and second-operand counts are equal
1 First-operand count is low
2 First-operand count is high
3 No movement performed because of destructive overlap

Program Exceptions:

Access (fetch, operand 2; store, operand 1)
Specification

Programming Notes
The instruction MOVE LONG can be used for clearing storage. Clearing can be accomplished by setting the padding character to zero and the second operand count to zero.

When the first-operand count 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.

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, and protection and addressing exceptions are indicated when called for by the operand designation.

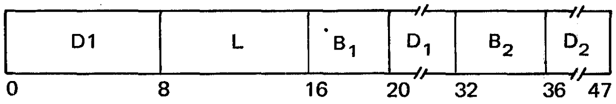
Since the execution of MOVE LONG is interruptible, the instruction cannot be used for situations where the program must rely on uninterrupted execution of the instruction or on the interval timer not being updated during the execution of the instruction. Similarly, the program should normally not let 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 if the instruction is refetched without an interruption.

Special precautions must be taken if MOVE LONG is made the subject of EXECUTE. See the programming notes under EXECUTE.

When the stop key is activated during the execution of MOVE LONG or COMPARE LOGICAL LONG, the CPU enters the stopped state at the completion of the execution of the next unit of operation. Similarly, in the instruction-step mode, only a unit of operation is performed. The amount of data processed in a unit of operation depends on the model and may depend on the particular condition that causes the execution of the instruction to be interrupted.

Move Numerics

MVN $D_1(L,B_1),D_2(B_2)$ [SS]



The low-order four bits of each byte in the second-operand field, the numerics, are placed in the low-order bit positions of the corresponding bytes in the first-operand field. The high-order four bits of each byte in the first-operand field remain unchanged.

Each operand field 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.

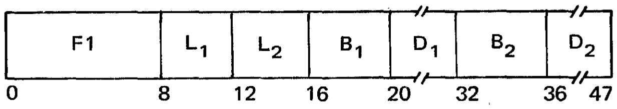
Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)

Programming Note
The execution of MVN consists in fetching the low-order four bits of each byte in the first-operand field, 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

MVO $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The second operand is placed to the left of and adjacent to the low-order four bits of the first operand.

The low-order four bits of the first operand are attached as low-order 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 first-operand and second-operand bytes are not checked for valid codes.

The result is obtained as if the fields were processed right to left. If necessary, the second operand is extended with high-order zeros. If the first-operand field is too short to contain all bytes of the second operand, the remaining information is ignored.

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 high-order 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 Note
In the execution of MVO, the fetch and subsequent store accesses to the low-order byte of the first operand do not necessarily occur one immediately after the other.

Move Zones

MVZ $D_1(L,B_1),D_2(B_2)$ [SS]

D3	L	B_1	D_1	B_2	D_2
0	8	16	20	32	36 47

The high-order four bits of each byte in the second-operand field (the zones) are placed in the high-order four bit positions of the corresponding bytes in the first-operand field. The low-order four bits of each byte in the first-operand field remain unchanged.

Each operand field 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 Note
The execution of MVZ consists in fetching the high-order four bits of each byte in the first-operand field, 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

MR R_1,R_2 [RR]

1C	R_1	R_2
0	8	12 15

M $R_1,D_2(X_2,B_2)$ [RX]

5C	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

Both multiplier and multiplicand are 32-bit signed integers. The product is always a 64-bit signed integer and occupies an even-odd register pair. Because the multiplicand is replaced by the product, the R_1 field of the instruction must refer to an even-numbered register. A specification exception occurs when R_1 is odd. The multiplicand is taken from the odd register of the pair. The contents of the even-numbered register replaced by the product are ignored, unless the register contains the multiplier. 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 Note
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. Since two's-complement notation is used, the sign bit is extended right until the first significant product digit is encountered.

Multiply Halfword

MH $R_1,D_2(X_2,B_2)$ [RX]

4C	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The product of the multiplier (second operand) and multiplicand (first operand) replaces the multiplicand. The second operand is two bytes in length and is considered to be a 16-bit signed integer.

Both multiplicand and product are 32-bit signed integers and may be located in any general register. The 16-bit multiplier is expanded to 32 bits before multiplication by propagating the sign-bit value through the 16 high-order bit positions. The multiplicand is replaced by the low-order part of the 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.

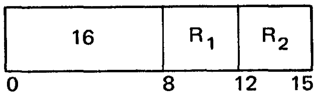
Program Exceptions:

Access (fetch, operand 2)

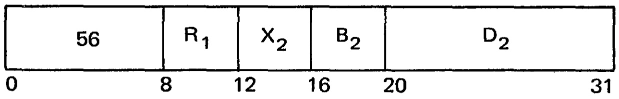
Programming Note
The significant part of the product usually occupies 46 bits or fewer, the exception being 47 bits when both operands are maximum negative. Since the low-order 32 bits of the product are stored unchanged, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow.

OR

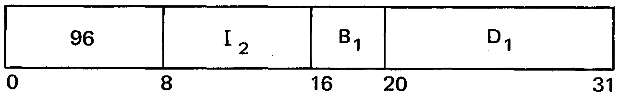
OR R_1,R_2 [RR]



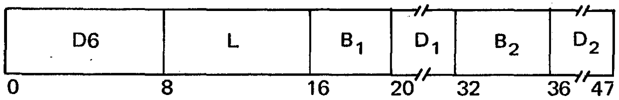
O $R_1,D_2(X_2,B_2)$ [RX]



OI $D_1(B_1),I_2$ [SI]



OC $D_1(L,B_1),D_2(B_2)$ [SS]



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

Operands are treated as unstructured logical quantities, and the connective OR is applied 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 field 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.

Resulting Condition Code:
0 Result is zero
1 Result not zero
2 -
3 -

Program Exceptions:

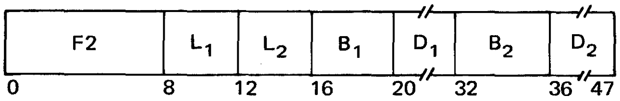
Access (fetch, operand 2, O and OC; fetch and store, operand 1, OI and OC)

Programming Note
The instruction OR may be used to set a bit to one.

The execution of OI and OC consists in fetching a first-operand byte from main 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 shared location in main storage if the possibility exists that another CPU may also be updating the location. For OI, only one byte is stored.

Pack

PACK $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



The format of the second operand is changed from zoned to packed, and the result is placed in the first-operand location.

The second operand is assumed to have the zoned format. All zones are ignored, except the zone over the low-order digit, which is assumed to represent a sign. The sign is placed in the right four bits of the low-order 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 field and are not checked for valid codes.

The result is obtained as if the fields were processed right to left. If necessary, the second operand is extended with high-order zeros. If the first-operand field is too short to contain all significant digits of the second-operand field, the remaining high-order digits are ignored.

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)

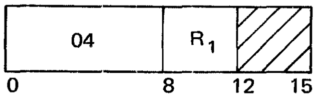
Programming Notes

The PACK instruction may be used to interchange the two hex digits in one byte by specifying a zero in the L_1 and L_2 fields and the same address for both operands.

To remove the zones of all bytes of a field, including the low-order byte, both operands must be extended with a dummy byte in the low-order position, which subsequently is ignored in the result field.

Set Program Mask

SPM R_1 [RR]



Bits 2-7 of the general register specified by the R_1 field replace the condition code and the program mask bits of the current PSW. Bits 12-15 of the instruction are ignored.

Bits 0, 1, and 8-31 of the register specified by the R_1 field are ignored. The contents of the register specified by the R_1 field remain unchanged.

The instruction permits setting of the condition code and the mask bits in either the problem or supervisor state.

Condition Code:
The code is set according to bits 2 and 3 of the register specified by R_1.

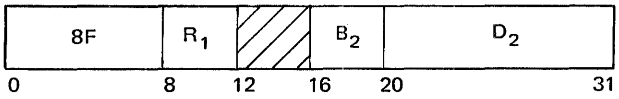
Program Exceptions:
None

Programming Note

Bits 2-7 of the general register may have been loaded from the PSW by BRANCH AND LINK.

Shift Left Double

SLDA R_1,D_2(B_2) [RS]



The double-length integer 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_1 field of the instruction specifies an even-odd pair of 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 number with 63 integer bits and a sign in the sign position of the even register. The sign remains unchanged. The high-order position of the odd register contains an integer bit, and the contents of the odd register participate in the shift in the same manner as the other integer bits. Zeros are supplied to the vacated positions of the registers.

If a bit unlike the sign bit is 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:

Specification
Fixed-Point Overflow

Programming Notes

The eight shift instructions provide the following three pairs of alternatives: left or right, single or double, and algebraic or logical. The algebraic shifts differ from the logical shifts in that, in the algebraic shifts, overflow is recognized, the condition code is set, and the high-order bit participates as a sign.

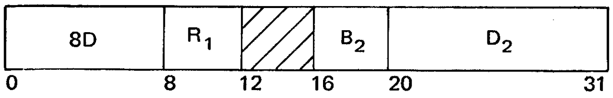
The maximum shift amount which can be specified is 63. For algebraic shifts this is sufficient to shift out the entire integer field. Since 64 bits participate in the double-logical shifts, the entire register contents cannot be shifted out.

A zero shift amount in the two algebraic double-shift operations provides a double-length sign and magnitude test.

The base register participating in the generation of the second-operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

Shift Left Double Logical

SLDL $R_1,D_2(B_2)$ [RS]



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_1 field of the instruction specifies an even-odd pair of 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.

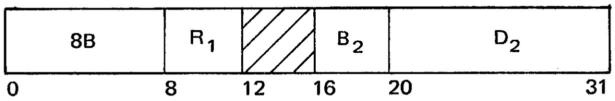
All 64 bits of the first operand participate in the shift. High-order bits are shifted out of the even-numbered register without inspection and are lost. Zeros are supplied to the vacated positions of the registers.

Condition Code:
The code remains unchanged.

Program Exceptions:
Specification

Shift Left Single

SLA $R_1,D_2(B_2)$ [RS]



The integer 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 sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the left shift. Zeros are supplied to the vacated low-order register positions.

If a bit unlike the sign bit is shifted out of 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

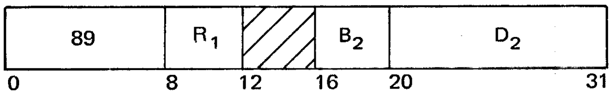
Programming Note

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.

Shift amounts from 31-63 cause the entire integer to be shifted out of the register. When the entire integer field for a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -2^{31}.

Shift Left Single Logical

SLL $R_1,D_2(B_2)$ [RS]



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. High-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated low-order register positions.

Condition Code:
The code remains unchanged.

Program Exceptions:
None

Shift Right Double

SRDA $R_1,D_2(B_2)$ [RS]

8E	R_1	▨	B_2	D_2
0	8	12	16	20 31

The double-length integer 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 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 number with 63 integer bits and a sign in the sign position of the even register. The sign remains unchanged. The high-order position of the odd register contains an integer bit, and the contents of the odd register participate in the shift in the same manner as the other integer bits. The low-order bits are shifted out without inspection and are lost. Bits equal to the sign are supplied to the vacated positions of the registers.

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

SRDL $R_1,D_2(B_2)$ [RS]

8C	R_1	▨	B_2	D_2
0	8	12	16	20 31

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 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. Low-order bits are shifted out of the odd-numbered register without inspection and are lost. Zeros are supplied to the vacated positions of the registers.

Condition Code:
The code remains unchanged.

Program Exceptions:
Specification

Shift Right Single

SRA $R_1,D_2(B_2)$ [RS]

8A	R_1	▨	B_2	D_2
0	8	12	16	20 31

The integer 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 sign of the first operand remains unchanged. All 31 integer bits of the operand participate in the right shift. Bits equal to the sign are supplied to the vacated high-order bit positions. Low-order bits are shifted out without inspection and are lost.

Resulting Condition Code:
0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 -

Program Exceptions:
None

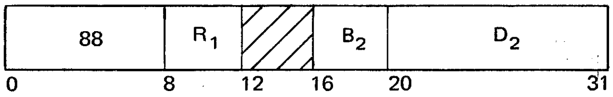
Programming Note

A right shift of one bit position is equivalent to division by two with rounding downward. When an even number is shifted right one position, the value of the field is that obtained by dividing the value by 2. When an odd number is shifted right one position, the value of the field is that obtained by dividing the *next lower* number by two. For example, +5 shifted right by one bit position yields +2, whereas -5 yields -3.

Shift amounts from 31-63 cause the entire integer to be shifted out of the register. When the entire integer field of a positive number has been shifted out, the register contains a value of zero. For a negative number, the register contains a value of -1.

Shift Right Single Logical

SRL $R_1,D_2(B_2)$ [RS]



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.

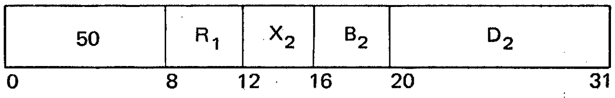
All 32 bits of the first operand participate in the shift. Low-order bits are shifted out without inspection and are lost. Zeros are supplied to the vacated high-order register positions.

Condition Code:
The code remains unchanged.

Program Exceptions:
None

Store

ST $R_1,D_2(X_2,B_2)$ [RX]



The first operand is stored at the second-operand location.

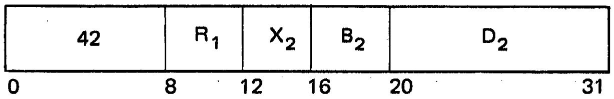
The 32 bits in the general register are placed unchanged at the second-operand location.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

Store Character

STC $R_1,D_2(X_2,B_2)$ [RX]



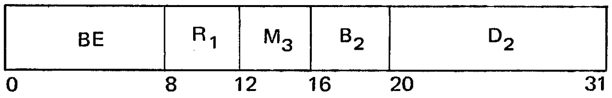
The contents of bit positions 24-31 of the general register designated by the R_1 field are placed unchanged at the second-operand location. The second operand is one byte in length.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

Store Characters Under Mask

STCM $R_1,M_3,D_2(B_2)$ [RS]



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, bit positions 12-15, are used as a mask. The four bits of the mask, 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 with the location designated by the second-operand address. The number of bytes stored is equal to the number of ones in the mask. The contents of the general register remain unchanged.

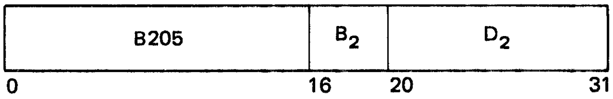
When the mask is not zero, exceptions associated with storage-operand access are recognized only for the number of bytes specified by the mask. When the mask is zero, no access exceptions are recognized.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

Store Clock

STCK $D_2(B_2)$ [S]



The current value of the time-of-day clock is stored at the eight-byte field designated by the second-operand address, provided the clock is in the set, stopped, or not-set state.

The value of the clock is expressed as an unsigned, 64-bit fixed-point number. Zeros are stored for the low-order bit positions that are not provided by the clock.

When the clock is in the error state, the value stored is unpredictable. When the clock is in the not-operational state, zeros are stored at the operand location.

The quality of the clock value stored by the instruction is indicated by the resultant condition code setting.

A serialization function is performed before the value of the clock is fetched and again after the value is placed in main storage. CPU operation is delayed until all previous accesses by this CPU to main storage have been completed, as observed by channels and other CPUs, and then the value of the clock is fetched. No subsequent instructions or their operands are fetched by this CPU until the clock value has been placed in main storage, as observed by channels and CPUs.

Resulting Condition Code:
0 Clock in set state
1 Clock in not-set state
2 Clock in error state
3 Clock in stopped state or not-operational state

Program Exceptions:
Access (store, operand 2)

Programming Notes
Condition code 0 normally indicates that the clock has been set by the control program. Accordingly, the value may be used in elapsed-time measurements and as a valid time-of-day and calendar indication. Condition code 1 indicates that the clock's value is the elapsed time since the power for the clock was turned on. In this case the value may be used in elapsed time measurements but is not a valid time-of-day indication. Condition codes 2 and 3 mean that the value provided by STORE CLOCK cannot be used for time measurement or indication.

Condition code 3 indicates that the clock is either in the stopped state or not-operational state. These two states can normally be distinguished since an all-zero value is stored when in the not-operational state.

Bit position 31 of the clock is incremented every 1.048576 seconds; hence, for timing applications involving human responses, the high-order clock word may provide sufficient resolution.

To provide compatible operation from one system to another requires the establishment of a standard time origin, or epoch, that is, the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 A.M. Greenwich Mean Time is recommended as the standard epoch for the clock, although some early support of the TOD clock is not based on this epoch. A program using the clock's value as a time-of-day and calendar indication may have to be aware of the support under which it is running. With the standard epoch, bit 0 of the TOD clock turns on May 11, 1971 at 11:56:53.685248 A.M. GMT. Therefore, in most cases, the program can test the high-order bit to determine if the TOD clock value is the standard epoch.

Because of the inaccuracies in setting the clock value on the basis of a synchronization signal provided by the operator, the low-order bit positions of the clock, expressing fractions of seconds, normally are not valid as indications of time of day. However, they permit elapsed time measurements of high resolution.

Store Halfword

STH $R_1,D_2(X_2,B_2)$ [RX]

40	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The contents of bit positions 16-31 of the general register designated by the R_1 field are placed unchanged at the second-operand location. The second operand is two bytes in length.

Condition Code:
The code remains unchanged.

Program Exceptions:
Access (store, operand 2)

Store Multiple

STM $R_1,R_3,D_2(B_2)$ [RS]

90	R_1	R_3	B_2	D_2
0	8	12	16	20 31

The set of general registers starting with the register specified by R_1 and ending with the register specified by R_3 is stored at the locations designated by the second-operand address.

The storage area where the contents of the general registers are placed starts at the location designated by the second-operand address and continues through as many locations as needed. The general registers are stored in the ascending order of their addresses, starting with the register 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)

Subtract

SR R_1,R_2 [RR]

1B	R_1	R_2
0	8	12 15

S $R_1,D_2(X_2,B_2)$ [RX]

5B	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The second operand is subtracted from the first operand, and the difference is placed in the first-operand location.

Subtraction is considered to be performed by adding the one's complement of the second operand and a low-order one to the first operand. All 32 bits of both operands participate, as in ADD. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow occurs. The 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 Note

The use of the one's complement and the low-order one instead of the two's complement of the second operand is necessary for proper recognition of overflow when subtracting the maximum negative number.

When, in the RR format, the R_1 and R_2 fields designate the same register, subtracting is equivalent to clearing the register.

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]

4B	R_1	X_2	B_2	D_2
0	8	12	16	20 31

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 considered to be a 16-bit signed integer.

The second operand is expanded to 32 bits before the subtraction by propagating the sign-bit value through the 16 high-order bit positions.

Subtraction is considered to be performed by adding the one's complement of the expanded second operand and a low-order one to the first operand. All 32 bits of both operands participate, as in ADD. If the carry out of the sign-bit position and the carry out of the high-order numeric bit position agree, the difference is satisfactory; if they disagree, an overflow occurs. The 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]

1F	R_1	R_2
0	8	12 15

SL $R_1,D_2(X_2,B_2)$ [RX]

5F	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The second operand is subtracted from the first operand, and the difference is placed in the first-operand location. The occurrence of a carry out of the sign position is recorded in the condition code.

Logical subtraction is considered to be performed by adding the one's complement of the second operand and a low-order one to the first operand. All 32 bits of both operands participate, without further change to the resulting sign bit. The instruction differs from SUBTRACT in the meaning of the condition code and in the absence of the interruption for overflow.

If a carry out of the sign position occurs, the leftmost bit of the condition code is made one. In the absence of a carry, the bit is made zero. When the sum is zero, the rightmost bit of the condition code is made zero. For a nonzero sum, the bit is made one.

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:

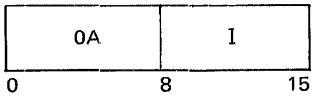
Access (fetch, operand 2 of SL only)

Programming Note

The use of the one's complement and the low-order one instead of the two's complement of the second operand results in the recognition of carry when subtracting zero or the maximum negative number. A zero difference cannot be obtained without a carry out of the sign position.

Supervisor Call

SVC [RR]



The instruction causes a supervisor-call interruption, with the I field of the instruction providing the interruption code.

The contents of bit positions 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.

Condition Code:

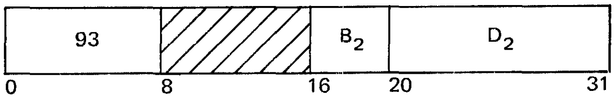
The code remains unchanged in the old PSW.

Program Exceptions:

None

Test and Set

TS $D_2(B_2)$ [S]



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 entire addressed 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 is permitted to this location 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 main 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 main storage, as observed by channels and other CPUs.

Resulting Condition Code:

0 Leftmost bit of byte specified is zero
1 Leftmost bit of byte specified is one
2 -
3 -

Program Exceptions:

Access (fetch and store, operand 2)

Programming Note

TEST AND SET can 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's testing the byte before the first program is able to set it.

It should be noted that TEST AND SET does not interlock against storage accesses by channels.

Test Under Mask

TM $D_1(B_1),I_2$ [SI]

91	I_2	B_1	D_1
0	8	16	20 31

The state of the first-operand bits selected by a mask is used to set 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 character in storage specified 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, the condition code is made 0. The code is also made 0 when the mask is all zeros. When the selected bits are all ones, the code is made 3; otherwise, the code is made 1. The character in storage is not changed.

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)

Translate

TR $D_1(L,B_1),D_2(B_2)$ [SS]

DC	L	B_1	D_1	B_2	D_2
0	8	16	20	32	36 47

The eight-bit bytes of the first operand are used as arguments to reference the list designated by the second-operand address. Each eight-bit function byte selected from the list replaces the corresponding argument in the first operand.

The L field applies only to 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 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.

Condition Code:

The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes

The instruction TRANSLATE may be used to convert data from one code to another code.

Another purpose for which the instruction may be used is 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.

Because the 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.

The fetch and subsequent store accesses to a particular byte in the first-operand field do not necessarily occur one immediately after the other.

Translate and Test

TRT $D_1(L,B_1),D_2(B_2)$ [SS]

DD	L	B_1	D_1	B_2	D_2
0	8	16	20	32	36 47

The eight-bit bytes of the first operand are used as arguments to reference the list designated by the second-operand address.

The L field applies only to the first operand.

Each eight-bit function byte thus selected from the list is used to determine the continuation of the operation. When the function byte is a zero, the operation proceeds by fetching and translating the next argument byte. When the function byte is nonzero, the operation is completed by inserting the related argument address in general register 1 and by inserting the function byte in general register 2.

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 the all-zero combination.

When the function byte is zero, the operation proceeds with the next operand byte. 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 low-order 24 bits of general register 1. This address points to the argument last translated. The high-order eight bits of register 1 remain unchanged. The function byte is inserted in the low-order eight bits of general register 2. Bits 0-23 of register 2 remain unchanged.

Condition code 1 is set when one or more argument bytes remain to be translated. Condition code 2 is set if the last function byte is the only nonzero byte.

Resulting Condition Code:

0 All function bytes are zero
1 Nonzero function byte before the first operand field is exhausted
2 The last function byte is the only nonzero byte
3 -

Program Exceptions:

Access (fetch, operands 1 and 2)

Programming Note

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

UNPK D1(L1,B1),D2(L2,B2) [SS]

F3	L_1	L_2	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The format of the second operand is changed from packed to zoned, and the result is placed in the first-operand location.

The digits and sign of the packed operand 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 packed operand. The operand sign and digits are not checked for valid codes.

The result is obtained as if the fields were processed right to left. The second operand is extended with high-order zero digits before unpacking, if necessary. If the first-operand field is too short to contain all significant digits of the second operand, the remaining high-order digits are ignored.

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 high-order digit of the byte remains available and is 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

A field that is to be unpacked can be destroyed by improper overlapping. If it is desired to save storage space for unpacking by overlapping the operand fields, the low-order position of the first operand must be to the right of the low-order position of the second operand by the number of bytes in the second operand minus two. If only one or two bytes are to be unpacked, the low-order positions of the two operands may coincide.

Decimal Instructions

Contents

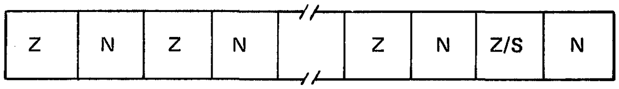
Data Format 147
Zoned Format 147
Packed Format 147
Number Representation 148
Instructions 148
ADD DECIMAL 149
COMPARE DECIMAL 149
DIVIDE DECIMAL 149
EDIT 150
EDIT AND MARK 152
MULTIPLY DECIMAL 153
SHIFT AND ROUND DECIMAL 153
SUBTRACT DECIMAL 154
ZERO AND ADD 155

Decimal instructions provide arithmetic, shifting, and editing operations on decimal data. These instructions constitute the decimal feature.

Data Format

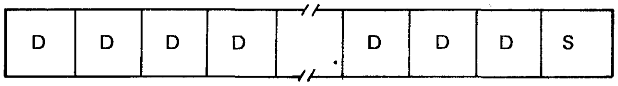
Decimal operands reside in main storage and may be in either the zoned or packed format.

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 alphameric character set, which includes also alphabetic and special characters. The zoned format is usually produced by source-document input devices, such as a card reader, and is usually used for printing decimal data on an output device.

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. These four instructions are not part of the decimal feature and are described in the chapter "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 main 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 size 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, an instruction such as ADD DECIMAL cannot be safely used to update a shared location in main storage 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 following table. The table also indicates when the condition code is set and the exceptions 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 IBM System/370 assembly language are shown with each instruction. For ADD DECIMAL, for example, AP is the mnemonic and

Name	Mnemonic	Characteristics										Code
ADD DECIMAL	AP	SS	C	PD	A		D	DF			ST	FA
COMPARE DECIMAL	CP	SS	C	PD	A		D					F9
DIVIDE DECIMAL	DP	SS		PD	A	SP	D		DK		ST	FD
EDIT	ED	SS	C	PD	A		D				ST	DE
EDIT AND MARK	EDMK	SS	C	PD	A		D			R	ST	DF
MULTIPLY DECIMAL	MP	SS		PD	A	SP	D				ST	FC
SHIFT AND ROUND DECIMAL	SRP	SS	C	PD	A		D	DF			ST	F0
SUBTRACT DECIMAL	SP	SS	C	PD	A		D	DF			ST	FB
ZERO AND ADD	ZAP	SS	C	PD	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
PD	Decimal feature
R	PER general register alteration event
SP	Specification exception
SS	SS instruction format
ST	PER storage alteration event

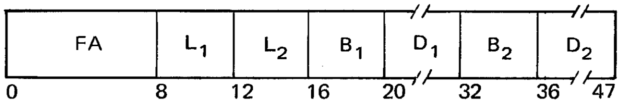
Decimal Instruction Summary

$D_1(L_1, B_1)$, $D_2(L_2, B_2)$ the operand designation.

Programming Note
The moving and logical comparing instructions may also be used in decimal calculations.

Add Decimal

AP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



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 is the loss of a carry out of the high-order digit position of the result field. The second cause is an oversized result, which occurs when the second-operand field is larger than the first-operand field and significant result digits are lost. The field sizes 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.

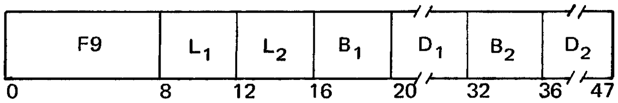
Resulting Condition Code:
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Exceptions:

Operation (if the decimal feature is not installed)
Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal Overflow

Compare Decimal

CP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



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.

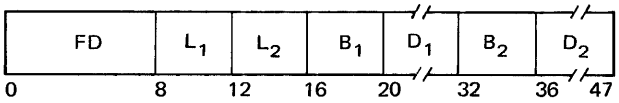
Resulting Condition Code:
0 Operands equal
1 First operand is low
2 First operand is high
3 -

Program Exceptions:

Operation (if the decimal feature is not installed)
Access (fetch, operands 1 and 2)
Data

Divide Decimal

DP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



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 size equal to the divisor size. 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 size of the quotient field in eight-bit 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 larger than seven (15 digits and sign) or larger 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. 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. A quotient larger than the number of digits allowed is recognized as a decimal-divide exception. The operation is suppressed, and a program interruption occurs. The divisor and dividend remain unchanged in their storage locations.

The divisor and dividend fields may overlap only if their low-order bytes coincide.

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the decimal feature is not installed)
Access (fetch, operand 2; fetch and store, operand 1)
Specification
Data
Decimal Divide

Programming Notes

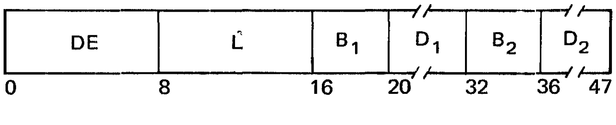
The maximum dividend size is 31 digits and sign. Since the smallest remainder size is one digit and sign, the maximum quotient size is 29 digits and sign.

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.

A decimal-divide exception occurs if the dividend does not have at least one leading zero.

Edit

ED $D_1(L,B_1),D_2(B_2)$ [SS]



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 has the zoned format and may contain any character. The source (the second operand) 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.

Summary: The following table 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 Côde:

0 Last field is zero
1 Last field is less than zero
2 Last field is greater than zero
3 -

Program Exceptions:

Operation (if the decimal feature is not installed)
Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Notes

As a rule, the source is shorter than the pattern because for each source digit a zone and numeric are inserted in the result.

The total number of digit selectors and significance starters in the pattern must equal the number of source digits to be edited.

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.

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.

Edit and Mark

EDMK $D_1(L,B_1),D_2(B_2)$ [SS]

DF	L	B_1	D_1	B_2	D_2
0	8	16	20	32	36 47

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
		1-9	yes	source digit	off
	on	0-9	no	source digit	on
		0-9	yes	source digit	off
Significance starter	off	0	no	fill character	on
		0	yes	fill character	off
		1-9	no	source digit	on
		1-9	yes	source digit	off
	on	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

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:

Operation (if the decimal feature is not installed)
Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Note

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_2 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

MP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

FC	L_1	L_2	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The product of the multiplier (the second operand) and the multiplicand (the first operand) replaces the multiplicand.

The multiplier size is limited to fifteen digits and sign and must be less than the multiplicand size. Length code L_2, larger than seven, or larger than or equal to the length code L_1, is recognized as a specification exception. 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 field size, 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 size 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. 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 zero.

The multiplier and product fields may overlap only if their low-order bytes coincide.

Condition Code:

The code remains unchanged.

Program Exceptions:

Operation (if the decimal feature is not installed)
Access (fetch, operand 2; fetch and store, operand 1)
Specification
Data

Shift and Round Decimal

SRP $D_1(L_1,B_1),D_2(B_2),I_3$ [SS]

F0	L_1	I_3	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

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:

	Shift Value
0	26 31

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:

Shift Value	Amount and Direction
0 1 1 1 1 1	31 digits to the left
0 0 0 0 0 1	One digit to the left
0 0 0 0 0 0	No shift
1 1 1 1 1 1	One digit to the right
1 0 0 0 0 0	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:

Operation (if the decimal feature is not installed)
Access (fetch and store, operand 1)
Data
Decimal overflow

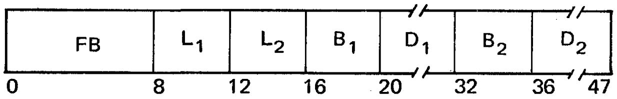
Programming Note

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 value, bits 26-31 of the second-operand address, is obtained directly from bits 42-47 of the instruction.

Subtract Decimal

SP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



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

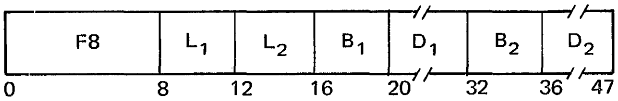
Program Exceptions:

Operation (if the decimal feature is not installed)
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

ZAP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]



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:

Operation (if the decimal feature is not installed)
Access (fetch, operand 2; store, operand 1)
Data
Decimal Overflow

Floating-Point Instructions

Contents

Data Format 157
Guard Digit 158
Number Representation 159
Normalization 159
Instructions 160
ADD NORMALIZED 160
ADD UNNORMALIZED 162
COMPARE 163
DIVIDE 163
HALVE 164
LOAD 165
LOAD AND TEST 165
LOAD COMPLEMENT 165
LOAD NEGATIVE 166
LOAD POSITIVE 166
LOAD ROUNDED 166
MULTIPLY 167
STORE 168
SUBTRACT NORMALIZED 169
SUBTRACT UNNORMALIZED 169

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 "Number Representation"); 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.

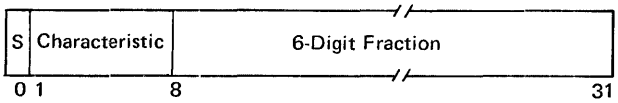
The rounding and extended-operand instructions are part of the extended-precision floating-point feature. The other floating-point instructions and the registers are part of the floating-point feature.

Data Format

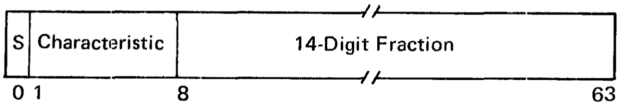
Floating-point data occupies a fixed-length format, which may be either a four-byte (short) format, an eight-byte (long) format, or a 16-byte (extended) format. The short and long formats may be designated as operands both in main storage and in the floating-point registers, whereas the extended formats 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_1 or R_2 field of a floating-point instruction causes the operation to be suppressed and a program interruption for specification exception to occur.

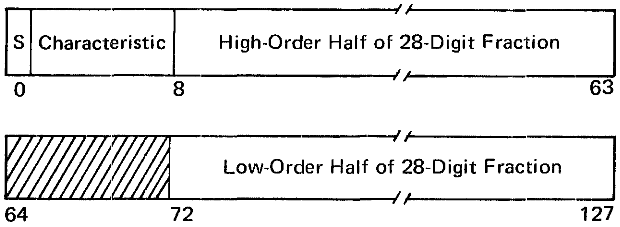
Short Floating-Point Number



Long Floating-Point Number



Extended Floating-Point Number


S
Characteristic
High-Order Half of 28-Digit Fraction
0
8
63
Low-Order Half of 28-Digit Fraction
64
72
127

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 zero, the number is said to be unnormalized. The process of normalization consists of 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 follow. The table indicates when the condition code is set and the exceptions 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 IBM System/370 assembly language are shown with each instruction. For a register-to-register operation using LOAD (short), for example, LER is the mnemonic and R_1,R_2 the operand designation.

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

Add Normalized

AER R_1,R_2
[RR, Short Operands]

3A	R_1	R_2
0	8	12 15

AE R_1,D_2(X_2,B_2)
[RX, Short Operands]

7A	R_1	X_2	B_2	D_2
0	8	12	16	20 31

ADR R_1,R_2
[RR, Long Operands]

2A	R_1	R_2
0	8	12 15

AD R_1,D_2(X_2,B_2)
[RX, Long Operands]

6A	R_1	X_2	B_2	D_2
0	8	12	16	20 31

AXR R_1,R_2
[RR, Extended Operands]

36	R_1	R_2
0	8	12 15

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-

Name	Mnemonic	Characteristics										Code
ADD NORMALIZED (extended)	AXR	RR	C	XP		SP	U	E		LS		36
ADD NORMALIZED (long)	ADR	RR	C	FP		SP	U	E		LS		2A
ADD NORMALIZED (long)	AD	RX	C	FP	A	SP	U	E		LS		6A
ADD NORMALIZED (short)	AER	RR	C	FP		SP	U	E		LS		3A
ADD NORMALIZED (short)	AE	RX	C	FP	A	SP	U	E		LS		7A
ADD UNNORMALIZED (long)	AWR	RR	C	FP		SP		E		LS		2E
ADD UNNORMALIZED (long)	AW	RX	C	FP	A	SP		E		LS		6E
ADD UNNORMALIZED (short)	AUR	RR	C	FP		SP		E		LS		3E
ADD UNNORMALIZED (short)	AU	RX	C	FP	A	SP		E		LS		7E
COMPARE (long)	CDR	RR	C	FP		SP						29
COMPARE (long)	CD	RX	C	FP	A	SP						69
COMPARE (short)	CER	RR	C	FP		SP						39
COMPARE (short)	CE	RX	C	FP	A	SP						79
DIVIDE (long)	DDR	RR		FP		SP	U	E	FK			2D
DIVIDE (long)	DD	RX		FP	A	SP	U	E	FK			6D
DIVIDE (short)	DER	RR		FP		SP	U	E	FK			3D
DIVIDE (short)	DE	RX		FP	A	SP	U	E	FK			7D
HALVE (long)	HDR	RR		FP		SP	U					24
HALVE (short)	HER	RR		FP		SP	U					34
LOAD (long)	LDR	RR		FP		SP						28
LOAD (long)	LD	RX		FP	A	SP						68
LOAD (short)	LER	RR		FP		SP						38
LOAD (short)	LE	RX		FP	A	SP						78
LOAD AND TEST (long)	LTDR	RR	C	FP		SP						22
LOAD AND TEST (short)	LTER	RR	C	FP		SP						32
LOAD COMPLEMENT (long)	LCDR	RR	C	FP		SP						23
LOAD COMPLEMENT (short)	LCER	RR	C	FP		SP						33
LOAD NEGATIVE (long)	LNDR	RR	C	FP		SP						21
LOAD NEGATIVE (short)	LNER	RR	C	FP		SP						31
LOAD POSITIVE (long)	LPDR	RR	C	FP		SP						20
LOAD POSITIVE (short)	LPER	RR	C	FP		SP						30
LOAD ROUNDED (extended to long)	LRDR	RR		XP		SP		E				25
LOAD ROUNDED (long to short)	LRER	RR		XP		SP		E				35
MULTIPLY (extended)	MXR	RR		XP		SP	U	E				26
MULTIPLY (long)	MDR	RR		FP		SP	U	E				2C
MULTIPLY (long)	MD	RX		FP	A	SP	U	E				6C
MULTIPLY (long to extended)	MXDR	RR		XP		SP	U	E				27
MULTIPLY (long to extended)	MXD	RX		XP	A	SP	U	E				67
MULTIPLY (short to long)	MER	RR		FP		SP	U	E				3C
MULTIPLY (short to long)	ME	RX		FP	A	SP	U	E				7C
STORE (long)	STD	RX		FP	A	SP					ST	60
STORE (short)	STE	RX		FP	A	SP					ST	70
SUBTRACT NORMALIZED (extended)	SXR	RR	C	XP		SP	U	E		LS		37
SUBTRACT NORMALIZED (long)	SDR	RR	C	FP		SP	U	E		LS		2B
SUBTRACT NORMALIZED (long)	SD	RX	C	FP	A	SP	U	E		LS		6B
SUBTRACT NORMALIZED (short)	SER	RR	C	FP		SP	U	E		LS		3B
SUBTRACT NORMALIZED (short)	SE	RX	C	FP	A	SP	U	E		LS		7B
SUBTRACT UNNORMALIZED (long)	SWR	RR	C	FP		SP		E		LS		2F
SUBTRACT UNNORMALIZED (long)	SW	RX	C	FP	A	SP		E		LS		6F
SUBTRACT UNNORMALIZED (short)	SUR	RR	C	FP		SP		E		LS		3F
SUBTRACT UNNORMALIZED (short)	SU	RX	C	FP	A	SP		E		LS		7F

Explanation:

A	Access exceptions	RR	RR instruction format
C	Condition code is set	RX	RX instruction format
E	Exponent-overflow exception	SP	Specification exception
FK	Floating-point divide exception	ST	PER storage alteration event
FP	Floating-point feature	U	Exponent underflow exception
LS	Significance exception	XP	Extended-precision floating-point feature

Floating-Point-Instruction Summary

underflow mask bit is one, the operation is completed by making the characteristic 128 larger 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:

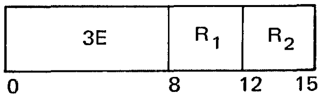
Operation (if the floating-point feature is not installed, or, for AXR, if the extended-precision floating-point feature is not installed)
Access (fetch, operand 2 of AE and AD only)
Specification
Exponent Overflow
Exponent Underflow
Significance

Programming Note

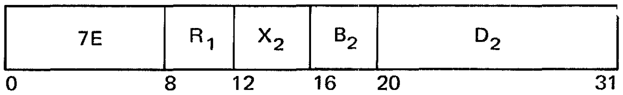
Interchanging the two operands in a floating-point addition does not affect the value of the sum.

Add Unnormalized

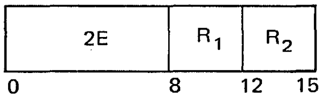
AUR R_1,R_2
[RR, Short Operands]



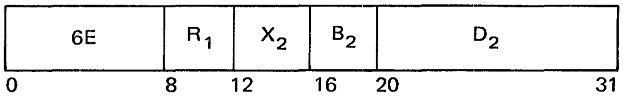
AU $R_1,D_2(X_2,B_2)$
[RX, Short Operands]



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:

Operation (if the floating-point feature is not installed)
Access (fetch, operand 2 of AU and AW only)
Specification
Exponent Overflow
Significance

Compare

CER R_1,R_2
[RR, Short Operands]

39	R_1	R_2
0	8	12 15

CE $R_1,D_2(X_2,B_2)$
[RX, Short Operands]

79	R_1	X_2	B_2	D_2
0	8	12	16	20 31

CDR R_1,R_2
[RR, Long Operands]

29	R_1	R_2
0	8	12 15

CD $R_1,D_2(X_2,B_2)$
[RX, Long Operands]

69	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 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:

Operation (if the floating-point feature is not installed)
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]

3D	R_1	R_2
0	8	12 15

DE $R_1,D_2(X_2,B_2)$
[RX, Short Operands]

7D	R_1	X_2	B_2	D_2
0	8	12	16	20 31

DDR R_1,R_2
[RR, Long Operands]

2D	R_1	R_2
0	8	12 15

DD $R_1,D_2(X_2,B_2)$
[RX, Long Operands]

6D	R_1	X_2	B_2	D_2
0	8	12	16	20 31

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, and a program interruption occurs. The result is normalized, the sign and fraction remain correct, and the characteristic is 128 less than the correct value.

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, its sign and fraction remain correct, and the characteristic is made 128 larger than the correct value. 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:

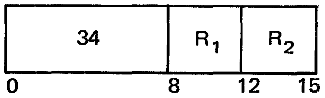
The code remains unchanged.

Program Exceptions:

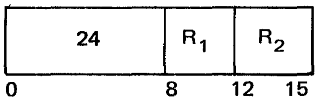
Operation (if the floating-point feature is not installed)
Access (fetch, operand 2 of DD and DE only)
Specification
Exponent Overflow
Exponent Underflow
Floating-Point Divide

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, its sign and fraction remain correct, and the characteristic is made 128 larger 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_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:

Operation (if the floating-point feature is not installed)
Specification
Exponent Underflow

Programming Notes
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.

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_1,R_2
[RR, Short Operands]

38	R_1	R_2
0	8	12 15

LE $R_1,D_2(X_2,B_2)$
[RX, Short Operands]

78	R_1	X_2	B_2	D_2
0	8	12	16	20 31

LDR R_1,R_2
[RR, Long Operands]

28	R_1	R_2
0	8	12 15

LD $R_1,D_2(X_2,B_2)$
[RX, Long Operands]

68	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The second operand is placed unchanged in the first-operand location.

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:

Operation (if the floating-point feature is not installed)
Access (fetch, operand 2 of LE and LD only)
Specification

Load and Test

LTER R_1,R_2
[RR, Short Operands]

32	R_1	R_2
0	8	12 15

LTDR R_1,R_2
[RR, Long Operands]

22	R_1	R_2
0	8	12 15

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_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:

Operation (if the floating-point feature is not installed)
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_1,R_2
[RR, Short Operands]

33	R_1	R_2
0	8	12 15

LCDR R_1,R_2
[RR, Long Operands]

23	R_1	R_2

0 8 12 15

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_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:

Operation (if the floating-point feature is not installed)

Specification

Load Negative

LNER R_1,R_2
[RR, Short Operands]

31	R_1	R_2

0 8 12 15

LNDR R_1,R_2
[RR, Long Operands]

21	R_1	R_2

0 8 12 15

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_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 -
3 -

Program Exceptions:

Operation (if the floating-point feature is not installed)
Specification

Load Positive

LPER R_1,R_2
[RR, Short Operands]

30	R_1	R_2

0 8 12 15

LPDR R_1,R_2
[RR, Long Operands]

20	R_1	R_2

0 8 12 15

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_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 -
2 Result is greater than zero
3 -

Program Exceptions:

Operation (if the floating-point feature is not installed)

Specification

Load Rounded

LRER R_1,R_2
[RR, Long Operand 2, Short Operand 1]

35	R_1	R_2

0 8 12 15

LRDR R_1,R_2
[RR, Extended Operand 2, Long Operand 1]

25	R_1	R_2
0	8	12 15

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_1 field must designate register 0, 2, 4, or 6; the R_2 field of LRER must designate register 0, 2, 4, or 6; and the R_2 field of LRDR must designate register 0 or 4. Otherwise, a specification exception is recognized.

Condition Code:
The code remains unchanged.

Program Exceptions:

Operation (if the extended-precision floating-point feature is not installed)

Specification

Exponent Overflow

Multiply

MER R_1,R_2
[RR, Short Multiplier and Multiplicand, Long Product]

3C	R_1	R_2
0	8	12 15

ME R_1,D_2(X_2,B_2)
[RX, Short Multiplier and Multiplicand, Long Product]

7C	R_1	X_2	B_2	D_2
0	8	12	16	20 31

MDR R_1,R_2
[RR, Long Operands]

2C	R_1	R_2
0	8	12 15

MD R_1,D_2(X_2,B_2)
[RX, Long Operands]

6C	R_1	X_2	B_2	D_2
0	8	12	16	20 31

MXDR R_1,R_2
[RR, Long Multiplier and Multiplicand, Extended Product]

27	R_1	R_2
0	8	12 15

MXD R_1,D_2(X_2,B_2)
[RX, Long Multiplier and Multiplicand, Extended Product]

67	R_1	X_2	B_2	D_2
0	8	12	16	20 31

MXR R_1,R_2
[RR, Extended Operands]

26	R_1	R_2
0	8	12 15

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 zero. 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 zero, 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 larger 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 zero or above.

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:

Operation (if the floating-point feature is not installed, or, for MXDR, MXD, and MXR, if the extended-precision floating-point feature is not installed)

Access (fetch, operand 2 of ME, MD, and MXD only)

Specification

Exponent Overflow

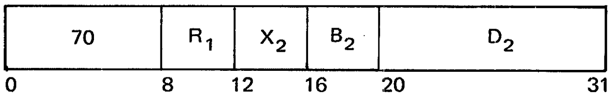
Exponent Underflow

Programming Note

Interchanging the two operands in a floating-point multiplication does not affect the value of the product.

Store

STE R_1,D_2(X_2,B_2)
[RX, Short Operands]



STD $R_1,D_2(X_2,B_2)$
[RX, Long Operands]

60	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The first operand is placed unchanged at the second-operand location.

The R_1 field must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Condition Code:
The code remains unchanged.

Program Exceptions:

Operation (if the floating-point feature is not installed)

Access (store, operand 2)

Specification

Subtract Normalized

SER R_1,R_2
[RR, Short Operands]

3B	R_1	R_2
0	8	12 15

SE $R_1,D_2(X_2,B_2)$
[RX, Short Operands]

7B	R_1	X_2	B_2	D_2
0	8	12	16	20 31

SDR R_1,R_2
[RR, Long Operands]

2B	R_1	R_2
0	8	12 15

SD $R_1,D_2(X_2,B_2)$
[RX, Long Operands]

6B	R_1	X_2	B_2	D_2
0	8	12	16	20 31

SXR R_1,R_2
[RR, Extended Operands]

37	R_1	R_2
0	8	12 15

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_1 field of SER, SE, SDR, and SD, and the R_2 field of SER and SDR must designate register 0, 2, 4, or 6. The R_1 and R_2 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:

Operation (if the floating-point feature is not installed, or, for SXR, if the extended-precision floating-point feature is not installed)
Access (fetch, operand 2 of SE and SD only)
Specification
Exponent Overflow
Exponent Underflow
Significance

Subtract Unnormalized

SUR R_1,R_2
[RR, Short Operands]

3F	R_1	R_2
0	8	12 15

SU $R_1,D_2(X_2,B_2)$
[RX, Short Operands]

7F	R_1	X_2	B_2	D_2
0	8	12	16	20 31

SWR R_1,R_2
[RR, Long Operands]

2F	R_1	R_2
0	8	12 15

SW R_1,D_2(X_2,B_2)
[RX, Long Operands]

6F	R_1	X_2	B_2	D_2
0	8	12	16	20 31

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_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:

Operation (if the floating-point feature is not installed)

Access (fetch, operand 2 of SU and SW only)

Specification
Significance

Machine-Check Handling

Contents

Machine-Check Detection 171
Recovery Mechanisms 172
Redundancy Correction 172
CPU Retry 172
Unit Deletion 172
Handling of Machine Checks 172
Handling of Invalid CBC in Storage 172
Programmed Validation of Storage 173
Handling of Invalid CBC in Keys in Storage 173
Handling of Invalid CBC in Registers 173
Validation of Registers 173
Check-Stop State 175
Machine-Check Interruption Conditions 175
Repressible Conditions 175
Exigent Conditions 175
Machine-Check Interruption 175
Interruption Action 175
Point of Interruption 176
Machine-Check Logout 177
Machine-Check Extended Interruption Information 177
Machine-Check Interruption Code. 178
Subclass 178
Time of Interruption Occurrence 179
Storage Error Type 179
Machine-Check Interruption Code Validity Bits 180
Machine-Check Extended Logout Length 181
Machine-Check Control Registers 181
Control Register 14 181
Check-Stop Control 181
Logout Controls 181
Machine-Check Subclass Masks 182
Control Register 15 182
Summary of Machine-Check Masking 183

The System/370 machine-check handling mechanism provides extensive machine-malfunction detection to ensure the integrity of system operation, automatic recovery from some malfunctions, and reporting by means of a machine-check interruption to assist in maintenance and repair and 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 redundancy bits appended to a group of information bits. These redundancy bits are referred to as "check bits." The group of data bits and the associated check bits are called the "checking block."

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 checking bit is sometimes referred to as a "parity bit." In other arrangements, a group of check bits is included, increasing the checking power and, in some cases, providing sufficient redundancy to permit both detection and correction.

For checking purposes, the entire content 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.*

Recovery Mechanisms

Three mechanisms may be used to provide recovery from machine-detected malfunctions: redundancy correction, retry, and unit deletion.

Redundancy Correction

When sufficient redundancy is included in circuitry, or in a checking block, failures can be corrected. For example, circuitry can be triplicated, with a voting circuit to take two out of three, thus correcting a single failure. An arrangement for redundancy correction of failures of one order and for detection of failures of a higher order is called error checking and correction (ECC). Normally, ECC allows detection of double-bit failures and correction of single-bit failures.

CPU Retry

In models with CPU-retry capability, information about the state of the machine is saved periodically. The point in the processing to which this saving of information pertains is referred to as a "hardware checkpoint." When a malfunction is detected, recovery is attempted by returning the machine state to that existing at the latest hardware checkpoint and proceeding from that point. The interval between checkpoints is model-dependent. In some cases, several checkpoints are established within a single instruction; in others, checkpoints are established only at the beginning of instructions, or even less frequently.

Unit Deletion

In some models, malfunctions in certain transparent units of the system can be circumvented by discontinuing the use of the unit. Examples of cases where transparent-unit deletion may be used include the disabling of all or a portion of a cache or of a translation-lookaside buffer (TLB).

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 programmable registers, in the keys in storage, and, if it is volatile, also in main storage.

Specification of an unavailable system component, such as a storage unit, 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 code setting.

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 is not used, the condition may or may not be indicated. In order to guarantee system integrity, however, CBC is preserved as invalid unless the contents of the entire checking block are replaced in the operation.

Depending on the model, and on the type of malfunction, a malfunction detected during an I/O operation may cause a machine-check interruption condition, may result in 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, it depends on the model whether the condition is reported.

When a machine malfunction is detected, the action taken depends upon the nature of the malfunction and the situation in which it occurs. In some cases an automatic hardware recovery mechanism may be invoked. When the recovery attempt is unsuccessful, or if a recovery mechanism does not exist, a damage condition is said to exist. Machine-check conditions may be reported as machine-check interruptions or I/O interruptions, or they may cause the CPU to enter the check-stop state.

Handling of Invalid CBC in Storage

When a checking block contains an invalid CBC and an attempt is made to store into the block without replacing the entire block, the data in the block (including the check bits) is regenerated by the storage unit, and no new data is entered into the block. Normally the contents of the block can only be changed by presenting an entire block of data to be entered on one storage cycle.

The size of the main-storage checking block depends on the model. When the main-storage checking block consists of multiple bytes and contains an invalid CBC, special procedures are necessary to restore or place new information into the block. The restoring of a valid CBC in a storage location is called storage validation. Validation of storage is provided as a program function and is also provided with the system clear-manual operation.

A checking block with invalid CBC is never validated under programming control unless the entire contents of the checking block are replaced. Even when an instruction, or an I/O input operation, specifies that the entire contents of a checking block are to be replaced, validation may or may not occur, depending on the operation and the model. Storage validation during the IPL input operations follows the same rules as for normal input operations.

Programmed Validation of Storage

Execution of the instruction MOVE (MVC) or MOVE LONG (MVCL) validates the main-storage area containing the first operand when the following conditions are satisfied:

- The first-operand field and second-operand field participating in the operation do not overlap.
- The first-operand field starts on a boundary of a checking block and is an integral number of checking blocks in length.
- For MVCL, the second-operand field, if nonzero in length, starts on a boundary of a checking block and, if it is shorter than the first-operand field, is an integral number of checking blocks in length.

An interruption or stopping of the CPU during execution of MVCL does not affect the validation function performed.

Handling of Invalid CBC in Keys in Storage

Depending on the model, each key in storage may consist of a single checking block, or the protection bits and the change and reference bits may be in separate checking blocks. Invalid CBC on the key in storage is ignored in storing or fetching with a zero protection key. References to main storage to which protection does not apply are treated as if a protection key of zero is used for the reference. This includes such references as channel references during the IPL procedure, implicit references such as in timer updating and interruption action, and DAT table accesses. The key in storage is validated by SET STORAGE KEY.

The table "Handling of Invalid CBC in Keys in Storage" describes the action taken when the key in storage has invalid CBC.

Handling of Invalid CBC in Registers

During a machine-check interruption, the contents of the general, floating-point, and control registers, and of the CPU timer and clock comparator if they are installed, are stored at fixed locations in main storage. Invalid CBC detected during this operation does not result in additional machine-check-interruption conditions; instead, the accuracy of the information stored is indicated by the appropriate setting of the validity bits in the machine-check-interruption code. On some models, registers with invalid CBC will be automatically validated during the interruption. On other models, programmed validation is required. The TOD clock and the prefix register are not stored during the machine-check interruption and are not validated.

On those models in which registers are not automatically validated as part of the machine-check interruption, a register with invalid CBC will not cause a machine-check interruption condition unless the contents of the register are actually used. In these models, each register may consist of one or more checking blocks, but multiple registers are not included in a single checking block. When only a portion of a register is accessed, invalid CBC in the unused portion of the same register may cause a machine-check interruption condition. For example, invalid CBC in the right half of a long operand of a floating-point register may cause a machine-check interruption condition if a LOAD (LE) operation attempts to replace the left half, or short form, of the register.

Invalid CBC associated with the check-stop control bit (control register 14, bit 0) and with the asynchronous fixed-logout control bit (control register 14, bit 9) will cause the CPU either to immediately enter check-stop state or to assume that bits 0 and 9 have their initialized values of one and zero, respectively.

Invalid CBC associated with the prefix register cannot be safely reported by the machine-check interruption, since the interruption itself requires that the prefix value be applied to convert real addresses to the corresponding absolute addresses. When the check-stop control bit (control register 14, bit 0) is one, invalid CBC in the prefix register causes the CPU to immediately enter the check-stop state. When the check-stop control bit is zero, invalid CBC in the prefix register either may cause the CPU to enter the check-stop state or may generate a system damage condition, depending on the model.

Validation of Registers

On those models which do not validate registers during a machine-check interruption, the following instructions will cause validation of a register, provided the information in the register is not used before the register is validated. Other instructions, although they replace the entire contents of a register, do not necessarily cause validation.

General registers are validated by BRANCH AND LINK (BAL, BALR), LOAD (LR), and LOAD ADDRESS (LA). LOAD (L) and LOAD MULTIPLE (LM) validate if the operand is on a

	Action Taken on Invalid CBC		
Type of Reference	For Protection Bits	For Reference and Change Bits	For Protection Bits and Reference and Change Bits
Set Storage Key	Complete; validate.	Complete; validate.	Complete; validate.
Insert Storage Key	PD; preserve.	PD in EC mode, PD or complete in BC mode; preserve.	PD; preserve.
Reset Reference Bit	PD or complete; preserve.	PD; preserve.	PD; preserve.
Fetch, Nonzero Protection Key	MC; preserve.	MC or complete; preserve.	MC; preserve.
Store, Nonzero Protection Key	MC[1]; preserve.	MC or complete; preserve or correct.	MC[1]; preserve.
Fetch, Zero Protection Key	Complete; preserve.	Complete; preserve.	Complete; preserve.
Store, Zero Protection Key	Complete; preserve.	Complete; preserve or correct.	Complete; preserve[2].

Explanation:

Complete The condition does not cause termination of the execution of the instruction and, unless an unrelated condition prohibits it, the execution of the instruction is completed, ignoring the error condition. No machine-check damage conditions are generated, but recovery-report conditions may be generated.

PD A machine-check instruction processing damage or system damage condition is recognized.

MC Same as PD for CPU references, but an I/O reference may result in the following combinations of I/O interruption and machine-check interruption.
a) Channel control check and no machine-check interruption.
b) Channel control check and a recovery report.
c) External damage and no I/O interruption.
d) System damage and no I/O interruption.

Validate The entire key is set to the new value with valid CBC.

Preserve The contents of the entire checking block having invalid CBC are left unchanged.

Correct The reference and change bits are set to one with valid CBC.

1 The contents of the main-storage location are not changed.

2 On models with separate checking blocks for protection bits and for change and reference bits; the protection bits are preserved, and the change and reference bits may be corrected or preserved.

Handling of Invalid CBC in Keys in Storage

word boundary, and LOAD HALFWORD (LH) validates if the operand is on a halfword boundary.

Floating-point registers are validated by LOAD (LDR) and, if the operand is on a doubleword boundary, by LOAD (LD).

Control registers may be validated either singly or in groups by using the instruction LOAD CONTROL (LCTL).

The CPU timer and clock comparator are validated by SET CPU TIMER (SPT) and SET CLOCK COMPARATOR (SCKC), respectively.

The TOD clock is validated by SET CLOCK (SCK) if the TOD clock security switch is in the enable-set position.

Programming Note

To provide for a model-independent machine-check first-level-interruption handler, registers must be validated before they are used. Examples: START I/O, SET SYSTEM MASK, and SET CLOCK should not be executed until control register 0 (containing block-multiplexing control, SSM-suppression control, and TOD clock synchronization control bits), is validated. MONITOR CALL should not be issued until control register 8, containing the monitor class masks, is validated. Extended channel masks, external masks, and machine-check controls should be validated before the associated interruptions are allowed. The clock comparator and CPU timer should be validated before clock-comparator and CPU-timer interruptions are allowed.

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.

When the CPU is in the check-stop state, the condition is indicated by an error indicator, an audible signal, or both. The system indicator is off, but the state of the manual indicator depends on the model. The exact indication of check-stop state is model-dependent and is described in the System Library (SL) publication for the CPU.

The machine enters the check-stop state only as a result of exigent conditions. The machine may be removed from the check-stop state by CPU reset.

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 suspended. The TOD clock is not normally affected by check-stop state. The CPU timer may or may not run in check-stop state, depending on the error and the model. The CPU cluster meter does not run, and the clock-out and metering-out lines are down. The stop key and start key are not operative during this state.

In a multiprocessing system, a CPU entering the check-stop state generates a request for a malfunction-alert external interruption to all CPUs configured to this CPU.

Machine-Check Interruption Conditions

Equipment malfunctions and other conditions responsible for machine-check interruptions are referred to as machine-check interruption conditions. Two major types of conditions are identified: exigent conditions and repressible conditions.

Repressible Conditions

Repressible conditions are those in which the sequential processing capability of the CPU has not been affected. Repressible conditions can be delayed until the completion of the current instruction, and in most cases, even longer, without affecting the integrity of the CPU operation. Repressible conditions are of three types: recovery, alert, and repressible damage. Each has one or more subclasses as follows:

A hardware malfunction successfully corrected or circumvented without loss of system integrity is called a *recovery condition.* Depending on the model and the type of malfunction, some 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 hardware malfunction is called an *alert condition.* The alert conditions contain two subclasses: degradation and warning.

A hardware malfunction resulting in the loss of integrity of a process in the system but not directly affecting the sequential CPU operation is called a *repressible damage condition.* Repressible damage conditions are divided into three subclasses, identifying the process affected: timer damage, timing-facility damage, and external damage.

Exigent Conditions

Exigent conditions are those in which direct damage has occurred to the CPU operation, and 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 process are indicated as system damage.

Machine-Check Interruption

The machine-check interruption provides a means of reporting equipment malfunction 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.

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; extended machine-check interruption information is stored, consisting of the information in all the control registers, general registers, floating-point registers, CPU timer, clock comparator, a region code, and a failing storage address. Then the machine-check interruption code (MCIC) of eight bytes is stored. A new PSW is fetched from location 112. Additionally, sometime before the storing of the machine-check interruption code, one or several machine-check logouts may have occurred. The machine-generated addresses to reference the old and new PSW, the interruption code and extended interruption information, and the fixed logout area are all real addresses. The extended machine-check logout address is also a real address. 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 if the check-stop control bit is one.

A machine-check interruption due to a repressible machine-check condition can occur only when both PSW bit 13 and the associated subclass mask are ones. A repressible machine-check interruption does not terminate the execution of the current instruction; the interruption is taken after the execution of the current instruction has come to its normal ending and the associated program or supervisor-call interruption, if any, has been taken. No program or supervisor-call interruptions are eliminated. If the repressible machine-check condition occurs during the execution of a system function such as a timer update, the machine-check interruption takes place after the system function has been completed.

A machine-check interruption due to an exigent machine-check condition can occur only when PSW bit 13 is one. The interruption terminates the execution of the current instruction and may eliminate the program and supervisor-call interruptions, if any, that would have occurred as a result of continuing execution of the instruction. Proper execution of the interruption steps, including the storing of the old PSW and diagnostic information, depends on the nature of the malfunction. When an exigent machine-check condition occurs during the execution of a system function, such as a timer update, the sequence is not necessarily completed.

When PSW bit 13 is zero and an exigent machine-check condition is generated, subsequent action depends on the state of the check-stop control bit, bit 0 of control register 14. When the check-stop control bit is zero, the machine-check condition is held pending, and an attempt is made to complete the execution of the current instruction and to proceed with the next sequential instruction. When the check-stop control bit is one, processing stops immediately, and the CPU enters the check-stop state. Depending on the model and the severity of the error, the CPU may enter the check-stop state even when the check-stop control bit is zero.

Similarly, if, during the execution of an interruption due to one exigent machine-check condition, another exigent machine-check condition is detected, subsequent action depends on the state of the check-stop control bit. If the check-stop control bit is one, the CPU enters the check-stop state; if the bit is zero, an attempt is made to proceed with the condition held pending for subsequent interruption. If an exigent machine-check condition is detected during an interruption due to a repressible machine-check condition, system damage is also reported.

Exigent machine-check conditions held pending while the check-stop control bit is zero remain pending and do not cause the CPU to enter the check-stop state if the check-stop control bit is subsequently set to one.

If a repressible machine-check condition is detected with the CPU disabled for the associated machine-check interruption condition, the condition is held pending. 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. The CPU never enters the check-stop state because of a repressible machine-check condition.

Only one machine-check interruption condition is held pending for each subclass, regardless of the number of conditions that may have been detected.

Machine-check interruptions can be initiated only by an interruption condition in a subclass for which the CPU is enabled. Conditions in other subclasses which are pending may also be indicated in the same interruption even though the CPU is not enabled for those subclasses. All conditions which are indicated are then cleared.

Machine-check interruption conditions are handled in the same manner in both the running and wait states. In the wait state, a machine-check interruption condition for which the CPU is enabled causes an immediate interruption.

Machine checks which occur while processing is in the instruction-step mode are handled in the same manner as in process mode; that is, normal recovery, logout, and machine-check interruptions occur when allowed. Machine checks occurring during a manual operation such as system reset, set IC, or store, may generate a system-recovery condition. If damage has been caused which is not corrected or not circumvented, the CPU enters the check-stop state.

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 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

Because of the checkpoint capability in models with machine retry, the interruption resulting from an exigent machine-check interruption condition may indicate a point in the recovery cycle which is prior to the error. Additionally, the model may have some choice as to which point in the recovery cycle the interruption will indicate, and, in some cases, the status which can be marked as valid depends on the point chosen.

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, including the associated program or supervisor-call interruption, if applicable, and before the next unit of operation is begun.

Exigent machine-check conditions which are delayed (disallowed and presented later when allowed) can occur only at the same points of interruption as repressible machine-check conditions. When an exigent machine-check condition is not delayed, the point of interruption may also be after the unit of operation is completed but before the 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.

Programming Note

When an exigent machine-check condition occurs, the point of interruption which is chosen affects the amount of damage which must be indicated. An attempt is made, when possible, to choose a point of interruption which permits the minimum indication of damage. In general, the preference is the interruption point immediately preceding the error. When a point of interruption is chosen which is after an associated program or supervisor-call interruption, the damage has not been isolated to a particular program, and system damage is indicated.

When all the status information stored as a result of an exigent machine-check condition does not reflect the same point, an attempt is made when possible to choose the point of interruption so that the instruction address which is stored in the machine-check old PSW is valid.

Machine-Check Logout

The storing of model-dependent information in main storage as a result of a machine check is referred to as a machine-check logout. Machine-check logouts are of four different types: synchronous fixed logout, asynchronous fixed logout, synchronous machine-check extended logout, and asynchronous machine-check extended logout.

When a machine-check logout occurs during the machine-check interruption it is called "synchronous." If a machine-check logout occurs without a machine-check interruption, or if the logout and the interruption are separated by instruction processing or by instruction retry, then the logout is called "asynchronous."

Machine-check-logout information can be placed in either or both of two areas. One area, the 96-byte area starting at location 256, is called the "fixed-logout area." Additionally, a machine-check extended-logout area (MCEL) is defined. The starting location of the MCEL area is specified by the contents of control register 15. The existence and length of the machine-check extended logout are model-dependent.

To preserve the initial machine-check conditions, some models perform an asynchronous logout before invoking automatic CPU recovery action. Depending on the model, logout may occur before recovery, after recovery, or at both times. If logout occurs at both times it may be into the same portion or two different portions of the logout area.

Machine-Check Extended Interruption Information

The machine-check extended interruption information consists of seven fields, which are stored at machine-check interruption time. Each of these fields has a validity bit associated with it 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.

Timing Facilities: When the system-timing facilities are present, any machine-check interruption causes the contents of the clock comparator and CPU timer to be placed in storage as part of the machine-check extended interruption information. The contents of the clock comparator are stored in the doubleword starting at location 224. The contents of the CPU timer are placed in the doubleword starting at location 216.

Failing-Storage Address: When a storage error uncorrected, storage error corrected, or key in storage error uncorrected has been indicated, the failing-storage address is stored in bits 8-31 of the word at location 248. Bits 0-7 of the word are set to zeros. In the case of storage errors, the failing-storage address may point to any byte within the checking block. For key in storage error uncorrected, the failing-storage address may point to any address within the 2,048-byte block of storage associated with the key in storage that is 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. The address stored is an absolute address; that is, the value stored is the address that is used to reference storage after dynamic address translation and prefixing, if any, have been applied.

Region Code: The word at location 252 contains model-dependent information which more specifically defines the location of the error. For example, it may contain a model-dependent address of the unit causing an external damage or recovery report.

Register Save Area: On all machine-check interruptions, the addressable registers are saved sequentially in storage. Floating-point registers 0, 2, 4, and 6 are stored starting at location 352; when the floating-point feature is not installed, these locations are left unchanged. General registers 0-15 are stored starting at location 384, and control registers 0-15 are stored starting at location 448. The information stored for control-register positions not associated with an installed feature is unpredictable.

Machine-Check Interruption Code

The machine-check interruption code (MCIC) is an eight-byte field starting at location 232 and has the format shown in the illustration.

Bits in the machine-check interruption code which are not assigned, or not implemented by a particular model, are stored as zeros.

Subclass

Bits 0-5, 7, and 8 identify the machine-check conditions causing the interruption. At least one bit will be stored as a one in the subclass field. When multiple errors have occurred, several 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.

Instruction Processing Damage (PD): Bit 1, when one, indicates that a malfunction has been detected in the processing of instructions. The exact meaning of bit 1 depends on the setting of the backed-up bit, bit 14.

When the backed-up bit is one, a valid instruction address stored in the machine-check old PSW, and the other machine status saved, point to the beginning of a unit of operation prior to the point at which the damage would have occurred. When the backed-up bit is one and all status is indicated as valid, the machine has successfully returned to a checkpoint prior to the malfunction, and no damage has yet occurred.

When the backed-up bit is zero, a valid instruction address points to the beginning of an instruction containing a unit of operation beyond the damaged unit of 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.

System Recovery (SR): Bit 2, when one, indicates that malfunctions were detected but have been successfully corrected or circumvented without the loss of system 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 damage conditions may result in a system recovery condition in addition to the I/O interruption. The indication of system recovery does not

S	P	S	T	C	E		D									S	S	K		W	M	P	I	F	R		F	G	C	L	S
D	D	R	D	D	D	0	G	W	0	0	0	0	0	B	D	E	C	E	0	P	S	M	A	A	C	0	P	R	R	G	T
0							7		9					14		16				20							27				31

														C	C	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	T	C	Machine-Check Extended Logout Length
32														46		48 63

Bits 0-5, 7, 8	Subclass
Bits 14-15	Time of interruption occurrence
Bits 16-18	Storage errors
Bits 20-31, 46, 47	Validity indicators
Bits 6, 9-13, 19, 26, 32-45	Not assigned, stored as zeros

Machine Check Interruption-Code Format

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.

Timer Damage (TD): Bit 3, when one, indicates that damage has occurred to the interval timer or to location 80.

Timing Facility Damage (CD): Bit 4, when one, indicates that damage has occurred to either 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 not-operational state.
2. The time-of-day clock enters the error state.
3. The time-of-day clock is not in the error state, and the STORE CLOCK instruction encounters an error which results in setting condition code 2. This condition also sets instruction processing damage.
4. The CPU timer is in error, and the CPU is enabled for CPU-timer interruptions. On some models, this condition may be recognized even when the CPU is not enabled for CPU-timer interruptions.
5. The CPU timer is in error, and STORE CPU TIMER is executed. This condition also sets instruction processing damage.
6. The clock comparator is in error, and the CPU is enabled for clock-comparator interruptions. On some models, this condition may be recognized even when the CPU is not enabled for clock-comparator interruptions.
7. The clock comparator is in error, and STORE CLOCK COMPARATOR is executed. This condition also sets instruction processing damage.

External Damage (ED): Bit 5, when one, indicates that damage has occurred to a channel, channel controller, switching unit, or other unit external to the CPU, or to a storage unit during operations not directly associated with the CPU. Channel-detected malfunctions are reported as external damage only when the channel is unable to report the malfunction by using the 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.

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 or when unit deletion has occurred. The presence and extent of the degradation-report capability depends on the model.

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.

Time of Interruption Occurrence

Bits 14 and 15 of the machine-check interruption code indicate when the interruption occurred in relation to the error.

Backed Up (B): Bit 14, when one, indicates that the point of interruption is at a hardware checkpoint before the point of error. This bit is meaningful only when instruction processing damage is also set to one. The presence and extent of the capability to indicate a backed-up condition depends on the model.

Programming Note

The backed-up situation is reported as instruction-processing damage rather than system recovery because the malfunction has not been circumvented and damage would have occurred if instruction processing had continued.

Delayed (D): Bit 15, when one, indicates that some or all of the machine-check conditions were delayed in being reported because the CPU was disabled for that type of interruption at the time the error was detected.

Storage Error Type

Bits 16-18 of the machine-check interruption code are used to indicate invalid CBC or near-valid CBC detected in main storage or invalid CBC in a key in storage. The failing-storage address field, when indicated as valid, identifies an address within the storage checking block or within the 2,048-byte block associated with the key in storage. The portion of the system affected by an invalid CBC is indicated in the subclass field of the machine-check interruption code. I/O-detected storage errors, when indicated as I/O interruptions, may not result in a machine-check interruption or may be reported as system recovery. CBC errors in storage or in the key in storage that are detected on prefetched or unused data may or may not be reported, depending on the model.

Storage Error Uncorrected (SE): Bit 16, when one, indicates that a checking block in main storage contains invalid CBC.

Storage Error Corrected (SC): Bit 17, when one, indicates that a checking block in main storage contained near-valid CBC and that the data portion of the information has been corrected before being used by the CPU or channel. Depending on the model, the contents of the checking block in main storage may or may not have been restored to valid CBC. The presence and extent of the storage-error-correction capability depends on the model.

Key in Storage Error Uncorrected (KE): Bit 18, when one, indicates that a key in storage contains invalid CBC.

Programming Note

The storage-error-type bits do not in themselves indicate the occurrence of damage because the error detected may not have affected the result. The subclass bits indicate, in conjunction with the storage-error-type bits, the area affected by the storage error.

Machine-Check Interruption Code Validity Bits

Bits 20-31 and bits 46 and 47 of the machine-check interruption code are validity bits. Each bit indicates the validity of a particular field in main 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 this specific field is valid with respect to the indicated point of interruption and that no error was detected when the data was stored. When the 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 during the storing of 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 ensure that the information in storage has valid CBC and thus reduce the possibility of additional machine checks being caused.

PSW EMWP Validity (WP): Bit 20, when one, indicates that bits 12-15 of the machine-check old PSW are correct.

PSW Masks and Key Validity (MS): Bit 21, when one, indicates that all PSW bits other than the interruption code, ILC, EMWP, instruction address, condition code, and program mask of the machine-check old PSW are correct.

Program Mask and Condition Code Validity (PM): Bit 22, when one, indicates that the program mask and condition code in the machine-check old PSW are correct.

Instruction Address Validity (IA): Bit 23, when one, indicates that the instruction address in the old PSW is correct.

Failing-Storage Address Valid (FA): Bit 24, when one, indicates that a correct failing-storage address has been stored. The presence and extent of the capability to indicate the failing-storage address depend on the model. When no storage errors are reported, that is, bits 16-18 of the machine-check interruption code are zeros, the failing-storage address is meaningless, even though it may be indicated as valid.

Region Code Valid (RC): Bit 25, when one, indicates that a correct region code has been stored. The presence of the region code depends on the model.

Floating-Point Registers Valid (FP): Bit 27, when one, indicates that the contents of the floating-point register save area reflect the correct state of the floating-point registers at the point of interruption. When the floating-point feature is not installed, this bit is set to zero.

General Registers Valid (GR): Bit 28, when one, indicates that the contents stored in the general register save area reflect the correct state of the general registers at the point of interruption.

Control Registers Valid (CR): Bit 29, when one, indicates that the contents stored in the control register save area reflect the correct state of the control registers at the point of interruption.

Logout Valid (LG): Bit 30, when one, indicates that the CPU extended logout information was correctly stored.

Storage Logical Validity (ST): Bit 31, when one, indicates that the contents of those storage locations which are modified by the instruction processing stream contain the correct information relative to the point of interruption. That is, all stores prior to the point of interruption are completed, and all stores, if any, beyond the point of interruption are suppressed. When a store prior to 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 is preserved as invalid.

CPU Timer Valid: Bit 46, when one, indicates that the CPU timer is not in error and that the contents stored in the CPU-timer save area (location 216) reflect the correct state of the CPU timer at the time the interruption occurred.

Clock Comparator Valid: Bit 47, when one, indicates that the clock comparator is not in error and that the contents stored in the clock-comparator save area (location 224) reflect the correct state of the clock comparator.

Programming Note

The validity bits must be used in addition to the subclass indication and time-of-occurrence bits in order to determine the extent of the damage caused by the machine-check condition. The four PSW validity bits, the three register validity bits, the two timing facility validity bits, and the storage logical validity bit must all be ones in addition to one of the following in order to indicate that no damage has yet occurred to the system:

- All of the damage subclass bits (0, 1, 3, 4, 5) are zeros.
- Instruction processing damage is the only damage subclass bit which is one, the backed-up bit is one, and the delayed bit is zero.

Machine-Check Extended Logout Length

Bits 48-63 of the machine-check interruption code contain a 16-bit binary value indicating the length in bytes of the information most recently stored in the extended logout area, starting at the location specified by the machine-check extended logout pointer. When no extended logout has occurred, this field is set to zero.

Programming Note

When asynchronous machine-check extended logouts are permitted (control register 14, bit 8 is one), more than one extended logout may have occurred. The length stored on interruption does not necessarily indicate the longest logout which has occurred.

Machine-Check Control Registers

Control Register 14

CS	SL	IL		RM	DM	EM	WM	AL	FL	
0			3	4						10

Control register 14 contains mask bits that specify whether certain conditions can cause machine-check interruptions and control bits that determine when a logout may occur. With the exception of bit 0, which is provided on all models, each of the bits is necessarily provided only if the associated function is provided.

Check-Stop Control

The check-stop control bit (CS), which is bit 0 of control register 14, controls the system action taken when an exigent machine-check condition occurs under one of the following two conditions:

1. When the CPU is disabled for machine-check interruptions (that is, PSW bit 13 is zero).
2. When a second exigent machine-check condition occurs during the process of storing the machine-check interruption code, storing the machine-check old PSW, or fetching the machine-check new PSW during a machine-check interruption.

If the check-stop control bit is one and either condition occurs, the machine enters the check-stop state; if the check-stop control bit is zero, the machine may attempt to continue or may enter the check-stop state, depending on the type of error and the model. The check-stop control bit is initialized to one. If damage occurs to control register 14, the check-stop control bit is assumed to be one.

Logout Controls

Synchronous Machine-Check Extended Logout Control (SL): Bit 1 of control register 14 controls the logout action during a machine-check interruption. If the bit is one, the machine-check extended logout area may be changed during the interruption; if the bit is zero, the area may be changed only under control of the asynchronous machine-check extended logout control bit (bit 8 of control register 14). Bit 1 of control register 14 is initialized to one.

Input/Output Extended Logout Control (IL): Bit 2 of control register 14, when one, permits channel logout into the I/O extended logout area as part of an I/O interruption. When the I/O extended logout mask is zero, I/O extended logouts cannot occur. This bit is initialized to zero.

Asynchronous Machine-Check Extended Logout Control (AL): Bit 8 of control register 14, in conjunction with PSW bit 13, controls asynchronous change of the machine-check extended logout area. When this bit and PSW bit 13 are both ones, the machine may change the machine-check extended logout area at any time. This bit is initialized to zero.

Asynchronous Fixed Logout Control (FL): Bit 9 of control register 14, when one, permits the fixed logout area to be changed at any time. When this bit is zero, the fixed logout area may be changed only during a machine-check interruption or during an I/O interruption. This bit is initialized to zero.

Programming Notes

The maximum logout information is obtained by setting both the synchronous and asynchronous machine-check extended logout control bits to ones. Both of these bits must be zeros to prevent any changes to the machine-check extended logout area. When asynchronous machine-check extended logout is allowed, use of the machine-check extended logout area may produce unpredictable results.

When the asynchronous fixed logout control bit is one, program use of the fixed logout area should be restricted to the fetching of data from this area. Store operations or channel programs reading into the fixed logout area may cause machine checks or undetected errors if the store occurs during CPU retry. *Note that this is an exception to the rule that programming errors do not cause machine-check indications.*

Machine-Check Subclass Masks

Bits 4-7 of control register 14, in conjunction with PSW bit 13, control various machine-check subclass conditions. When PSW bit 13 is one and the subclass mask 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.

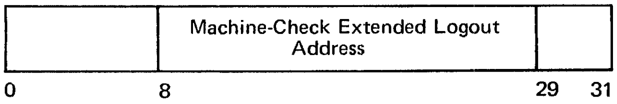
Recovery Report Mask (RM): Bit 4 of control register 14 controls recovery-interruption conditions. This bit is initialized to zero.

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: timer damage, timing facility damage, and external damage. This bit is initialized to one.

Warning Mask (WM): Bit 7 of control register 14 controls all warning conditions. This bit is initialized to zero.

Control Register 15



Bits 8-28 of control register 15, with three low-order zeros appended, specify the starting location of the machine-check extended logout area. The contents of control register 15 are initialized by setting bit 22 to one and all other bits to zeros, which specifies a starting address of 512 (decimal). The machine-check extended logout address is a real address.

When a model provides machine-check extended logout, control register 15 is implemented.

Programming Note

The availability and extent of the machine-check extended logout area differs among models and, for any particular model, may depend on the features or engineering changes installed. In order to provide for such variations, the program should determine the extent of the logout by means of STORE CPU ID whenever a storage area for the extended logout is to be assigned. A length of zero in the MCEL field indicates that no MCEL is provided.

Summary of Machine-Check Masking

Subclass Condition		Subclass Mask	Action When CPU Disabled for Condition	
			Check-Stop Control = 0	Check-Stop Control = 1
SD	System Damage	–	P*	Check stop
PD	Instruction Processing Damage	–	P*	Check stop
TD	Timer Damage	EM	P	P
CD	Timing Facilities Damage	EM	P	P
SR	System Recovery	RM	Y	Y
ED	External Damage	EM	P	P
DG	Degradation	DM	P	P
W	Warning	WM	P	P

Explanation:

P Indication held pending.

Y Indication may be held pending or may be discarded.

* In this situation, the system integrity may have been lost, and the system cannot be considered dependable.

Machine-Check Condition Masking

PSW Bit 13	SMCEL Control CR 14 Bit 1	AMCEL Control CR 14 Bit 8	Machine-Check Extended Logout Action
0	X	X	No MCEL
1	0	0	No MCEL
1	1	0	MCEL may occur only during machine-check interruption.[1]
1	0	1	MCEL may occur at any time.[2]
1	1	1	MCEL may occur at any time.

AFL Control	Fixed Logout Action
0	CPU portion of fixed logout area may be changed only during machine-check interruption.[1]
1	CPU portion of fixed logout area may be changed at any time.

Explanation:

X Indicates the same action occurs whether the bit is zero or one.

[1] Logout prior to instruction retry is not permissible in this state even though recovery reports are enabled.

[2] In some models the AMCEL mask bit is ignored, and no logout occurs in this state.

Machine-Check Logout Control

Bit	Description	Control Register 14 Bit Position	State of Bit on Initial Program Reset
CS	Check-stop control	0	1
SL	Synchronous MCEL mask	1	1
IL	IOEL control	2	0
RM	Recovery report mask	4	0
DM	Degradation report mask	5	0
EM	External damage report mask	6	1
WM	Warning mask	7	0
AL	Asynchronous MCEL control	8	0
FL	Asynchronous fixed logout control	9	0

Machine-Check Control Register Bits

Input/Output Operations

Contents

Attachment of Input/Output Devices . . . 186
Input/Output Devices . . . 186
Control Units . . . 187
Channels . . . 187
Modes of Operation . . . 187
Types of Channels . . . 188
System Operation . . . 189
Compatibility of Operation. . . . 191
Control of Input/Output Devices . . . 191
Input/Output Device Addressing . . . 192
States of the Input/Output System . . . 192
Resetting of the Input/Output System . . . 194
I/O System Reset . . . 195
I/O Selective Reset . . . 195
Effect of Reset on a Working Device . . . 195
Reset Upon Malfunction . . . 195
Condition Code . . . 195
Instruction Formats . . . 197
List of Instructions . . . 197
CLEAR I/O . . . 198
HALT DEVICE . . . 199
HALT I/O . . . 202
START I/O . . . 204
START I/O FAST RELEASE . . . 204
STORE CHANNEL ID . . . 206
TEST CHANNEL . . . 207
TEST I/O . . . 208
Input/Output Instruction Exception Handling . . . 210
Execution of Input/Output Operations . . . 210
Blocking of Data . . . 210
Channel Address Word . . . 210
Channel Command Word . . . 211
Command Code . . . 212
Designation of Storage Area . . . 212
Chaining . . . 213
Data Chaining . . . 213
Command Chaining . . . 215
Skipping . . . 215
Program-Controlled Interruption . . . 215
Channel Indirect Data Addressing . . . 216
Commands . . . 217
Write . . . 218
Read . . . 218
Read Backward . . . 218
Control . . . 219
Sense . . . 219
Transfer in Channel . . . 221
Command Retry . . . 221
Conclusion of Input/Output Operations . . . 222
Types of Conclusion . . . 222
Conclusion at Operation Initiation . . . 222
Immediate Operations . . . 222
Conclusion of Data Transfer . . . 223
Termination by HALT I/O or HALT DEVICE . . . 224
Termination by CLEAR I/O . . . 225
Termination Due to Equipment Malfunction . . . 226
Input/Output Interruptions . . . 226
Interruption Conditions . . . 226

Conclusion of Input/Output Operations (continued)

Priority of Interruptions 227
Interruption Action 228
Channel Status Word 228
Unit Status Conditions 229
Attention 229
Status Modifier 229
Control Unit End 230
Busy 230
Channel End 231
Device End 232
Unit Check 232
Unit Exception 233
Channel Status Conditions 233
Program-Controlled Interruption 233
Incorrect Length. 233
Program Check 234
Protection Check 234
Channel Data Check. 235
Channel Control Check 235
Interface Control Check 235
Chaining Check 235
Contents of Channel Status Word 235
Information Provided by Channel Status Word 235
Protection Key 236
Command Address 236
Count 236
Status 237
Channel Logout 238
I/O Communications Area 239

The transfer of information to or from main storage, other than to or from the central processing unit or by means of the direct control path, 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 manual 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) or Systems Reference Library (SRL) publication for the particular device.

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), 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, the IBM 2702 Transmission Control handles transmission of information between the data processing system and a remote station, and its input and output are signals on a transmission line. Furthermore, in this latter case, the 2702 may be time-shared for concurrent operation of a number of remote stations, and the 2702 is distinguished as a particular I/O device only during the time period associated with the operation on the corresponding remote station.

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.

In most configurations, communication between the control unit and the channel takes place over the I/O interface. The control unit accepts control signals from the channel, controls the timing of data transfer over the I/O interface, 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 user's point of view, most functions performed by the control unit can be merged with those performed by the I/O device. Therefore, this manual 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 sharing of the control unit among a number of devices affects the execution of I/O operations.

Channels

The 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.

The channel provides a standard interface for connecting different types of I/O devices to the CPU and to main 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 over the interface with main-storage cycles. To accomplish this, the channel maintains and updates an address and a count that describe the destination or source of data in main storage. Similarly, when an I/O device provides signals that should be brought to the attention of the program, the channel transforms the signals to information that can be used in the CPU.

The 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 main-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 main 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 main storage.

Alternatively, the system may use the facilities of the CPU for controlling I/O devices. When the CPU and the channel, or the CPU, channel, and control unit, 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.

Modes of Operation

An I/O operation occurs in one of two modes: burst or byte interleave.

In burst mode, the I/O device monopolizes the I/O interface and channel and stays logically connected to the channel for the transfer of a burst of information. No other device can communicate over the interface 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 a channel status condition 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 tape, for not more than approximately one-half minute. Equipment malfunction may be indicated when an absence of data transfer exceeds this time.

In byte-interleave 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 over the interface. 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.

A short burst of data can be handled in either byte-interleave or burst mode. The distinction between a short burst occurring in the byte-interleave mode and an operation in the burst mode is in the length of the bursts. 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-interleave mode. Some control units have a manual switch for setting the mode of operation.

Operation in burst and byte-interleave 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-interleave 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 by time-out. A new I/O operation is initiated only after the channel has serviced all outstanding requests for data transfer from devices previously placed in operation.

Types of Channels

A system can be equipped with three types of channels: selector, byte-multiplexer, and block-multiplexer. Channels are classified according to their capability for multiplexing and to the modes of operation they can sustain. A byte-multiplexer channel can operate either in byte-interleave mode or in burst mode, depending on the device. A selector channel operates only in burst mode and allows no multiplexing. A block-multiplexer channel operates only in burst mode and can allow multiplexing between blocks.

The channel facilities required for sustaining a single I/O operation are termed a *subchannel.* The subchannel consists of the channel 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.

The selector channel 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. The selector channel cannot perform any multiplexing and therefore can be involved in only one data-transfer operation 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.

The byte-multiplexer channel contains multiple subchannels and can operate in either byte-interleave or burst mode. In byte-interleave mode, more than one device may operate concurrently, each on a separate subchannel. In burst mode, only one device on the channel may be transferring data. 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-interleave mode and partially in the burst mode.

The block-multiplexer channel has multiple subchannels and always forces the I/O device to transfer data in burst mode. When multiplexing is allowed on the block-multiplexer channel, the burst is forced to extend only over the block of data. Multiplexing is permitted between blocks of data when command chaining is specified or when command retry is performed. Whether or not multiplexing occurs between blocks depends on the state of the block-multiplexing control bit, the design of the I/O device, and, on some models, whether the subchannel is shared or nonshared.

The multiplexing capability of a block-multiplexer channel is under control of the block multiplexing control bit, bit 0 of control register 0. When this bit is zero, multiplexing is inhibited, and 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 interruption conditions.

When a multiplexer channel is transferring data in burst mode, the subchannel associated with the burst operation monopolizes the data-transfer facilities of the channel. Other subchannels on the multiplexer channel cannot respond to requests from devices until the burst is completed.

Subchannels on a multiplexer channel may be either *nonshared* or *shared.*

A subchannel is referred to as nonshared if it is associated and can be used only with 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 a 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 Note

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.

System Operation

Input/output operations are initiated and controlled by information with three types of formats: instructions, channel command words (CCWs), and orders. *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 main 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 conditions for the execution of an operation at the device.

Functions peculiar to a device, such as rewinding tape or positioning the access mechanism on a disk drive, are specified by *orders.* Orders are decoded and executed by I/O devices. The control information specifying an order may appear in the modifier bits of a control-CCW command code, may be transferred to the device as data during a control or write operation, or may be made available to the device by other means.

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 main storage. The CAW contains the protection key and designates the location in main 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, under certain conditions, 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 over the interface, 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, under certain conditions, 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 to a particular CCW in the channel program.

To complement the dynamic address translation facility available in the CPU, which can make data stored in more than one noncontiguous page of main storage appear as one storage area, channel indirect data addressing is available. A flag in the CCW specifies that an indirect-data-address list is to be used to designate the storage areas for that CCW. Each time the boundary of a 2,048-byte block of storage is reached, the list is referenced to determine the next block of storage to be used. By extending the storage addressing capabilities of the channel, channel indirect data addressing permits essentially the same CCW sequences to be used for a program running with dynamic address translation in the CPU that would be used if it were operating with equivalent contiguous real storage.

The concluding of an I/O operation normally is indicated by two conditions: channel end and device end. The channel-end condition indicates that the I/O device has received or provided all data associated with the operation and no longer needs channel facilities. The device-end signal indicates that the I/O device has concluded execution of the operation. The device-end condition can occur concurrently with the channel-end condition or later.

Operations that keep the control unit busy after releasing channel facilities may, under certain conditions, cause a third type of signal. This signal, called 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.

The conditions signaling the concluding 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. In either case, these conditions cause storing of the CSW, which contains additional information concerning the execution of the operation. At the time the channel-end condition is generated, the channel identifies to the program the last CCW used and provides its residual byte count, thus indicating the extent of main storage used. Both the channel and the device can provide indications of unusual conditions with channel end. The control-unit-end and device-end conditions can be accompanied by error indications from the device.

Facilities are provided for the program to initiate 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 by means of the same sequence of signals over the I/O interface as the first command specified by START I/O or START I/O FAST RELEASE. The ending signals occurring at the concluding of an operation caused by a CCW specifying command chaining are not made available to the program when another operation is initiated by the command chaining; the channel continues execution of the channel program. If, however, an unusual condition has been detected, the ending signals cause suppression of command chaining and a termination of the channel program.

Conditions that initiate I/O interruptions are asynchronous to activity in the CPU, and more than one condition can occur at the same time. The channel and the CPU establish priority among the conditions so that only one interruption request is processed at a time. The conditions are preserved in the I/O devices or subchannels until accepted by the CPU.

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 channel-end condition for 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 the control-unit-end condition when it becomes free.
4. The I/O device is busy from the initiation of the first operation until the device-end condition associated with the operation is accepted or cleared by the CPU.

A pending device-end condition causes the associated device to appear busy, but normally does not affect the state of any other part of the system. A pending control-unit-end condition normally blocks communications through the control unit to any device attached to it, and a pending channel-end condition 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 main-storage area. Furthermore, since in most instances the channel may share main storage with the CPU and other channels, activity in the rest of the system affects the accessibility of main 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 instruction 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 I/O address is a 16-bit binary number and consists of two parts: a channel address in the eight high-order bit positions and a device address in the eight low-order bit positions.

The channel-address field provides for identifying up to 256 channels. Channel 0 is a byte-multiplexer channel; channels numbered 1-255 may be either multiplexer or selector channels.

The number and type of channels 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 byte-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. For example, the IBM 3830 Storage Control Model 2, with four disk units installed, responds to all of the 16 addresses within the set assigned to it. 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.

Except for the rules described, the assignment of channel and device addresses is arbitrary. 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 following table. The name of the state is followed by its abbreviation and a brief definition.

A portion of the I/O system that is available, interruption-pending, or working is called "operational." A portion of the I/O system 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 pend-

Name	Abbreviation and Definition	
Channel		
Available	A	None of the following states
Interruption pending	I	Interruption immediately available from channel
Working	W	Channel operating in burst mode
Not operational	N	Channel not operational
Subchannel		
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
I/O Device		
Available	A	None of the following states
Interruption pending	I	Interruption condition pending in device
Working	W	Device executing an operation
Not operational	N	Device not operational

Input/Output System States

ing 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.

The device referred to in the preceding table 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 on the basis of the states of the channel and subchannel, the components further removed are not interrogated. Thus, ten composite states are identified as conditions for the execution of the I/O instruction. Each composite state is identified in the following discussion by three alphabetic characters; the first character position identifies the state of the channel, the second identifies the state of the subchannel, and the third refers to the state of the device. Each character position can contain A, I, W, or N, denoting the state of the component. The symbol X in place of a letter 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.

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 a pending interruption condition. These situations are possible:

1. The device is executing an operation, such as rewinding tape or seeking on a disk file, after signaling the channel-end condition.
2. The control unit associated with the device is executing an operation, such as backspacing file on a magnetic tape unit, after signaling the channel-end condition.
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 switched off the I/O interface. 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 (for example, the second set of lines on the IBM 2702 Transmission

Control). See also "Input/Output Device Addressing."

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 (for example, access mechanism 7 on the IBM 3830 Storage Control that has only access mechanisms 0-3 installed) 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 "Unit Check."

Interruption Pending in Subchannel (AIX): The addressed channel is available. An interruption condition is pending in the addressed subchannel because of the concluding of the portion of the operation involving the use of channel facilities. The subchannel is in a position to provide information for a complete 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 concluded operation, 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 the 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 a pending 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 a pending interruption condition. A channel with a pending 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 "Interruption Conditions.")

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 reached. The states of the addressed device and, in the case of the multiplexer channel, of the subchannel are not significant. Depending on the channel type and system model, 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 and SRL publications for the device.

I/O System Reset

The I/O system reset is performed when the CPU to which the channel is configured performs a program reset, initial-program reset, system-clear reset, or power-on reset, when a power-on sequence is performed by the channel, and, under certain conditions, when a channel detects equipment malfunctions.

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

If the device is currently communicating over the I/O interface, 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

The type of reset executed in the channel depends on the type of malfunction and the channel. When a reset occurs upon malfunction, the program is alerted by an interruption or, when the malfunction is detected during the execution of an I/O instruction, by the setting of the condition code. In either case the CSW identifies the condition. The device addressed by the I/O instruction is not necessarily the device that is reset. In channels sharing equipment with the CPU, malfunctioning detected by the channel may be indicated by a machine-check interruption, which may or may not be followed by an I/O interruption. When no I/O interruption takes place, a CSW is not stored, and a device is not identified. The method of identifying malfunctions depends on the model.

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 channel has performed the instruction and, if not, the reason for the rejection. In the case of START I/O FAST RELEASE executed independently of the device, a condition code 0 may be set that is later superseded by a deferred condition code stored in the CSW. Branch-on-condition operations following an operation that sets the condition code use the code for decision-making.

The following table lists the conditions identified and the corresponding condition codes for each I/O instruction. The states of the I/O system and associated abbreviations were previously defined in "States of the Input/Output System." The digits in the table represent the decimal value of the code. The instructions START I/O and START I/O FAST RELEASE can set 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.

The available condition is indicated only when no errors are detected during the execution of the I/O instruction.

When a subchannel on the multiplexer channel contains a pending interruption condition (state AIX), the I/O device associated with the concluded operation normally is in the interruption-pending state. When the channel detects during the execution of TEST I/O that the device is not operational, condition code 3 is set. Similarly, condition code 3 is set when HALT I/O or HALT DEVICE is addressed to a subchannel in the working state (state AWX), but the device turns out to be not operational.

Error conditions, including all equipment or programming errors detected by the channel or the I/O device during execution of the I/O instruction, generally cause the CSW to be stored. On some models, however, a channel equipment error may cause a machine-check interruption but no I/O interruption to occur, with no storing of the CSW. Three types of

		Condition Code Settings						
Conditions	I/O State	SIO SIOF	TIO	CLRIO[1]	HIO	HDV	TCH	STIDC
Available	AAA	0,1*@	0	0	1*	1*	0	0
Interruption pending in device	AAI	1*@	1*	0	1*	1*	0	0
Device working	AAW	1*@	1*	0	1*	1*	0	0
Device not operational	AAN	3@	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	##
Channel not operational	NXX	3	3	3	3	3	3	3

Explanation:

[1] The entries in this column indicate the condition-code setting when the CLRIO function is executed.

* 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.

***A 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, a 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.

+ The condition code depends on the I/O interface sequence, the channel type, and the system model. If the channel ascertains that the device received the signal to terminate, a condition code 1 is set and the CSW stored. Otherwise, a condition code 2 is set.

## When the channel is unable to store the channel ID because of the working or interruption pending state, a condition code 2 is set. If the working or interruption pending state does not preclude storing the channel ID, a 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 a pending 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

errors can occur:

Channel Equipment Error: The channel can detect the following equipment errors during execution of START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, and HALT DEVICE:

1. The device address that the channel received on the interface during initial selection either has a parity error or is not the same as the one the channel sent out. Some device other than the one addressed may be malfunctioning.
2. The unit-status byte that the channel received on the interface during initial selection has a parity error.
3. A signal from the I/O device occurred at an invalid time or had invalid duration.
4. The channel detected an error in its control equipment. (This is also true for STORE CHANNEL ID and TEST CHANNEL.)

The channel may perform an I/O selective reset or an I/O system reset or may generate a halt signal, depending on the type of error and the model. If a CSW is stored, channel control check or interface control check is indicated, depending on the type of error.

Channel Programming Error: The channel can detect the following programming errors during execution of START I/O or START I/O FAST RELEASE. All of the error conditions are indicated during START I/O, and during START I/O FAST RELEASE when it is executed as START I/O, by the condition-code setting and by the status portion of the CSW. When the SIOF function is performed, the first two error conditions are indicated as for START I/O, and the remaining conditions are indicated in a subsequent interruption.

1. Invalid CCW address specification in CAW.
2. Invalid CAW format.
3. Invalid CCW address in CAW.
4. First-CCW location protected against fetching.
5. First CCW specifies transfer in channel.
6. Invalid command code in first CCW.
7. Invalid count in first CCW.
8. Invalid format for first CCW.
9. If channel indirect data addressing (CIDA) was specified, an invalid data address specification in the first CCW.
10. If CIDA was specified, an invalid data address in the first CCW.
11. If CIDA was specified, the first-IDAW location protected against fetching.
12. If CIDA was specified, invalid format for the first IDAW.

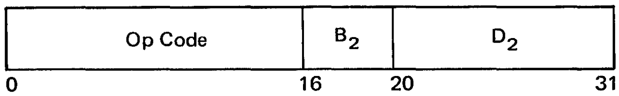
The CSW indicates program check, except for items 4 and 11, for which protection check is indicated.

Device Error: Programming or equipment errors detected by the device during the execution of START I/O, or START I/O FAST RELEASE are indicated by unit check or unit exception in the CSW.

The conditions responsible for unit check and unit exception for each type of I/O device are detailed in the SL or SRL publication for the device.

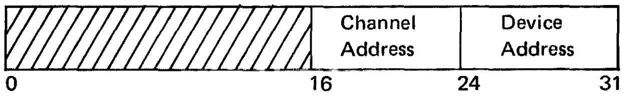
Instruction Formats

All I/O instructions use the following S format:



Except for STORE CHANNEL ID, bit positions 8-14 of these instructions are ignored. Bit position 15 is ignored by the instruction TEST CHANNEL but is decoded as part of the operation code for START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, and HALT DEVICE.

The second-operand address specified by the B_2 and D_2 fields is not used to designate data, but instead is used to identify the channel and I/O device. Address computation follows the rules of address arithmetic. The address has the following format:



Bit positions 0-7 are not part of the address. Bit positions 8-15, which constitute the high-order portion of the three-byte address, are ignored. Bit positions 16-23 of the sum contain the channel address, while bit positions 24-31 identify the device on the channel and, additionally in the case of the multiplexer channel, the subchannel.

All I/O instructions cause a serialization function to be performed. CPU operation is delayed until all previous CPU accesses to main storage have been completed, as observed by channels and other CPUs, and then the addressed channel is selected. No subsequent instructions or their operands are accessed until the execution of the I/O instruction has been completed.

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the IBM System/370 assembly language are shown with each instruction. In the case of START I/O, for example, SIO is the mnemonic and $D_2(B_2)$ the operand designation.

List of Instructions

The mnemonics, format, and operation codes of the I/O instructions follow. The table 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.

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, and STORE CHANNEL ID are 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

CLRIO $D_2(B_2)$ [S]

9D01	B_2	D_2
0	16	20 31

Either a TIO or CLRIO function is performed, depending on the channel and the block-multiplexing control: control register 0, bit 0. The TIO function is performed when the CLRIO function is not implemented by the channel or 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 in the interruption-pending state 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 instruction causes the device to be 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 in the available or interruption-pending state, and the subchannel contains an interruption-pending condition for the addressed device or is working with the addressed device. The protection-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 protection-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

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	9D00*

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.

Input/Output-Instruction Summary

the addressed device or is working with the addressed device. The protection-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 condition 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 graphically as follows. The condition code set when CLEAR I/O causes the TIO function to be performed is shown graphically in the definition of the instruction TEST I/O.

Programming Note

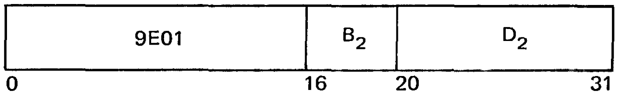
Since some channels cause a condition code 2 to be set when the instruction is received and the channel is in the working state, it may be useful to issue a halt instruction and then CLEAR I/O to the desired address. Using HALT DEVICE will ensure that condition code 2 is received on the CLEAR I/O only 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.

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.

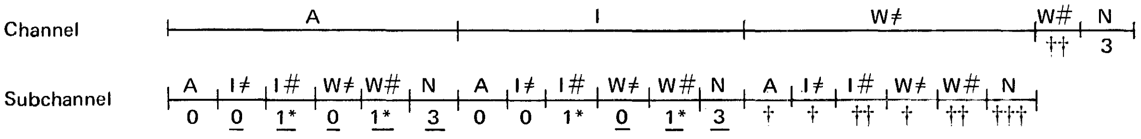
The combination of HALT DEVICE followed by CLEAR I/O can be used to clear out all activity on a channel by executing the two instructions for all device addresses on the channel.

Halt Device

HDV $D_2(B_2)$ [S]



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. HALT DEVICE is executed only when the CPU is in the supervisor state. Bits 8-14 of the instruction are ignored.



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 conditions that can occur only on the multiplexer channel.

Condition Codes Set by CLEAR I/O

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 in the interruption pending state 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 be set up to signal termination of the device operation the next time the device requests or offers a byte of data, if any. Chaining, if indicated in the subchannel, is suppressed. If the subchannel is available, the subchannel is not affected.

When the channel is either available or in the interruption pending state with the subchannel either working with a device other than the one addressed or in the interruption pending state, no action is taken.

When the channel is working in burst mode with the addressed device, data transfer across the I/O interface for the operation is immediately terminated, and the device immediately disconnects from the channel. Chaining, if indicated in the subchannel, is suppressed.

When the channel is working in burst mode with a device other than the one addressed, and the subchannel is available, in the interruption pending state, 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 is set up to signal termination of the device operation the next time the device requests or offers a byte of data, if any. Chaining, if indicated in the subchannel, 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 in the interruption pending state, 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 a status byte 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 conditions numbered below occurs, HALT DEVICE causes the 16-bit unit- 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 under what conditions the I/O operation at the addressed subchannel is terminated. The extent of data transfer, and the conditions 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 conditions 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 the busy and status modifier bits. 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 error condition. 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 condition 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 HALT DEVICE causes data transfer over the I/O interface 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 concluded. Concluding 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 across the I/O interface, 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 across the I/O interface, 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

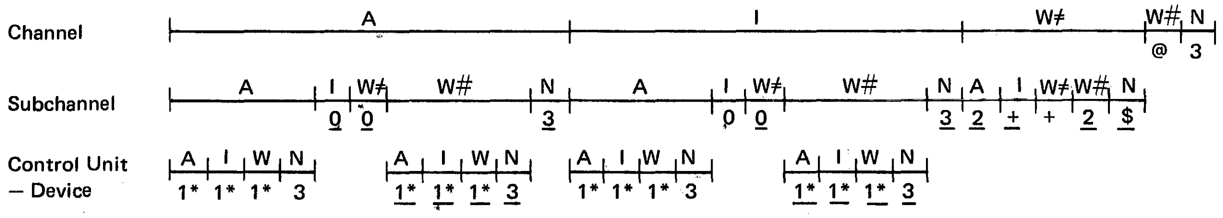
The condition code set by HALT DEVICE for all possible states of the I/O system is shown graphically. See "States of the Input/Output System" for a detailed definition of the A, I, W, and N states.

Programming Note

Some selector and byte-multiplexer channels designed prior to the defining of HALT DEVICE (for example, the 2860), will execute HALT DEVICE as HALT I/O. A program can ensure complete compatibility between HALT DEVICE and HALT I/O on such channels by observing the following conventions:

1. On a byte-multiplexer channel, do not issue HALT DEVICE to a multiplexing device when a burst operation is in progress on the channel.
2. On a byte-multiplexer channel, do not issue HALT DEVICE to a device on a shared subchannel while that subchannel is working with a device other than the one addressed.
3. On a selector channel in the working state, do not issue HALT DEVICE to any device other than the one with which the channel is working.

The execution of HALT DEVICE always causes data transfer across the I/O interface for the addressed device 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 conditions under which the execution of a HALT DEVICE for the same address will cause the control unit to be signaled to terminate.



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 and the conditions in the channel. Condition code 1 (with CSW stored) can be set only if the control unit has received the signal to terminate.

+ In the W≠IX and W≠W≠X states, either condition code 0 or 2 may be set, depending on the channel and the conditions in the channel.

$ In the W≠NX state, either condition code 2 or 3 may be set, depending on the channel type and system model.

Note: Underscored condition codes pertain to conditions that can occur only on the multiplexer channel.

Condition Codes Set by HALT DEVICE

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.

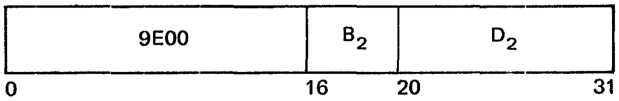
Condition Code 1 with Zeros in the Status Field of the CSW indicates that the addressed device was selected and signaled to terminate the current operation, if any.

Condition Code 2 indicates that the control unit cannot be signaled until the end of a busy condition in the channel. The end of the busy condition 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 to allow HALT DEVICE to signal the control unit to terminate.

Halt I/O

HIO $D_2(B_2)$ [S]



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. The instruction HALT I/O is executed only when the CPU is in the supervisor state. 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 in the interruption pending state, 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 provides 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-interleave 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 has a pending interruption condition, with the channel in the available or interruption pending state, HALT I/O does not cause any action.

When any of the following conditions occurs, HALT I/O causes the status portion, bit positions 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 under what conditions the I/O operation at the addressed subchannel is concluded. The extent of data transfer, and the conditions 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 has been 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 has pending 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, has been set up to signal termination the next time the device identifies itself. The CSW unit-status field contains the busy and status modifier bits. 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 condition 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 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 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, 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 or SRL publication for the device. The HALT I/O signal has no effect on devices that are not in the working state or are executing an operation of a fixed duration, 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. Pending attention or device end conditions at the device are not reset.

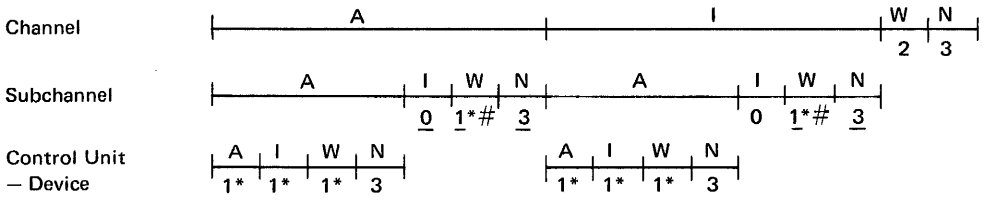
Program Exceptions:

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 graphically as follows. See "States of the 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
*	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 conditions 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 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

SIO $D_2(B_2)$ [S]

9C00	B_2	D_2
0	16	20 31

Start I/O Fast Release

SIOF $D_2(B_2)$ [S]

9C01	B_2	D_2
0	16	20 31

A write, read, read backward, control, or sense operation is initiated with the addressed I/O device and subchannel. The instruction is executed only when the CPU is in the supervisor state. 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: control register 0, bit 0. The SIO function causes the operation to be initiated only after the device is selected. The SIOF function causes the operation to be initiated independently of the device. The instruction START I/O always causes the SIO function to be performed, as does START I/O FAST RELEASE when block multiplexing is not specified. When block multiplexing is specified, START I/O FAST RELEASE, depending on the channel, may 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 protection key for the subchannel and the address of the first CCW. This CCW specifies the operation to be performed, the main-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 is in the interruption pending state, and errors or exceptional conditions 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 condition 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 is in the interruption-pending state, and errors or exceptional conditions 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 condition 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 either available or in the interruption-pending state and the subchannel is available before the execution of the instruction, the following conditions cause a CSW to be stored, in a manner determined by 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 condition causes the same action as for the SIO function. The remaining conditions 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 condition. The channel-end and busy bits are off, 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 pending interruption conditions, are on. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation has not been 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, if CIDA is specified, with the first IDAW; or, for the SIOF function, the channel detects an equipment

error after completion of the instruction. The CSW identifies the error condition. The channel-end and busy bits are off, 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 pending interruption conditions, are on. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation has not been 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 (1) no command chaining is specified and no command retry occurs, or (2) chaining is suppressed because of unusual conditions detected during the operation. The CSW contains the channel-end bit and any other indications provided by the channel or the device. The busy bit is off. The I/O operation has been 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. The CSW contains the PCI bit if specified in the first CCW.
4. The I/O device contains a pending interruption condition, or the control unit contains a pending interruption condition for the addressed device. The CSW unit-status field contains the busy bit, identifies the interruption condition, and may contain other bits provided by the device or control unit. The interruption condition is cleared. The channel-status field indicates any error conditions detected by the channel and contains the PCI bit if specified in the first CCW.
5. The I/O device or the control unit is executing a previously initiated operation, or the control unit has pending an interruption condition associated with a device other than the one addressed. The CSW unit-status field contains the busy bit or, if the control unit is busy, the busy and status-modifier bits. The channel-status field indicates any error conditions detected by the channel and contains the PCI bit if specified in the first CCW. When the SIOF function is performed, the control unit busy condition may cause the same action as the SIO function.
6. The I/O device or control unit detected an equipment or programming error during the initiation, or the addressed device is in the not-ready state. The CSW identifies the error condition. The channel-end and busy bits are off, unless the device was found to be busy, in which case the busy bit, as well as any bits indicating pending interruption conditions, are on. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation has not been initiated. No interruption conditions are generated at the I/O device or subchannel. The CSW contains the PCI bit if specified in the first CCW.

When the SIO or SIOF function cannot be executed because of a pending logout condition 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 SIOF function causes condition code 0 to be set and subsequently a condition 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. 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.

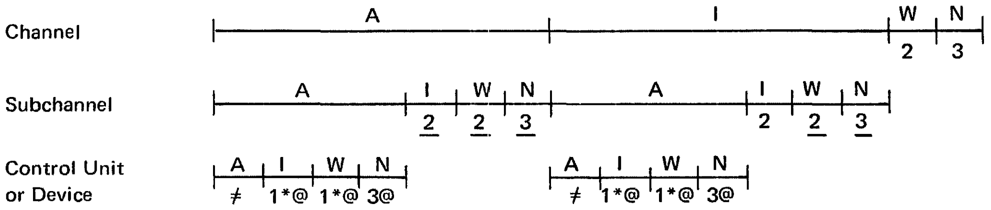
Program Exceptions:

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 graphically as follows. See "States of the 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
*	CSW stored
@	When the SIOF function is performed, condition code 0 is set. The other condition code shown will be specified as a deferred condition code.

Note: Underscored condition codes pertain to conditions that can occur only on the multiplexer channel.

≠
- 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 condition has been detected by the control unit or device, for the SIO function condition code 1 is set, and the CSW is stored. For the SIOF function condition code 0 is set, and a deferred condition code 1 interruption condition is generated.

Condition Codes Set by START I/O and START I/O FAST RELEASE

Programming Notes

The advantage of START I/O FAST RELEASE over START I/O is that less CPU time is required for the execution of the instruction. For a START I/O instruction the device must be selected and it must determine if the command and device conditions allow the initiation of the operation prior to the setting of the condition code, which allows the CPU to proceed to the next instruction. When the START I/O FAST RELEASE instruction is used, the condition code is set and the CPU proceeds to its next instruction as soon as the control unit indicates it is capable of communicating with the channel. Thus, the CPU is freed for other activity earlier. A disadvantage, however, is that if a deferred condition code is presented, the resultant CPU execution time may be greater than that required in executing START I/O.

When the channel detects a programming error during execution of the SIO function and the addressed device contains an interruption condition, with the channel and subchannel in the available state, the instruction may or may not clear the interruption condition, depending on the type of error and the model. If the instruction has caused the device to be interrogated, as indicated by the presence of the busy bit in the CSW, the interruption condition has been cleared, and the CSW contains program or protection check, as well as the status from the device.

Two major differences exist between START I/O and START I/O FAST RELEASE:

1. Nonchained immediate commands on certain channels (that is, those which execute START I/O FAST RELEASE independently of the device) result in a condition code 0 for START I/O FAST RELEASE when the block-multiplexing control bit is set to one, whereas condition code 1 is set for START I/O. See also programming note 2 following "Command Retry."

2. Condition code 0 is set by these certain channels for START I/O FAST RELEASE when the block-multiplexing control bit is set to one, even though the addressed device is not available or the command is rejected by the device. The device information will be supplied by means of an interruption.

Store Channel ID

STIDC D2(B2) [S]

B203	B2	D2
0	16	20 31

Information identifying the designated channel is stored in the four-byte field at location 168.

STORE CHANNEL ID is executed only when the CPU is in the supervisor state.

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 format of the information stored at location 168 is:

Type	Channel Model Number
0	4 15

Maximum IOEL Length
16 31

Bits 0-3 specify the channel type. When a channel can operate as more than one type, the code stored identifies the channel type at the time the instruction is executed. The following codes are assigned:

0000	Selector
0001	Byte multiplexer
0010	Block multiplexer

Bits 4-15 identify the channel model. When the channel model is implied by the channel type and the CPU model, zeros are stored in the field.

Bits 16-31 contain the length in bytes of the longest I/O extended logout that can be stored by the channel during an I/O interruption. If the channel never stores logout information using the IOEL pointer, then this field is set to zero.

When the channel detects an equipment malfunction during the execution of STORE CHANNEL ID, the channel causes the status portion, bits 32-47, of the CSW to be replaced by a new set of status bits. With the exception of the channel control check bit (bit 45), which is stored as a one, all bits in the status field are stored as zeros. The contents of the other fields of the CSW are not changed.

When STORE CHANNEL ID cannot be executed because of a pending logout condition which affects the operational capability of the channel, a full CSW is stored. The fields in the CSW are all set to zero, 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.

Program Exceptions:

Privileged operation

Resulting Condition Code:

0 Channel ID correctly stored
1 CSW stored
2 Channel activity prohibited storing ID
3 Not operational

The condition code set by STORE CHANNEL ID for all possible states of the I/O system is shown graphically as follows. See "States of the Input/Output System" for a detailed definition of the A, I, W, and N states.

	A	I	W	N
Channel	0	$	$	3

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

TCH $D_2(B_2)$ [S]

9F00	B_2	D_2
0	16	20 31

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.

The instruction TEST CHANNEL is executed only when the CPU is in the supervisor state.

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 aware of any outstanding interruption conditions from its devices, or is not operational. When the channel is operating in the burst mode and contains a pending interruption condition, the condition code is set as for operation in the burst mode. When none of these conditions exist, the available state is indicated. No device is selected and, on the multiplexer channel, the subchannels are not interrogated.

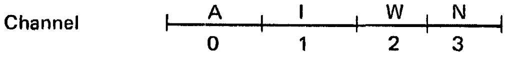
Program Exceptions:

Privileged operation

Resulting Condition Code:

0 Channel available
1 Interruption or logout condition pending 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 graphically as follows. See "States of the Input/Output System" for a detailed definition of the A, I, W, and N states.

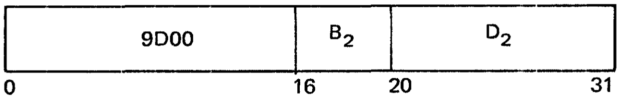


A	Available
I	Interruption pending
W	Working
N	Not operational

Condition Codes Set by TEST CHANNEL

Test I/O

TIO D$_2$(B$_2$) [S]



The state of the addressed channel, subchannel, and device is indicated by setting the condition code in the PSW and, under certain conditions, by storing the CSW. Pending interruption conditions may be cleared. Bits 8-14 of the instruction are ignored.

The instruction TEST I/O is executed only when the CPU is in the supervisor state.

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, on some channels and under certain circumstances, by CLEAR I/O.

When the channel is operating in burst mode and the addressed subchannel contains a pending interruption condition, the TIO function causes condition code 1 or 2 to be set, depending on the channel type and system model. 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 condition in the following paragraph occurs with the channel either available or in the interruption pending state, or, on some channels, in the working state, 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 a pending interruption condition due to a terminated operation at the addressed device. The CSW identifies the interruption condition, and the interruption condition is cleared. The protection key, command address, and count fields contain the final values for the I/O operation, and the status may include other 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 in the working state and has not yet accepted the interruption condition from the device.

When any of the following conditions occurs with the channel either available or in the interruption-pending state, 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 a pending interruption condition or the control unit contains a pending 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 off. 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 a pending interruption condition associated with a device other than the one addressed. The CSW unit-status field contains the busy bit or, if the control unit is busy, the busy and status modifier bits. 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 in the not-ready state and does not have any pending interruption condition. The CSW identifies the error conditions. 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 condition 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 tape control units, for example, are in such a state when they have provided the channel end condition 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 contains the busy and status modifier bits, 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, such as the 2702 Transmission Control, 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 contains the status modifier bit, 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

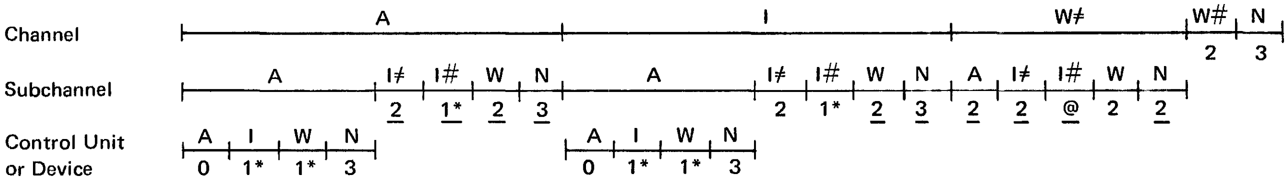
Resulting Condition Code:
0 Available
1 CSW stored
2 Channel or subchannel busy
3 Not operational

The condition code set by the TIO function for all possible states of the I/O system is shown graphically as follows. See "States of the Input/Output System" for a detailed definition of the A, I, W, and N states.

Programming Notes

Disabling the CPU for I/O interruptions provides the program a means of controlling the priority of I/O interruptions selectively by channels. The priority of devices attached on a channel is fixed and cannot be controlled by the program. The instruction TEST I/O permits the program to clear interruption conditions selectively by I/O device.

When a CSW is stored by the TIO function, the interface-control-check and channel-control-check indications may be due to a condition already existing in the channel or due to a condition created by the TIO function. Similarly, presence of the unit check bit in the absence of channel end, control unit end, or device end bits may be due to a condition created by the preceding operation, the not-ready



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≠I #X state, either condition code 1 may be set with the CSW stored, or condition code 2 may be set, depending on the channel and the conditions in the channel.

Note: Underscored condition codes pertain to conditions that can occur only on the multiplexer channel.

Condition Codes Set by TEST I/O

state, or an equipment error detected during the execution of TEST I/O. The instruction TEST I/O cannot be used to clear a pending interruption condition due to the PCI flag while the subchannel is in the working state.

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 conditions detected at this time cause a program interruption. When the interruption occurs, the current PSW is stored as the program old PSW and is replaced by the program new PSW. The interruption code in the old PSW identifies the cause of the 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 occurs.

Blocking of Data

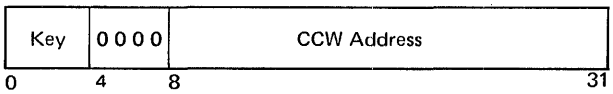
Data recorded by an I/O device may be divided into blocks. The length of a block depends on the device; for example, a block can be a card, a line of printing, or the information recorded between two consecutive gaps on magnetic tape.

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 main storage area is exhausted or the end of the block is reached, whichever occurs first. For some operations, such as writing on a magnetic tape unit or at an inquiry station, blocks are not defined, and the amount of information transferred is controlled only by the program.

Channel Address Word

The channel address word (CAW) specifies the storage protection 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 real location 72 of the CPU issuing the instruction. 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:



The fields in the CAW are allocated for the following purposes:

Protection Key: Bits 0-3 form the protection key for all commands associated with START I/O and START I/O FAST RELEASE. This key is matched with a key in storage whenever a reference is made to main storage during an I/O operation.

CCW Address: Bits 8-31 designate the location of the first CCW in absolute main storage.

Bit positions 4-7 of the CAW must contain zeros. The three low-order bits of the command 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 location protected against fetching or outside the main storage of the particular installation, START I/O and START I/O FAST RELEASE cause the status portion of the CSW to be stored with the protection check or program-check bit on. In this event, the I/O operation is not initiated.

Programming Note

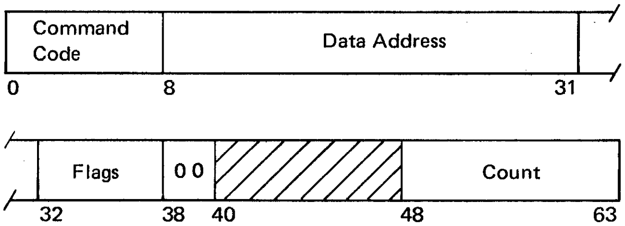
Bit positions 4-7 of the CAW, which presently must contain zeros, may in the future be assigned for the control of new functions. It is therefore recommended that these bit positions not be set to one for the purpose of obtaining an intentional program-check indication.

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 main 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 independently 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 main 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 the location of an eight-bit byte in absolute main 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 an incorrect-length condition is to be indicated to the program. When this bit is one and the CD flag is zero, the incorrect-length indication is suppressed. When both the CC and SLI flags are one, command chaining takes place regardless of the presence of an incorrect-length condition.

Skip (SKIP) Flag: Bit 35, when one, specifies suppression of 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.

Indirect Data Address (IDA) Flag: Bit 37, when one, specifies indirect data addressing. (The flag is valid in both BC and EC modes.)

Count: Bits 48-63 specify the number of eight-bit byte locations in the storage area designated by the CCW.

Bit positions 38-39 of every CCW other than one specifying transfer in channel must contain zeros. Additionally, if indirect addressing is specified, bits 30-31 of the CCW must be zeros, indicating a word boundary, and bits 0-7 of the first entry of the indirect data address list must be zeros. (See "Channel Indirect Data Addressing.") 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, if being executed as START I/O, START I/O FAST RELEASE. 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 38-39 of the CCW, which presently must contain zeros, may in the future be assigned for the control of new functions. It is therefore recommended that these bit positions not be set to one for the purpose of obtaining a 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 high-order bit positions contain modifier bits. The modifier bits specify to the device how the command is to be executed. They may cause, for example, the device to compare data received during a write operation with data previously recorded, and they may specify such conditions as recording density and parity. For the control command, the modifer 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 or SRL 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	0100	Sense
xxxx	1000	Transfer in Channel
mmmm	1100	Read Backward
mmmm	mm01	Write
mmmm	mm10	Read
mmmm	mm11	Control

Whenever the channel detects an invalid command code during the initiation of a command, the program-check condition 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, if being executed as START I/O, START I/O FAST RELEASE. 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

Note: For a description of the storage area associated with a CCW when channel indirect data addressing is invoked, see "Channel Indirect Data Addressing."

The main-storage area associated with an I/O operation is defined by one or more CCWs. A CCW defines an area by specifying the address of the first eight-bit byte to be transferred and the number of consecutive eight-bit 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 main 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 main-storage location available to the channel can be used in the transfer of data to or from an I/O device, provided that the location is not protected against the type of reference. Similarly, the CCWs can be located in any part of available main storage, provided the location is not protected against a fetch-type reference. When the channel attempts to refer to a protected location, the protection check condition is generated, and the device is signaled to terminate the operation.

In the event the channel refers to a location not provided in the system, the program-check condition is generated. When the first CCW designated by the CAW is at a nonexistent location, 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. Invalid data addresses, as well as any invalid CCW addresses detected on chaining or subsequent to the executing of START I/O FAST RELEASE, are indicated to the program with the interruption conditions at the conclusion of the operation or chain of operations.

During an output operation, the channel may fetch data from the main storage before the time the I/O device requests the data. Any number of bytes specified by the current CCW may be prefetched and buffered. When data chaining during an output operation, the channel may fetch the next CCW at any time during the execution of the current CCW. When the I/O operation uses data and CCWs from locations near the end of the available storage, such prefetching may cause the channel to refer to locations that do not exist. Invalid addresses 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 information. 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, the condition is not brought to the attention of the program.

The count field in the CCW can specify any number of bytes up to 65,535. Except for a CCW specifying transfer in channel, where the count field is ignored, the count field may not contain the value zero. Whenever the count field in the CCW initially contains a zero, the program-check condition 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 eight 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 accompanying table.

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.

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 main storage or to the device, data chaining causes the operation to continue, using the storage area designated by the new CCW. The contents of the command-code field of the new CCW are ignored, unless they specify transfer in channel.

Data chaining is considered to occur immediately after the last byte of data designated by the current CCW has been transferred to main storage or to the device. When the last byte of the transfer has been placed in main 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 sends channel end after exhausting the count of the current CCW but before transferring any data to or from the storage area designated by the new CCW, the 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 the channel-end condition 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 the address of the new CCW is invalid, the location is protected against fetching, or programming errors are detected in an intervening transfer-in-channel command. A data

Flags in Current CCW			Action in Channel upon Exhaustion of Count or Receipt of Channel End			
				Regular Operation		
CD	CC	SLI	Immediate 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:

I Count exhausted, end of block at device not reached.

II Count exhausted and channel end from device.

III Count not exhausted and channel end from device.

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.

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.

IL Incorrect length is indicated with the interruption condition.

– Incorrect length is not indicated.

Chain command The channel performs command chaining upon receipt of device end.

Chain data The channel immediately fetches a new CCW for the same operation.

* The situation where the 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.

Channel Chaining Action

address referring to a nonexistent or protected area causes an error indication only after the I/O device has attempted to transfer data to or from the invalid location.

Data chaining during an input operation causes the new CCW to be fetched when all data designated by the current CCW has been placed in main storage. On an output operation, the channel may fetch the new CCW from main storage ahead of the time data chaining occurs. Any programming errors in the 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. If the device concludes the operation before all data designated by the current CCW has been transferred, the conditions associated with the prefetched CCW are not indicated to the program.

Only one CCW describing a data area may be prefetched and buffered in the channel. If the prefetched CCW specifies transfer in channel, only one more CCW is fetched before the exhaustion of the current CCW.

Programming Note

Data chaining may be used to rearrange information as it is transferred between main storage and an I/O device. Data chaining permits blocks of information to be transferred to or from noncontiguous areas of storage, and, when used in conjunction with the skipping function, data chaining enables the program to place in main storage selected portions of a block of data.

When, during an input operation, 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 a channel program data-chains to a CCW placed in storage by the CCW specifying data chaining, the input block is said to be self-describing. A self-describing block contains one or more CCWs that specify storage locations and counts for subsequent data in the same input block.

The use of self-describing blocks is equivalent to the use of unchecked data. An I/O data-transfer malfunction that affects validity of a block of information is signaled only at the completion of data transfer. The error condition normally does not prematurely terminate or otherwise affect the 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 thus read is in error, use of the CCW in the current operation may cause subsequent data to be placed in wrong locations in main storage with resultant destruction of its contents, subject to the control of the protection system.

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 the receipt of the device-end signal for the current operation. When command chaining takes place, the completion of the current operation does not cause an I/O interruption, and the count indicating the amount of data transferred during the current operation is not made available to the program. For operations involving data transfer, the new command always applies to the next block of data at the device.

Command chaining takes place and the new operation is initiated only if no unusual conditions have been detected in the current operation. In particular, the channel initiates a new I/O operation by command chaining upon receipt of a status byte containing only the following bit 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 a channel end must have been signaled before device end, with all other status bits off. If a condition such as attention, unit check, unit exception, incorrect length, program check, or protection check has occurred, the sequence of operations is concluded, and the status associated with the current operation causes an interruption condition to be generated. The new CCW in this case is not fetched. The incorrect-length condition does not suppress command chaining if the current CCW has the SLI flag on.

An exception to sequential chaining of CCWs occurs when the I/O device presents the status-modifier condition with the device-end signal. When command chaining is specified and no unusual conditions have been detected, the combination of status-modifier and device-end bits causes the channel to fetch and chain to the CCW whose main-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 of data by means of a single START I/O or START I/O FAST RELEASE. It also permits a subchannel to be set up for 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 main-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 main 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.

No checking for invalid or protected data addresses takes place during skipping.

Programming Note

Skipping, when combined with data chaining, permits the program to place in main storage selected portions of a block of information from an I/O device.

Program-Controlled Interruption

The program-controlled interruption (PCI) function permits the program to cause an I/O interruption during 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 on, the channel attempts to interrupt the program. 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 the channel is not masked. No predictable relation 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 condition is carried over to the new CCW. This carryover occurs both on data and command chaining and, in either case, the condition is propagated through the transfer in channel command. The PCI conditions 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 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 command address is eight higher 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 conditions, such as channel data check, program check, or protection check by the time the interruption occurs, the corresponding channel-status bit is on, although the condition in the subchannel is not reset and is indicated again upon the termination of the operation.

The presence of any unit-status bit 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 added.

However, when the interruption condition due to the PCI flag has been delayed until the operation at the subchannel has been concluded, two interruptions from the subchannel may still take place, with the first interruption indicating and clearing the PCI condition alone, and the second providing the CSW associated with the ending status. Whether one or two interruptions occur depends on the model and on whether the PCI condition has been assigned the highest priority for interruption at the time of concluding. TEST I/O or CLEAR I/O addressed to the device associated with an interruption condition in the subchannel clears the PCI condition as well as the one associated with the concluding.

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 Note

Since no unit-status bits are placed in the CSW associated with the concluding of an operation of the selector channel by HALT I/O or HALT DEVICE, the presence of a unit-status bit with the PCI bit is not a necessary condition for the operation to be concluded. When the selector channel contains the PCI bit 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.

Program-controlled interruption provides a means of alerting the program of the progress of chaining during an I/O operation. It permits programmed dynamic main-storage allocation.

Channel Indirect Data Addressing

Channel indirect data addressing (CIDA), a companion facility to dynamic address translation, provides assistance in translating data addresses for I/O operations. It permits a single channel command word to control the transmission of data that spans noncontiguous pages in real main storage.

Channel indirect data addressing is specified by a flag bit in the CCW which, when one, indicates that the data address in the CCW is not used to directly address data. Instead, the address points to a list of words, called indirect-data-address words (IDAWs), each of which contains an absolute address designating a data area within a 2,048-byte block of main storage.

When the indirect data addressing bit in the CCW is one, bits 8-31 of the CCW specify the location of the first indirect data address word (IDAW) to be used for data transfer for the command. Additional IDAWs, if needed for completing the data transfer for the CCW, are in successive locations in storage. The number of IDAWs required for a CCW is determined by the count field of the CCW and by the data address in the initial IDAW. When, for example, the CCW count field specifies 4,000 bytes and the first IDAW specifies a location in the middle of a 2,048-byte block, three IDAWs are required.

Each IDAW is used for the transfer of up to 2,048 bytes. The IDAW specified by the CCW can designate any location. Data is then transferred, for read, write, control, and sense commands, to or from successively higher storage locations or, for a read backward command, to successively lower storage locations, until a 2,048-byte block boundary is reached. The control of data transfer is then passed to the next IDAW. The second and any subsequent IDAWs must specify, depending on the command, the first or last byte of a 2,048-byte block. Thus, for read, write, control, and sense commands, these IDAWs will have zeros in bit positions 21-31. For a read backward command, these IDAWs will have ones in bit positions 21-31.

Except for the unique restrictions on the specification of the data address by the IDAW, all other rules for the data address, such as for protected storage and invalid addresses, and the rules for data prefetching, remain the same as when indirect data addressing is not used.

A channel may prefetch any of the IDAWs pertaining to the current CCW. An IDAW takes control of the data transfer when the last byte has been transferred for the previous IDAW for that CCW. Errors detected in prefetched IDAWs are not indicated until the IDAW takes control of the data transfer.

The format of the IDAW and the significance of its fields are as follows:

00000000	Data Address
0	8 31

Bit positions 0-7 are reserved for future use and must contain zeros. If any of the bits is detected to be a one, a program-check status condition is generated, and the operation is terminated.

Bits 8-31 specify the location of the first byte to be used in the data transfer. In the first IDAW for a CCW, any location can be specified. For subsequent IDAWs, depending on the command, either the first or the last location of a 2,048-byte block located on a 2,048-byte boundary must be specified. For read, write, control, and sense commands, the beginning of the block must be specified, and bits 21-31 of the IDAW will be zeros. For a read backward command, the end of the block must be specified, and bits 21-31 of the IDAW will be ones. Improper data address specification causes the program-check status conditions to be generated and causes the operation to be terminated.

Commands

The following table 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.

Name	Code	Flags					
Write	MMMM MM01	CD	CC	SLI		PCI	IDA
Read	MMMM MM10	CD	CC	SLI	SKIP	PCI	IDA
Read Backward	MMMM 1100	CD	CC	SLI	SKIP	PCI	IDA
Control	MMMM MM11	CD	CC	SLI		PCI	IDA
Sense	MMMM 0100	CD	CC	SLI	SKIP	PCI	IDA
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

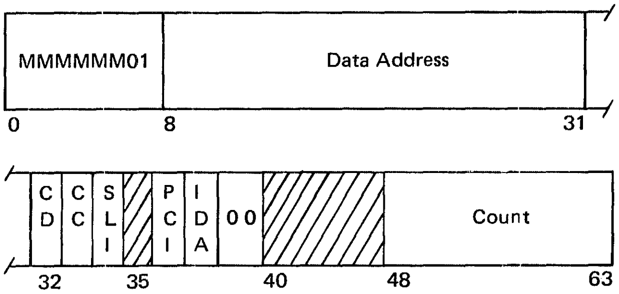
All flags have individual significance, except that the CC and SLI flags are ignored when the CD flag is on. The SLI flag is ignored on immediate operations, in which case the incorrect-length indication is suppressed regardless of the setting of the flag. The PCI flag is ignored during initial program loading.

Each command is described below with an illustration of its CCW format.

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 main storage should not be destroyed until the program has verified that no malfunction affecting the transfer and recording of data was detected.

Write



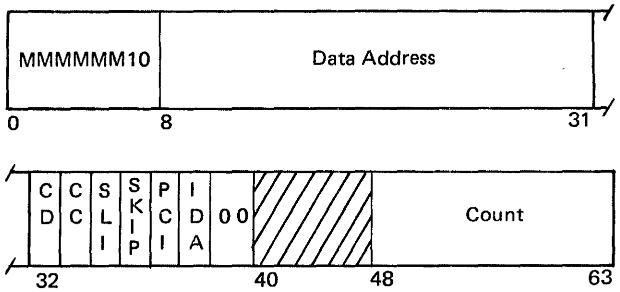
A write operation is initiated at the I/O device, and the subchannel is set up to transfer data from main storage to the I/O device. Data in storage is fetched in an ascending order of addresses, starting with the address specified in the CCW.

A CCW used in a write operation is inspected for the CD, CC, SLI, PCI, and IDA 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.

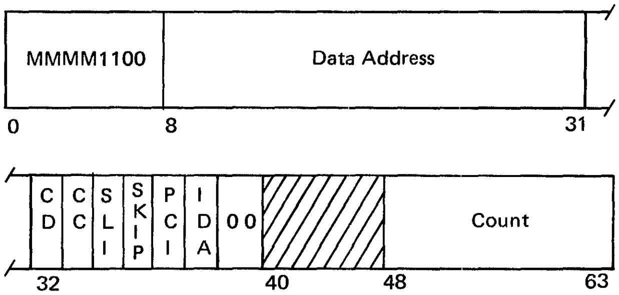
Read



A read operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to main 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 in storage is placed 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 one of the six flags--CD, CC, SLI, SKIP, PCI, and IDA. Bit positions 0-5 of the CCW contain modifier bits.

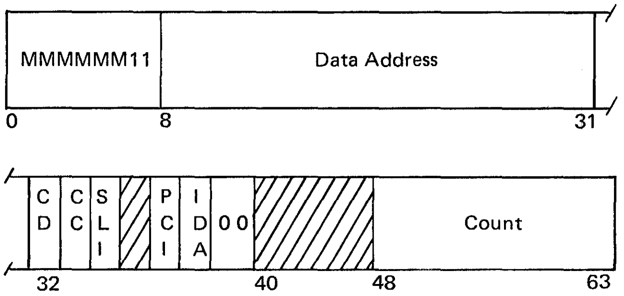
Read Backward



A read-backward operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to main 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 an eight-bit 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 one of the six flags--CD, CC, SLI, SKIP, PCI, and IDA. Bit positions 0-3 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 main 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 over the I/O interface as an immediate operation (see "Immediate Operations"). 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 an order code further specifying the operation to be performed or an address, such as the disk address for the seek function, and is transferred in response to requests by the device.

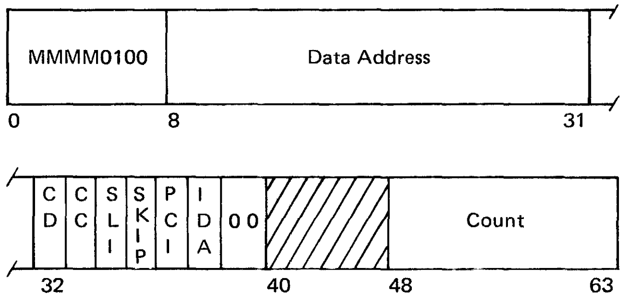
A control command code containing zeros for the six modifier bits is defined as a *no-operation.* The no-operation order causes the addressed device to respond with channel end and device end without causing any action at the device. The order can be executed as an immediate operation, or the device 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 or SRL publication for the device.

A CCW used in a control operation is inspected for the CD, CC, SLI, PCI, and IDA flags. The setting of the skip flag is ignored. Bit positions 0-5 of the CCW contain modifier bits.

Programming Note

Since a CCW 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 main 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 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 conditions detected during the last operation. These bits are common to all devices having this type of information and are designated as follows:

Bit	Designation
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 previously designating the data block.

Intervention Required: The last operation could not be executed because of a condition requiring some type of intervention at the device. This bit indicates conditions such as an empty hopper in a card punch or the printer being out of paper. It is also turned on when the addressed device is in the not-ready state, 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 over the I/O interface. During writing, bus-out check indicates that incorrect data has been recorded at the device, but the condition 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 suppresses checking for command reject and intervention required conditions.

Equipment Check: During the last operation, the device or the control unit has detected equipment malfunctioning, such as an invalid card hole count or 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 conditions 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 main storage. The control unit forces correct parity on data sent to the channel. On writing, this condition 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 main storage may be shifted to fill the gap. On an output operation, overrun indicates that data recorded at the device may be invalid. The overrun bit is also turned on 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 or SRL publication for the device.

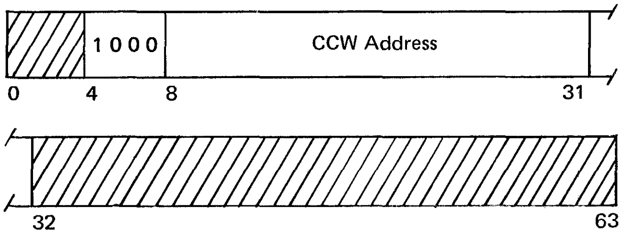
The basic sense command has zero modifier bits. This command initiates a sense operation on all devices and cannot cause the command-reject, intervention-required, data-check, or overrun bits to be turned on. If the control unit detects an equipment malfunction, or invalid parity of the sense command code, the equipment-check or bus-out-check bits are turned on, 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 pertaining to the last I/O operation or unit action may be reset any time after the completion of a sense command addressed to that device. The sense information may also be reset by any other command addressed to the control unit, provided the busy bit is not included in the initial status byte, except where the command is a TEST I/O or a no-operation and is addressed to the device that causes the sense.

A CCW used in a sense operation is inspected for every one of the six flags--CD, CC, SLI, SKIP, PCI, and IDA. Bit positions 0-3 of the CCW contain modifier bits.

Transfer in Channel



The next CCW is fetched from the location in absolute main storage designated by the data-address field of the CCW specifying transfer in channel. The transfer-in-channel command does not initiate any I/O operation at the channel, and the I/O device is not signaled of the execution of the command. The purpose of the transfer-in-channel command is to provide chaining between CCWs not located in adjacent doubleword locations in an ascending order of addresses. The command can occur in both data and command chaining.

The first CCW designated by the CAW may not specify transfer in channel. When this restriction is violated, no I/O operation is initiated, and the program-check condition is generated. The error causes the status portion of the CSW with the program-check indication 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 independently of the device, the error causes an I/O 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 may not be fetched from a location designated by an immediately preceding transfer in channel. When either of these errors is detected or when an invalid address is specified in transfer in channel, the program-check condition is generated. When the transfer-in-channel command designates a CCW in a location protected against fetching, the protection-check condition is generated. Detection of these errors during data chaining causes the operation at the I/O device to be terminated and an interruption condition to be generated, whereas during command chaining it causes only an interruption condition to be 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 I/O interface sequence. When immediate retry can be performed, it presents a channel-end, unit-check, and status-modifier status bit combination, together with device end. When immediate retry cannot be performed, the presentation of device end is delayed until the control unit is prepared. When 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 condition code 1, to be set. The subsequent termination of the command execution causes an interruption condition to be created.

Programming Notes

The following possible results of a command retry must be anticipated by the program:

1. A CCW containing a PCI may, if retried because of command retry, cause multiple PCI interruptions to occur.
2. A channel program consisting of a single, unchained CCW specifying an immediate command may cause a condition code 0 rather than 1 to be set. This setting of the condition code occurs if the control unit signals command retry at the time initial status is presented to the command. The channel program then causes a later interruption 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 prior to the presentation of device end, as when a PCI interruption is taken, contains the address of the command to be retried + 8.
5. If a HALT I/O, HALT DEVICE, or CLEAR I/O instruction is issued between the initiation of a retry but prior to the presentation of de-

vice end, the CSW contains the address of the command to be retried + 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 conditions. These conditions 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. The status conditions, as well as an address and a count indicating the extent of the operation sequence, are presented to the program in the form of a channel status word (CSW).

Types of Conclusion

Normally an I/O operation at the subchannel lasts until the device signals channel end. The channel-end condition 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 both 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 condition 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 conditions that precluded the initiation 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 of the busy or not-operational condition.

When an unusual condition 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 conditions are indicated to the program by means of the corresponding status bits in the CSW. The not-operational condition, 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, is indicated by means of the interface-control-check bit. The new operation at the I/O device is not started.

When START I/O FAST RELEASE is executed by a channel independently of the addressed device, tests on most program-specified information, on control-unit and device availability, on control-unit and device status, and on most error conditions are performed subsequent to the execution of START I/O FAST RELEASE. Some conditions 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

Instead of accepting or rejecting a command, the I/O device can signal the channel-end condition immediately upon receipt of the command code. An I/O operation causing the channel-end condition to be signaled during the initiation sequence is called an "immediate operation."

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. If no command chaining occurs, the channel-end condition 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, and 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 independently 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 conditions 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 the channel-end condition 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. The data address in the CCW is not checked for validity.

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 or SRL 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 said to be working 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 conditions, the channel-end condition 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 the unusual conditions, if any. 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 the channel-end condition. 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 conditions occurs:

The storage areas specified for the operation are exhausted or filled.
Program-check condition is detected.
Protection-check condition is detected.
Chaining-check condition is detected.

The first of these conditions 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 conditions are due to errors and cause premature concluding of data transfer. In either case, the concluding is signaled in response to a service request from the device and causes data transfer to cease. If the device has no blocks defined for the operation (such as writing on magnetic tape), it concludes the operation and generates the channel-end condition.

The device can control the duration of an operation and the timing of channel end by blocking of data. On certain operations for which blocks are defined (such as reading on magnetic tape), the device does not 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.

Checking for the validity of the data address is performed only as data is transferred to or from main storage. When the initial data address in the CCW is invalid, no data is transferred during the operation, and the device is signaled to conclude the operation in response to the first service request. On writing, devices such as magnetic tape units request the first byte of data before any mechanical motion is started and, if the initial data address is invalid, the operation is 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 or SRL publication for the device.

When command chaining takes place, the subchannel appears to be in the working state from the time the first operation is initiated until the device signals the channel-end condition of the last operation of the chain. On the selector channel, the device executing the operation stays connected to the channel and the whole channel appears to be in the working state for the duration of the execution of the chain of operations. On the multiplexer channel an operation in the burst mode causes the channel to appear to be in the Working state only for the duration of the transfer of the 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 on the interface.

Any unusual conditions cause command chaining to be suppressed and an interruption condition to be generated. The unusual conditions 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 condition by the time the channel-end signal for the operation is received, the chain is ended as if the operation during which the condition 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 the burst mode, the channel issues the halt signal to the device operating with the channel, regardless of either the current activity in the channel and on the interface or the address of the device. 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 the 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 command address and count in the associated CSW indicate how much data has been transferred, and the channel-status bits reflect the unusual conditions, 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 generates the channel-end condition. 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 protection key, command address, count, and channel status fields of the associated CSW. The channel-end condition 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 generated. These conditions 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, the channel-end condition 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, and indicates at what point data transfer was terminated. 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 information pertaining to the current operation, if any, with the addressed device to be stored in the CSW and the subchannel to be set to the available state. Unless a channel check had been detected prior to the execution of CLEAR I/O or occurs during the execution of CLEAR I/O, the contents of the CSW indicate the point at which the channel program was terminated. The count field and the incorrect length indication are, however, unpredictable.

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 stored.

The first interruption condition arises on a selector channel when channel-end status is presented to the channel. This occurs only when the interruption-pending status 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 presented 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 will be stored.

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 stored 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 stored in the CSW.

Termination Due to Equipment Malfunction

When channel equipment malfunctioning is detected or invalid signals are received over the I/O interface, 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 the channel-control-check or interface-control-check condition. In channels sharing common equipment with the CPU, malfunctioning detected by the channel may be indicated by a machine-check interruption, in which case no CSW is stored. Equipment malfunctioning may cause the channel to perform the I/O-selective-reset or I/O-system-reset function 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

The conditions causing requests for I/O interruptions to be initiated are called I/O interruption conditions. An I/O interruption condition can be brought to the attention of the program only once and is cleared when it causes an interruption. Alternatively, an I/O 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 the attention, device-end, and control-unit-end conditions, and the channel-end condition when provided by a device after concluding of the operation.

The device attempts to initiate a request to the channel for an interruption whenever it detects any of the following conditions:

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 conditions:

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 condition 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 condition 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 condition 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 conditions. Furthermore, more than one interruption condition associated with the same device can be cleared at the same time. As an example, when the channel-end condition is not cleared at the device by the time device end is generated, both conditions may be indicated in the CSW and cleared at the device concurrently.

However, at the time the channel assigns 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 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 with the condition at the subchannel, even if the status is generated before the CPU accepts the condition.

The method of processing a request for interruption due to equipment malfunctioning depends on the model. In channels sharing common equipment with the CPU, malfunctioning detected by the channel may be indicated by causing a machine-check interruption.

When the channel detects any of the following conditions, it initiates a request for an I/O interruption 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.

The channel available interruption (CAI) condition is provided on all 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 in the I/O old PSW in BC mode and in the I/O communications area in EC mode 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 busy condition thus indicated no longer exists. When the busy condition which caused condition code 2 was due to a subchannel busy with a device other than the one addressed, the concluding of the busy condition is not signaled by a CAI. Since any other interruption condition (except PCI) accomplishes the same function as CAI, a pending CAI condition is reset upon the occurrence of any interruption (except PCI) on that channel. Some channels also reset a pending CAI condition when another interruption condition (except PCI) is cleared by a TEST I/O on the same channel. The occurrence of another channel-busy condition prior to the CAI causes the CAI condition to be suspended until the busy condition is past.

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 of a channel busy condition does not subsequently respond with a condition code 0 to a TEST CHANNEL without clearing an interruption condition in the interim.

Priority of Interruptions

All requests for I/O interruption are 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 requests is controlled by two types of mechanisms--one establishes the priority among interruption conditions associated with devices attached to the same channel, and another establishes priority among requests from different channels. A channel requests an I/O interruption only after it has established priority among requests from its devices. The conditions responsible for the requests are preserved in the devices or channels until accepted by the CPU.

Assignment of priority to 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.

The selector channel assigns the highest priority to conditions associated with the portion of the operation in which the channel is involved. These conditions include channel end, program-controlled-interruption, HALT I/O or HALT DEVICE in the channel, and errors prematurely concluding a chain of operations. The selector channel cannot handle any interruption conditions other than those due to the PCI flag while operation is in progress.

As soon as the selector channel has cleared the interruption conditions associated with data transfer it starts monitoring devices for attention, control-unit-end, and device-end conditions and for the channel-end condition associated with operations concluded by HALT I/O, HALT DEVICE, or CLEAR I/O. The highest priority is assigned to the I/O device that first identifies itself on the interface.

On the byte-multiplexer channel the priority among requests for interruption is based on response from devices. The highest priority is assigned to the device that first identifies itself with an interruption condition or that requests service for data transfer and contains the PCI condition in the subchannel.

The assignment of priority among interruption conditions for a block-multiplexer channel differs among models. Some block-multiplexer channels assign priorities as done by the byte-multiplexer channel. Others assign priorities in a manner which appears random.

Except for conditions associated with concluding of data transfer, the current assignment of priority for interruption among devices on a channel may be canceled when START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, or HALT DEVICE is issued to the channel. Whenever the assignment is canceled, the channel resumes monitoring for interruption conditions and reassigns the priority on completion of the activity associated with the I/O instruction.

The assignment of priority among requests for interruption from channels is based on the type of channel and its address assignment. The priorities of channels 1-15 are in the order of their addresses, with channel 1 having the highest priority. The interruption priority of multiplexer channel 0 is not fixed, and depends on the model and on the current activity in the channel. Its priority may be above, below, or between those of channels 1-15.

Interruption Action

An I/O interruption can occur only when the channel accommodating the device is not masked and after the execution of the current instruction in the CPU has been finished. If a channel has established the priority among requests for interruption from devices while the CPU was disabled for interruptions from the channel, the interruption occurs immediately after the finishing of the instruction removing the mask and before the next instruction is executed. This interruption is associated with the highest priority condition on 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 finishing of the instruction removing the mask. 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 I/O device or, in the case of control-unit end, the control unit causing the interruption is identified in BC mode by the channel address in bit positions 16-23 and by the device address in bit positions 24-31 of the old PSW. In EC mode, the I/O device or control unit is identified in the I/O-address field (locations 186-187) of the I/O communications area (IOCA). The CSW associated with the interruption identifies the condition responsible for the interruption and provides further details about the progress of the operation and the status of the device.

Programming Note

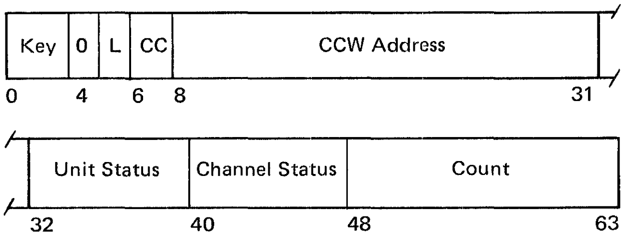
When a number of I/O devices on a shared control unit are concurrently executing operations such as rewinding tape or positioning a disk-access mechanism, the initial device-end signals generated on completion of the operations are provided in the order of generation, unless command chaining is specified 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 either because the channel is masked 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 conditions under 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 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 placed in main storage at real location 64 of the CPU to which the channel is configured, 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.

When the CSW is stored as a result of an I/O interruption, the I/O device is identified in BC mode in the interruption code of the old PSW and in EC mode in the I/O-address field of the I/O communications area (IOCA). 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:

Protection Key: Bits 0-3 form the protection 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 pending logout condition 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 conditions 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		
Bit 6	Bit 7	Meaning
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 absolute address that is eight higher than the address of the last CCW used.

Status: Bits 32-47 identify the conditions in the device and the channel that caused the storing of the CSW. Bits 32-39, the unit status, are obtained over the I/O interface and indicate conditions detected by the device or the control unit. Bits 40-47, the channel status, are provided by the channel and indicate conditions associated with the subchannel. Each of the 16 bits represents one type of condition, 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 Conditions

The following conditions are detected by the I/O device or control unit and are indicated to the channel over the I/O interface. The timing and causes of these conditions for each type of device are specified in the SL or SRL 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 conditions 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 condition and is specified in the SL or SRL publication for the device.

The channel does not modify the status bits received from the I/O device. These bits appear in the CSW as received over the interface.

Attention

Attention is generated when the device detects an asynchronous condition that is significant to the program. The condition is interpreted by the program and is not associated with the initiation, execution, or concluding of an I/O operation.

The device can signal the attention condition to the channel when no operation is in progress at the I/O device, control unit, or subchannel. Attention can be indicated with device end upon completion of an operation, and it can be presented to the channel during the initiation of a new I/O operation. Otherwise, the handling and presentation of the condition 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 accompanying device end 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 the status-modifier condition is signaled in response to TEST I/O and the bit appears in the CSW in the absence of any other status bit, presence of the bit 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 key, command address, and count fields. The 2702 Transmission Control is an example of a type of device that cannot execute TEST I/O.

When the status-modifier bit appears in the CSW together with the busy bit, it indicates that the busy condition 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 tape 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 the status modifier and device end means that the normal sequence of commands must be modified. The handling of this set of bits by the channel depends on the operation. If command chaining is specified in the current CCW and no unusual conditions have been detected, presence of status modifier and device end causes the channel to fetch and chain to the CCW whose main-storage address is 16 higher than that of the current CCW. If the I/O device signals the status modifier condition at a time when no command chaining is specified, or when any unusual conditions have been detected, no action is taken in the channel, and the status modifier bit is placed in the CSW.

Status modifier is presented 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.

The control-unit-end condition 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 conditions 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 "Status Modifier."
2. The control unit detected an unusual condition during the portion of the operation after channel end had been signaled to the channel. The indication of the unusual condition 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 conditions 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.

The control unit end condition can be signaled with channel end, 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 conditions, the interruption may be identified by any address assigned to the control unit. A pending 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 it is executing a previously initiated operation or because it contains a pending interruption condition. The interruption condition for the addressed device, if any, accompanies the busy indication. If the busy condition applies to the control unit, busy is accompanied by status modifier.

The following table lists the conditions for devices connected to only one channel when the busy bit appears in the CSW and when it is accompanied by the status-modifier bit. 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.

Condition	CSW Status Stored By				
	SIO or SIOF≠	TIO	CLRIO+	HIO or HDV	I/O Interrupt #
Subchannel available					
DE or attention in device	B, cl	–, cl	*	*	–, cl
Device working, CU available	B	B	*	*	*
CU end or channel end in CU:					
for the addressed device	B, cl	–, cl	–	*	–, cl
for another device	B, SM	B, SM	–	*	–, cl
CU working	B, SM	B, SM	–	*	B, SM
Interruption pending in subchannel for the addressed device because of:					
chaining terminated by busy condition	*	B, cl	–, cl	*	B, cl
other type of termination	*	–, cl	–, cl	*	–, cl
Subchannel working					
CU available	*	*	–	–	*
CU working	*	*	–	B, SM	*

Explanation:

B Busy bit appears in CSW.

cl Interruption condition cleared; status appears 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 off.

≠ 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 is executed as TEST I/O, 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 a channel-end condition to be generated, and there is only one Channel End for an operation. The channel-end condition is not generated 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. The channel-end condition is not made available to the program when a chain of commands is prematurely concluded because of an unusual condition 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 generated depends on the operation and the type of device. For operations such as writing on magnetic tape, the channel-end condition 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, the channel-end condition occurs when the gap on tape reaches the read-write head. On devices equipped with buffers, such as the IBM 3211 Printer Model 1, the channel-end condition 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 the condition may be delayed until completion of the operation. Operations that do not cause any data to be transferred can provide the channel-end condition during the initiation sequence.

A channel-end condition pending in the control unit causes the control unit to appear busy for 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 or, on some devices, by manually changing the device from the not-ready to the ready state. The condition normally indicates that the I/O device has become available for use for another operation.

Each I/O operation causes a device-end condition, and there is only one device end to an operation. The device-end condition 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 condition is detected during the initiation of a chained command, in which case the chain is concluded without the device-end indication.

The device-end condition associated with an I/O operation is generated either simultaneously with the channel-end condition or later. On 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, such as an IBM 3211 Printer Model 1, the device-end condition 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 in the subchannel, receipt of the device-end signal, in the absence of any unusual conditions, 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 condition that is detailed by the information available to a sense command. Unit check may indicate that a programming or an 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 condition other than the one identified by unit exception has occurred. The unit-check bit provides a summary indication of the conditions identified by sense data.

An error condition 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 condition pertains to the activity initiated by a command and is of immediate significance to the program, the condition 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. The IBM 2150 Console, for example, accepts and executes the alarm control order when the printer is not ready. Selection of a device in the not-ready state does not cause a unit-check indication when the sense command is issued, and whenever 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 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 condition 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 condition 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 a pending interruption condition at the time of selection. Under these circumstances, the device responds by providing the busy bit and indicating the pending interruption condition, if any. The command code invalidity is not indicated.

Concluding of 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 Note

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.

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 condition that usually does not occur. Unit exception includes conditions 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.

The unit-exception condition 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 condition is of immedicate 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 condition 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 condition was detected. Any unusual conditions associated with an operation, but detected after device end has been cleared, are indicated by signaling unit exception with attention.

A command does not cause unit exception when the device responds to the command during the initial selection with busy status.

Concluding an operation with the unit-exception indication causes command chaining to be suppressed.

Channel Status Conditions

The following conditions are detected and indicated by the channel. Except for the conditions caused by equipment malfunctioning, they can occur only while the subchannel is involved with the execution of an I/O operation.

Program-Controlled Interruption

The program-controlled interruption condition is generated when the channel fetches a CCW with the program-controlled interruption (PCI) flag on. The 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 of masking of the channel or other activity in the system.

Detection of the PCI condition does not affect the progress of the I/O operation.

Incorrect Length

Incorrect length occurs when the number of bytes contained in the storage areas assigned for the I/O operation is not equal to the number of bytes requested or offered by the I/O device. Incorrect length is indicated for one of the following reasons:

Long Block on Input: During a read, read-backward, or sense operation, the device attempted to transfer one or more bytes to storage after the assigned storage areas were filled. The extra bytes have not been placed in main 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 main-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.

The incorrect-length indication is suppressed when the current CCW has the SLI flag and does not have the CD flag. The indication does not occur for immediate operations and for operations rejected during the initiation sequence.

Presence of the incorrect-length condition suppresses command chaining unless the SLI flag in the

CCW is on or unless the condition occurs in an immediate operation. See the table in the chaining section of this manual for the effect of the CD, CC, and SLI flags on the indication of incorrect length.

Programming Note

The setting of the incorrect-length indication is unpredictable in the CSW stored during CLEAR I/O.

Program Check

Program check occurs when programming errors are detected by the channel. The condition 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 low-order bits of the CCW address are not zero.

Invalid CCW Address: The channel has attempted to fetch a CCW from a main-storage location which is not available to the channel. An invalid CCW address can occur in the channel because the program has specified an invalid address in the CAW or in the transfer-in-channel command or because on chaining the channel attempts to fetch a CCW from an unavailable location.

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.

Invalid IDAW Address Specification: Channel indirect data addressing is specified, and the data address does not designate the first IDAW on an integral word boundary.

Invalid IDAW Address: The channel has attempted to fetch an IDAW from a main-storage location which is not available to the channel. An invalid IDAW address can occur in the channel because the program has specified an invalid address in a CCW that specifies indirect data addressing or because the channel, on sequentially fetching IDAWs, attempts to fetch from an unavailable location.

Invalid Data Address: The channel has attempted to transfer data to or from a main-storage location which is not available to the channel. An invalid data address can occur in the channel because the program has specified an invalid address in the CCW, or in an IDAW, or because the channel, on sequentially accessing storage, attempts to access an unavailable location.

Invalid IDAW Specification: Bits 0-7 of the IDAW are not all zeros, or the second or subsequent IDAW does not specify the beginning or, for read-backward operations, the ending byte of a 2,048-byte block.

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 38-39.

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 the program-check condition during the initiation of an operation causes execution of the operation to be suppressed. When the condition 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. The program-check condition causes command chaining to be suppressed.

Protection Check

Protection check occurs when the channel attempts a storage access that is prohibited by the protection mechanism. Protection applies to the fetching of CCWs, IDAWs, and output data, and to the storing of input data. Storage accesses associated with each I/O operation are performed using the key provided in the CAW associated with that operation.

When the protection-check condition occurs during the fetching of a CCW that specifies the initiation of an I/O operation, or occurs during the fetching of the first IDAW, 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. The condition 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 main 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 anywhere inboard of the I/O interface: in the channel, in main storage, or on the path between the two. Channel data check may be indicated for data with an invalid checking block code (CBC) in main storage when that data is referred to by the channel but 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 over the I/O interface, and all data placed in main storage has valid CBC. When, on an input operation, the channel attempts to store less than a complete checking block, and invalid CBC is detected on the checking block in storage, the contents of the location remain unchanged, with invalid CBC. On an output operation, whenever a channel data check is indicated, all bytes that came from a checking block with invalid CBC have been transmitted on the interface with parity errors.

A condition indicated as 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 I/O 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 any machine malfunctioning affecting channel controls. The condition includes invalid CBC on CCW and data addresses and invalid CBC on the contents of the CCW. The condition also includes 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. Conditions responsible for channel control check may cause the contents of the CSW to to invalid and conflicting. The CSW as generated by the channel has valid CBC.

Detection of the channel-control-check condition causes the current operation, if any, to be immediately concluded.

In some situations, the channel-control-check condition may be reported as an external-damage or system-damage machine-check condition. Channel control check is set whenever CSW bit 5, logout pending, is set.

Interface Control Check

Interface control check indicates that an invalid signal has occurred on the I/O interface. The condition is detected by the channel and ususally indicates malfunctioning of an I/O device. It can be due to the following reasons:

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 by raising the disconnect-in-line on the I/O interface.

Detection of the interface control check condition 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. The condition occurs when the I/O data rate is too high for the particular resolution of data addresses. Chaining check cannot occur on output operations.

Detection of the chaining-check condition causes the I/O device to be signaled to conclude the operation. It causes command chaining to be suppressed.

Contents of Channel Status Word

The contents of the CSW depend on the condition causing the storing of the CSW and on the programming method by which the information is obtained. The status portion always identifies the condition that caused storing of the CSW. The protection key, command 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

Conditions associated with the execution or concluding 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 conditions associated with command chaining and equipment malfunctioning, the storing can be caused by the PCI or channel-end condition 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, 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 thc 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 condition causes command chaining to be suppressed, the premature concluding of the chain is not explicitly indicated in the CSW. A CSW associated with a concluding due to a condition occurring at channel-end time contains the channel-end bit and identifies the unusual condition. When the device signals the unusual condition with control unit end or device end, the channel-end indication is not made available to the program, and the channel provides the current protection key, command address, and count, as well as the unusual indication, with the control-unit-end or device-end bit in the CSW. The command address and count fields pertain to the operation that was executed.

When the execution of a chain of commands is concluded by an unusual condition detected during initiation of a new operation, the command address and count fields pertain to the rejected command. Except for conditions caused by equipment malfunctioning, concluding at the 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 turned off.

A CSW associated with conditions occurring after the operation at the subchannel has been concluded contains zeros in the protection key, command address, and count fields, provided the conditions are not cleared during START I/O or START I/O FAST RELEASE and provided the logout-pending condition is not indicated. These conditions include attention, control unit end, and device end (and channel end when it occurs after the concluding of an operation on the selector channel by HALT I/O or HALT DEVICE).

When the above conditions, 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 protection key, command address, 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 the channel-control-check or interface-control-check conditions are indicated. Channel control check indicates that equipment errors have been detected which can cause any part of the CSW, as well as the address in the PSW identifying the I/O device, 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.

When any I/O instruction cannot be executed because of a pending logout condition 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.

Protection Key

A CSW stored to reflect the progress of an operation at the subchannel contains the protection key used in that operation. The contents of this field are not affected by programming errors detected by the channel or by the condition causing termination of the operation.

Command Address

When the CSW is formed to reflect the progress of the I/O operation at the subchannel, the command address is normally eight higher than the address of the last CCW used in the operation.

The following table lists the contents of the command address field for all conditions that can cause the CSW to be stored. The conditions are listed in order of priority; that is, if two conditions are indicated or occur, the CSW appears as indicated for the condition higher on the list. The programming errors listed in the table refer to conditions included in program check. When a CSW has been stored and the situation exists that a command retry request has been recognized but the CCW has not been reexecuted, "last-used CCW + 8" is the CCW that is to be retried.

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

Condition	Contents of Field
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
Invalid CCW address specified in transfer in channel (TIC)	Address of TIC + 8
Invalid CCW address in TIC	Address of TIC + 8
Invalid CCW address generated	First invalid CCW address + 8
Invalid command code	Address of invalid CCW + 8
Invalid count	Address of invalid CCW + 8
Invalid data address	Address of invalid CCW + 8
Invalid CCW format	Address of invalid CCW + 8
Invalid sequence – 2 TICs	Address of second TIC + 8
Protection check	Address of protected CCW + 8
Chaining check	Address of last-used CCW + 8
Termination under count control	Address of last-used CCW + 8
Termination by I/O device	Address of last-used CCW + 8
Termination by HALT I/O	Address of last-used CCW + 8
Termination by CLEAR I/O	Address of last-used CCW + 8
Suppression of command chaining due to unit check or unit exception with device end or control unit end	Address of last CCW used in the completed operation + 8
Termination on command chaining by busy, unit check, or unit exception	Address of CCW specifying the new operation + 8
Deferred condition code 1 for START I/O FAST RELEASE	Address of CCW specifying the new operation + 8
PCI flag in CCW	Address of last-used CCW + 8
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 CSW Command Address Field

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 main storage; on an output operation, the difference is equal to the number of bytes transferred to the I/O device.

The following table lists the contents of the count field for all conditions that can cause the CSW to be stored. The conditions are listed in the order of priority; that is, if two conditions are indicated or occur, the CSW appears as for the condition higher on the list.

Status

The status bits identify the conditions 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 error conditions, all conditions may be indicated or only one may appear in the CSW, depending on the condition and model. Conditions 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 condition. When an operation is concluded by program check, protection check, or chaining check, the channel identifies the condition responsible for the concluding 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 contains the PCI flag but a programming error in the contents of the CCW precludes the initiation of the operation, whether the PCI bit appears in the CSW associated with the interruption condition is unpredictable. Similarly, if a programming error in the

Condition	Contents of Field
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 CSW Count Field

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 contain the PCI bit. Furthermore, when the channel detects a programming error in the CAW or in the first CCW, the PCI bit may unpredictably appear in a CSW stored by START I/O or START I/O FAST RELEASE without the PCI flag being on in the first CCW associated with the instruction.

However, if the CCW fetched on command chaining contains the PCI flag but an unusual condition signaled by the device precludes the initiation of the operation, the PCI bit appears 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.

Conditions detected by the channel are not related to those identified by the I/O device.

The following table summarizes the handling of status bits. The table 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.

Channel Logout

When a channel stores a CSW that indicates the channel-control-check condition in the absence of the logout-pending indication, or the interface-control-check condition, or, on some channels, the channel-data-check condition, a channel logout, useful for error-recovery or diagnostic purposes, accompanies the storing of the CSW. The logout may be a limited channel logout, a full channel logout, or both. The type of logout that occurs and, for the full logout, the length of the logout and the location at which it is stored, depend on the channel type and model number.

The limited channel logout contains model-independent information and is stored at real locations 176-179 of the CPU to which the channel is configured. When it is stored, bit 0 of the logout is always stored as a zero.

The full channel logout contains model-dependent information. When the length of the full channel logout exceeds 96 bytes, it is stored at the location specified by the I/O extended logout (IOEL) pointer in real locations 173-175 of the CPU to which the channel is configured. When the length of the full channel logout is 96 bytes or fewer, the channel may either use the IOEL pointer or may store the logout in the fixed-logout area, real locations 256-351 of

			Upon Termination of Operation at								
Status	When I/O Is Idle	When Subchannel is Working	Sub-channel	Control Unit	I/O Device	During Command Chaining	By SIO or SIOF	By TIO	By CLRIO+	By HIO or HDV	By I/O Inter-ruption
Attention	C*				C*	C*	S	S	S		S
Status modifier					C	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 the status condition at the indicated time. A CSW or its status portion is not necessarily stored at this time.

Conditions such as channel end or device end are created at the indicated time. Other conditions may have been created previously, but are made accessible to the program only at the indicated time. Examples of such conditions 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 equipment malfunctioning have 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 condition has been created previously. The letter C appearing with the S indicates that the status condition 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, equipment malfunctioning may be detected during an I/O interruption, causing channel or interface control check to be indicated; or a device such as the 2702 may signal the control-unit-busy condition 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 condition generates an interruption condition.

Channel end and device end do not result in interruption conditions when command chaining is specified and no unusual conditions 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 concluding 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 is executed as TEST I/O, the entries in the TIO column apply.

Contents of the CSW Status Fields

the CPU to which the channel is configured. The information stored by the STORE CHANNEL ID instruction implies whether the IOEL is used and, if it is used, specifies the maximum full-channel-logout length.

I/O Communications Area

Real locations 160-191 of the CPU to which the channel is configured comprise a permanently assigned area of main storage used for I/O, designated the I/O communications area (IOCA).

160			
164			
168	Channel ID		
172		IOEL Pointer	
176	Limited Channel Logout (LCL)		
180			
184		00000000	I/O Address
188			

I/O Communications Area

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. (See STORE CHANNEL ID.)

I/O Extended Logout Pointer (Locations 173-175): The I/O Extended Logout (IOEL) pointer (locations 173-175) is program-set to designate an area to be used by channels not capable of storing or not choosing to store the channel logout information in the fixed logout area (locations 256-351). The low-order three bits of the pointer are reserved and are ignored by the channel so that the I/O extended logout always begins on a doubleword boundary. Channel logout information may be stored in the IOEL area only when the IOEL mask bit (control register 14, bit 2) of the CPU to which the channel is configured is one.

Whether the IOEL facility is used depends on the channel type and model number. Channels with a logout length not exceeding 96 bytes use either the IOEL area or locations 256-351 as the logout area. Channels with a logout length exceeding 96 bytes use the IOEL area.

Programming Note

The extent of the channel-extended-logout area differs among channels and, for any particular channel, may depend on the features or engineering changes installed. In order to provide for such variations, the program should determine the extent of the logout by means of STORE CHANNEL ID whenever a storage area for the extended logout is to be assigned.

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 limited-channel-logout facility may not be available on all channels. The field, if stored, may be stored only when the CSW or a portion of the CSW is stored and may or may not be accompanied by the full channel logout. Channels which do not store the limited-channel-logout field instead usually store equivalent information in the full channel logout.

The bits of the field are defined as follows:

0 This bit will always be 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.

1-3 *Identity of the storage control unit (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 meaningless. 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 une 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 inter-unit 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 in this field. When a reasonable determination cannot be made, all bits in this field are set to zero.

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 *Reserved.* Stored zero.

14 *Reserved.* Stored zero.

15 *Reserved.* Stored zero.

16-23 *Field validity flags.* These bits indicate the validity of the information stored in the designated fields. When the designated field is stored by the channel with the correct contents, the validity bit is one. When the designated field is stored by the channel with unpredictable contents, the validity bit is zero. The validity bits for nonstored fields are meaningless.

The fields designated are:

Bit 16 -- Interface address
Bit 17 -- (Reserved. Stored zero)
Bit 18 -- (Reserved. Stored zero)
Bit 19 -- Sequence code
Bit 20 -- Unit status
Bit 21 -- Command address and key
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 System reset

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 on the I/O interface 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 channel program address, if validly stored and if nonzero, is the address of the current CCW + 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 over the interface. 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 conditions occurs:

1. When the command 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/O 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 2 or 3 applies. It may also be used at other times when a channel cannot distinguish between code 2 or 3.

110 *Reserved.*

111 *Reserved.*

I/O Address (Locations 185-187): A three-byte field is provided for storing the I/O address on each I/O interruption in EC mode. Zeros are stored at location 185, and the channel and device addresses are stored at locations 186 and 187, respectively.

Locations 160-167, 180-184, and 188-191 are reserved for future I/O use.

System Console

Contents

Operator Section 243
- Address-Compare Controls 244
- Check-Stop Indicator 245
- Configuration Controls 245
- Display-and-Enter Controls. 245
- Emergency-Pull Switch 245
- Enable-System-Clear Key 245
- IMPL Controls 245
- Interrupt Key 245
- Load Indicator 245
- Load Key 245
- Load-Unit-Address Controls 246
- Manual Indicator 246
- Power-Off Key 246
- Power-On Key 246
- Rate Control 246
- Restart Key 246
- Start Key 247
- Stop Key 247
- Store-Status Key 247
- System Indicator 247
- System-Reset Key 247
- Test Indicator 247
- Thermal/CB Power-Check Indicator 247
- TOD Clock Key 247
- Wait Indicator 248

Remote Operator-Control Panel 248

Customer-Engineer Section 248

The system console provides the functions necessary to operate and control the system. It consists of the system control panel and, in most cases, an associated console device, which may be used either as an I/O device or as a manual display-and-enter device. The system console may be implemented in various technologies and configurations; however, certain functions are basic to all models.

The system-control-panel part of the system console contains an operator section and a customer-engineer section.

The operator section provides a means for the control and indication of power, the indication of system status, operator-to-system communication, the control of time-of-day clock security, initial program loading, resets, and other controls required by the operator for intervening in normal programmed operation. It may include controls used for loading microprograms into reloadable control storage and for loading resident diagnostic programs.

The customer-engineer section includes controls intended only for customer-engineer use. Additional customer-engineer controls may be available on the main-storage and channel frames.

The console device, when used as an I/O device, provides a means of communicating control information and of entering and displaying text. Implementation of this device is model-dependent; the device typically may be a keyboard and an associated printer or cathode-ray tube (CRT).

The specific console device provided on a particular model of System/370 is described in the Systems Reference Library (SRL) and System Library (SL) publications for that model.

Operator Section

The operator section contains the basic controls required by the operator and the controls required by the operator for intervening in normal programmed operation. These controls may be interspersed with customer-engineer controls and may, depending on the model, contain switch positions and nomenclature additional to those described here.

Machine errors detected during manual operations do not cause an immediate interruption or logout but may, in some cases, create a pending machine-check-interruption condition. This interruption request, unless it is removed by the use of a reset function, is honored when the CPU is again in the operating state and enabled for the interruption. A machine error that occurs in a manual operation from which recovery cannot be made terminates the operation, and the CPU is placed in the check-stop state.

The following table lists operator controls that are provided on all models. Some models provide additional controls.

Address-Compare Controls

These controls provide a means of setting a main-storage address and stopping the CPU when the address set up on the controls matches the address used in a main-storage reference.

One of the address-compare controls provides two or more settings to specify the action to be taken, if any, when the address match occurs. The two settings are labeled "normal" and "stop." Placing this control in other than the normal setting causes the test indicator to go on.

The "normal" setting disables the address-compare operation.

The "stop" setting causes the CPU to enter the stopped state on a match. The point at which the CPU enters the stopped state depends on the model and the type of reference. Pending I/O, external, and machine-check interruptions may or may not be taken before the stopped state is entered.

In addition to the control for selecting the action to be taken, another control may be provided for specifying the type of main-storage reference which is to be compared. A model may provide one or more of the following settings, in addition to others:

The "any" setting causes the address comparison to be performed on all main storage references.

The "data-store" setting causes address comparison to be performed when main storage is addressed to store data.

The "I/O" setting causes address comparison to be performed when main storage is addressed by a channel to transfer data or fetch a channel command word. Whether references to a channel address word

Name	Typical Implementation
Address-Compare Controls	Rotary switches and toggle switch[2]
Check-Stop Indicator	One or more lights[1]
Configuration Controls	Rotary switches, toggle switches, and pushbuttons
Display-and-Enter Controls	Keyboard-printer[2, 3]
Emergency-Pull Switch	Pull switch
Enable-System-Clear Key	Pushbutton[2]
IMPL Controls	Pushbutton[2, 3]
Interrupt Key*	Pushbutton[2]
Load Indicator*	Light[1]
Load Key*	Pushbutton[2]
Load-Unit-Address Controls*	Rotary switches[2]
Manual Indicator*	Light[1]
Power-Off Key*	Pushbutton
Power-On Key*	Pushbutton, backlighted
Rate Control	Rotary switch[2]
Restart Key	Pushbutton[2]
Start Key	Pushbutton[2]
Stop Key	Pushbutton[2]
Store-Status Key	Special keyboard mnemonic[2, 3]
System Indicator*	Light[1]
System-Reset Key	Pushbutton[2]
Test Indicator*	Light[1]
Thermal/CB Power Check Indicator	One or more lights
TOD Clock Key*	Spring-return toggle switch
Wait Indicator*	Light[1]

Explanation:

* Remote operator control panel, if available, also contains this function.

[1] Or the indicator may be on a CRT display.

[2] Or the function may be provided by an equivalent keyboard input or a CRT-menu selection.

[3] Or the function may be provided by pushbuttons, rotary switches, or both.

or a channel status word cause a match to be indicated depends on the model.

The "IC" setting causes address comparison to be performed when main storage is addressed to fetch an instruction. The low-order 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 is not indicated for the instruction designated by EXECUTE.

Depending on the model and the type of reference, address comparison may be performed on logical, real, or absolute addresses, or controls may be provided to specify the type of address.

The address-compare-control settings can be changed without disrupting CPU operations other than causing the address-comparison stop.

Check-Stop Indicator

The check-stop indicator is on when the CPU is in the check-stop state. The check-stop indicator may be a separate indicator, or some models may use a combination of other indicators to signal the check-stop state. Performance of CPU reset turns off the indicator. The manual indicator may also be on in the check-stop state.

Configuration Controls

Configuration controls provide a means for setting and controlling the configuration of the system. Some of these controls are associated specifically with multiprocessing. The detailed function provided depends on the model.

Display-and-Enter Controls

The system console provides controls and procedures to permit the operator to display and enter information in main storage, the general, floating-point, and control registers, the PSW, and the keys in storage. The CPU must first be placed in the stopped state.

The display-and-enter functions are provided on some models by means of controls on the operator section of the system control panel; on other models, they are provided by the use of the console device. The CRT-menu-selection method may be provided to facilitate the selection of storage facilities and addresses.

Controls may be provided for data formatting, checking, and error indication by means of interactive procedures used on a keyboard-printer or a keyboard-CRT device. In some models, enter-and-display operations cause the manual indicator to be turned off and may cause the start and restart keys to be inoperative.

Main storage addresses for display-and-enter operations are real addresses when dynamic address translation is specified. Some models also include the capability of specifying a logical or absolute address.

Emergency-Pull Switch

Activating the emergency-pull switch turns off all power beyond the power-entry terminal on every unit that is part of the system or that can be switched onto the system.

The switch latches in the out position and can be restored to its in position only by maintenance personnel.

When the emergency-pull switch is in the out position, the power-on key is ineffective.

Enable-System-Clear Key

Activating the enable-system-clear key, in conjunction with the load or system-reset key, results in the performance of the system-clear-reset functions described in detail in "Resets" in the chapter "System Control." In some models, the combination of the enable-system-clear key with the load key or the system-reset key is provided as an integrated selectable function.

IMPL Controls

Controls are provided in some models for initial microprogram loading (IMPL). These controls are model-dependent.

Interrupt Key

Activating the interrupt key causes an external-interruption condition to be generated.

The interruption is taken when the CPU is enabled for the interruption and is in the operating state. Otherwise, the interruption request remains pending.

The interrupt key is effective while power is on the system.

Load Indicator

The load indicator is on during initial program loading; it goes on when the load key is activated and goes out after the loading of the new PSW is completed successfully.

Load Key

Activating the load key causes a reset function to be performed and initial program loading to be started. Whether the enable-system-clear control is activated at the same time determines the type of reset function performed and the resultant effect on the system. In a multiprocessing system, the effect of the key is propagated to all CPUs configured, for reset purposes, to this CPU. See the detailed discussion under "Initial Program Loading" and "Resets" in the chapter "System Control."

Activating the load key may change the configuration, including the connection with channels, storage units, and other CPUs.

The load key is effective while power is on the system.

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.

Manual Indicator

The manual indicator is on when the CPU is in the stopped state. Some functions and several manual controls are effective only when the CPU is in the stopped state.

Power-Off Key

Activating the power-off key initiates a power-off sequence when the power-on key is lighted white or red, that is, when power is on the system.

The contents of nonvolatile main storage (but not the keys in storage associated with the protection facility) are preserved, provided the CPU is in the stopped state when power is turned off.

Power-On Key

Activating the power-on key does the following:

1. Initiates the power-on sequence for the CPU and, depending on the model and configuration, for the main storage, channels, and other components of the system. The sequence is performed in such a manner that no instructions or I/O operations are executed until explicitly specified.
2. Initiates the initial-microprogram-loading (IMPL) sequence for models with volatile control storage.
3. Initiates a power-on reset. See "Resets" in the chapter "System Control" for the detailed description.

The power-on key is effective only when the emergency-pull switch is in the in position and the power-on key is not illuminated white.

Associated with the power-on sequence is a series of power-on key illuminations. Activating the key turns the lens red, indicating initiation of the power-on sequence. The lens remains red until the sequence is completed without power-check conditions, whereupon the lens turns white.

Should any condition prevent completion of the sequence, the lens remains red if partial power is present. If the power is off (excluding control voltages) and the system is not performing a power-on sequence, the lens has no illumination.

Should the CPU complex lose power or indicate a thermal condition after the lens is white, the lens turns red if partial power remains. After a power-check condition is cleared, activating the power-on key results in power being brought up on the system.

Operation Note

Should the power-on IMPL sequence fail to be completed successfully, the manual indicator remains off. The IMPL sequence should then be reinitiated by using the IMPL controls.

Rate Control

The setting of the rate control determines the manner in which instructions are executed.

The rate control has two or more settings, depending on the model. The normal setting is labeled "process." When the rate control is in this setting, the system starts operating at normal speed when the start key is activated. The second setting is labeled "instruction step." When the start key is activated with the rate control in this setting, one instruction or, for interruptible instructions, one unit of operation is executed, and all pending, allowed interruptions are subsequently taken. The CPU then returns to the stopped state.

Any instruction can be executed with the rate control in the instruction-step setting. The performance of input/output operations is not affected. When the CPU is in the wait state, no instruction is executed, but pending, allowed interruptions, if any, are taken before the CPU returns to the stopped state. When the rate control is in the instruction-step setting, initial program loading is completed with the loading of the new PSW. The interval timer is not updated while the rate control is in the instruction-step setting.

The test indicator is on when the rate control is not in the process setting.

If the setting of the rate control is changed while the CPU is in the operating state, the results are unpredictable.

Restart Key

Activating the restart key initiates the restart interruption. See "Restart" in the chapter " Interruptions."

The restart key is effective in both the operating and stopped states. The key is not effective when the CPU is in the check-stop state.

Start Key

Activating the start key causes the CPU to enter the operating state. See "Stopped and Operating States" in the chapter "System Control." The key is effective only when the CPU is in the stopped state, and the effect is unpredictable when the stopped state has been entered by reset.

Stop Key

Activating the stop key causes the CPU to perform the stop function. See "Stopped and Operating States" in the chapter "System Control."

The stop key is effective only while the CPU is in the operating state.

Operation Note

Activating the stop key has no effect when a continuous string of interruptions occurs, when the prefix register contains an invalid address, or when the CPU is unable to complete an instruction because of a machine malfunction.

Store-Status Key

Activating the store-status key initiates the store-status function. Although the store-status function is performed compatibly on all models, the initiation of the function differs among models. Some models initiate the function by the use of a special keyboard mnemonic and some by the use of a pushbutton, while others may provide a CRT-menu selection.

See "Store Status" in the chapter "System Control" for a description of the function.

The manual control for performing store status is effective only when the CPU is in the stopped state.

Operation Note

The store-status function would ordinarily be used in conjunction with a standalone dump program for the analysis of program malfunctions. For such an operation, the following sequence would be called for:

- Activating of the system-reset key.
- Initiation of the store-status function.
- Normal IPL of a standalone dump program.

System Indicator

The system indicator is on when the CPU cluster meter or customer-engineer meter is running.

System-Reset Key

Activating the system-reset key causes a reset function to be performed. Whether the enable-system-clear key is activated at the same time determines the type of reset and the resultant effect on the system. In a multiprocessing system, the effect of the key is propagated to all CPUs configured, for reset purposes, to this CPU. See the detailed discussion under "Resets" in the chapter "System Control."

Activating the system-reset key may change the configuration, including the connection with channels, storage units, and other CPUs.

The key is effective while power is on the system.

Test Indicator

The test indicator is on when a manual control is not in its normal position or when a maintenance function is being performed for the CPU, channels, or storage.

The test indicator is on whenever a control on the system control panel or on any separate maintenance panel for the CPU, storage, or channels is in an abnormal position that can affect the normal operation of a program.

The test indicator may be on when one or more diagnostic functions under control of DIAGNOSE are activated or when certain abnormal circuit-breaker or thermal conditions occur.

The test indicator does not reflect the state of marginal voltage controls.

Thermal/CB Power-Check Indicator

The thermal/CB power-check indicator, which is one or more red lights, goes on when a thermal condition, a circuit-breaker-trip condition, or both conditions are detected in the CPU complex. This indicator is turned off from the customer-engineer power-control panel.

TOD Clock Key

The TOD (time-of-day) clock key provides an interlock in the execution of the instruction SET CLOCK for the purpose of guarding against an unauthorized or inadvertent change to the time-of-day clock value.

When the TOD clock key is not activated, that is, is in the position labeled "secure," the value of the TOD clock is protected against alteration by the program, and the execution of SET CLOCK does not change the value of the clock.

When the TOD clock key is activated, that is, is in the position labeled "enable set," alteration of the clock value by means of SET CLOCK is permitted.

The key is situated between the power-on key and the power-off key. It is spring-loaded, with the spring restoring the key to the secure position.

In a multiprocessing system, the TOD clock is secure only when the TOD clock on this CPU and the keys on all CPUs configured to this CPU are in the secure position. The TOD clock in each CPU of a multiprocessing configuration is enabled for setting when the TOD clock key on that CPU, or on any CPU configured to it, is in the enable-set position.

Wait Indicator

The wait indicator is on when the CPU is in the wait state.

Operation Note

The wait indicator, manual indicator, and system indicator may be used by the operator to determine the status of the system. The following table shows the possible conditions when power is on and the CPU is not in the load or check-stop states.

Remote Operator-Control Panel

In some models, a remote operator-control panel (ROCP) is provided. The remote operator-control panel functions are effective concurrently with those on the system console. The settings of the load-unit-address controls on the remote panel are selected if the load key on the remote panel is activated.

Customer-Engineer Section

The customer-engineer section of the system control panel contains controls intended only for customer-engineer use.

Operation Note

Improper use of the customer-engineer-section controls 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 system operation, including instruction execution and channel operation, to the extent that the operation does not comply with that specified in this manual. While the abnormal setting of such controls causes the test indicator to be turned on in this CPU, in a multiprocessing system the operation of another CPU may be affected even though its test indicator is not turned on.

Manual Indicator[1]	System Indicator[2]	Wait Indicator	CPU State	State of I/O System[3]
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.

[1] The manual indicator may be off in some models while display-and-enter operations are being performed.

[2] When the system indicator is turned on, it remains on for a minimum of approximately one second.

[3] The operation of the console device is included here as an I/O operation. In a multiprocessing system, the system indicator may be on because of activity in another CPU. When this is the case, the state of the I/O system in undetermined.

System Status Indications

Appendix A. System/370 Features

Central Processing Unit

Every CPU incorporates the commercial instruction set, which includes the standard instruction set and the decimal instructions (listed in Appendix C), and the associated basic computing functions, including:

- Byte-oriented operands
- General registers
- Control registers, with bit positions for the block-multiplexing control bit (if block multiplexing is provided), for the interrupt-key and interval-timer masks, for channel masks associated with installed channels, for monitor masks, for control of installed machine-check-handling facilities, and for the IOEL control (if an installed channel has the I/O-extended-logout facility).
- Storage protection
- Interval timer
- Time-of-day clock
- Basic system console functions

Additionally, the following features are available:

Floating-Point Feature

Includes the floating-point instructions (listed in Appendix C) and the floating-point registers.

Universal Instruction Set

Includes the instructions of the commercial instruction set and the floating-point feature.

Extended-Precision Floating-Point Feature

Includes the extended-precision floating-point instructions (listed in Appendix C).

External-Signal Feature

Includes the extension to external interruptions for external signals, the control-register position for the external-signal mask, and the means to accept external signals.

Direct-Control Feature

Includes the external-signal feature and the instructions READ DIRECT and WRITE DIRECT.

CPU-Timer and Clock-Comparator Feature

Includes the clock comparator, the CPU timer, the associated extensions to external interruption, control-register positions for the clock-comparator and CPU-timer masks, and these instructions: SET CLOCK COMPARATOR, STORE CLOCK COMPARATOR, SET CPU TIMER, and STORE CPU TIMER.

Translation Feature

Includes the following facilities:

- *Dynamic Address Translation (DAT).* The DAT facility includes the translation mechanism, with the associated control-register positions and program-interruption codes, and reference and change recording.
- *Program-Event Recording (PER).* The PER facility includes the associated control-register positions and extensions to the program-interruption code.
- *Extended-Control (EC) Mode.*
- *SSM Suppression.* This facility includes the control-register position for the SSM-suppression-control bit and the program-interruption code for special operation.
- *Store Status and Program Reset.*

As part of these facilities, the following instructions are provided: LOAD REAL ADDRESS, PURGE TLB, RESET REFERENCE BIT, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK.

Multiprocessing Feature

Includes the following facilities, which permit the formation of a two-CPU multiprocessing system:

- *Shared Main Storage.*
- *Prefixing.*
- *CPU Signaling and Response.*
- *TOD Clock Synchronization.*

These facilities include four extensions to external interruption (external call, emergency signal, TOD clock sync check, and malfunction alert), control-register positions for the TOD-clock-sync control bit and for the masks for the four external-interruption conditions, and the instructions SET PREFIX, SIGNAL PROCESSOR, STORE CPU ADDRESS, and STORE PREFIX.

Conditional-Swapping Feature

Includes the instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP.

PSW-Key-Handling Feature

Includes the instructions SET PSW KEY FROM ADDRESS and INSERT PSW KEY.

I/O Channels

Every system includes at least one byte-multiplexer, block-multiplexer, or selector channel, with these possible additional features:

Command Retry

Fast Release

Limited Channel Logout (LCL)

I/O Extended Logout (IOEL)

Channel Indirect Data Addressing (CIDA)

Clear I/O

Availability of CPU Features on System/370 Models

	System/370 Model							
CPU Feature	115	125	135	145	155	158	165	168
Commercial instruction set (includes standard instruction set and decimal feature)	Std	Std	Std	Std	Std	Std	Std[1]	Std
Floating-point feature	Opt	Opt	Opt	Opt	Std	Std	Std	Std
Extended-precision floating-point feature	Opt		Opt		Opt	Opt	Std	Std
Direct-control feature	Opt[2]	Opt[2]	Opt	Opt	Opt	Opt	Std	Std
Translation feature	Std	Std	Std	Std	Opt[3]	Std	Opt[3]	Std
CPU-timer and clock-comparator feature	Std	Std	Opt	Opt		Std		Std
Conditional-swapping feature	Std	Std	Opt	Opt		Std		Std
PSW-key-handling feature	—	—	—	Opt		Std		Std
Multiprocessing feature	—	—	—	—	—	Opt	—	Opt

Explanation:

Std = Standard
Opt = Optional
— = Not available

Boxed features are offered as a single package.

[1] Includes the MONITOR CALL instruction and associated control-register positions only when the translation feature is installed.

[2] Only the external-signal feature portion is offered.

[3] These combined four features are available for field installation only on purchased models.

Appendix B. Functions That Differ from System/360

This appendix summarizes the areas where System/370 differs from System/360. Not included are System/370 functions, such as block multiplexing, which are suppressed on initialization, and new System/370 functions, such as new instructions, which are specified in such a manner that they cause program exceptions on System/360.

Removal of USASCII-8 Mode

System/360 provides for USASCII-8 by a mode under control of PSW bit 12. USASCII-8 was a proposed zoned-decimal code that has since been rejected. When bit 12 of the System/360 PSW is one, the codes preferred for the USASCII-8 are generated for decimal results. When PSW bit 12 is zero, the codes preferred for EBCDIC are generated.

In System/370, the USASCII-8 mode and the associated meaning of PSW bit 12 are removed. In System/370, all instructions whose execution in System/360 depends on the setting of PSW bit 12 are executed generating the code preferred for EBCDIC.

Bit 12 of the PSW is handled in System/370 as follows:

- In models that do not have the extended-control (EC) mode installed, a one in PSW bit position 12 causes a program interruption for specification exception.
- In models that have the EC mode installed, a one in PSW bit position 12 causes the CPU to operate in the EC mode.

Handling Invalid Decimal Sign

In System/360, an invalid decimal operand causes the operation to be terminated. In System/370 the operation is suppressed, instead of terminated, when an invalid sign is detected. The action applies to all instructions that check the validity of decimal operands: ADD DECIMAL, SUBTRACT DECIMAL, ZERO AND ADD, COMPARE DECIMAL, MULTIPLY DECIMAL, DIVIDE DECIMAL, and CONVERT TO BINARY. It includes also the System/370 instruction SHIFT AND ROUND DECIMAL.

Recognizing Protection Exception in Edit

In System/360, when a pattern character in an EDIT or EDIT AND MARK operation is fetched from a location protected against storing but remains unchanged in the operation, it depends on the model whether or not the protection exception is recognized. In System/370, the protection exception in the above case is recognized.

Operation Code for HALT DEVICE

In System/370, the first eight bits of the operation code assigned to HALT DEVICE are the same as those assigned to HALT I/O, the distinction between the two instructions being specified by bit position 15. In System/360, bit position 15 is ignored, and HALT I/O is performed in both cases.

Extent of Logout Area

In System/360, the logout area starts with location 128 and extends through as many locations as the given model requires. Portions of this area are used for machine-check logout, and other portions may be used for channel logout. While no limit is set on the size of the logout area, the extent of the area used on most System/360 models is less than that stored by a comparable System/370 model.

On System/370, the machine-check interruption causes information to be stored at locations 216-239, 248-255, and 352-511. Additionally, the model may store logout information in the fixed logout area, locations 256-351, and the model may also have a machine-check extended logout (MCEL), which, on initialization, is specified to start at location 512. Channels may place logout information in the limited channel logout area, locations 176-179, and in the fixed logout area, locations 256-351.

Command Retry

System/370 channels may provide command retry, whereby the channel, in response to a signal from the device, can retry the execution of a channel command. Since I/O devices announced prior to System/370 do not signal for command retry, no problem of compatibility exists on these devices. However, some new devices, which would otherwise be compatible with former devices, do signal for command retry.

The occurrence of command retry will usually have no significant effect on the result produced by a channel program; however, the following is a list of some of the effects of command retry:

1. An immediate command specifying no chaining may result in setting condition code 0 rather than condition code 1.
2. Multiple PCI interruptions may be generated for a single CCW with the PCI flag.
3. Since CCWs may be refetched, programs which dynamically modify CCWs may be affected.
4. The residual count in the CSW reflects only the last execution of the command and does not necessarily reflect the maximum storage used in previous executions.

Logout on Channel Data Check

In System/360, logout is not permitted on channel data check. System/370 permits logout to occur when the channel causes an I/O interruption with the channel-data-check indication.

Channel Prefetching

In System/360, on an output operation as many as 16 bytes may be prefetched and buffered; similarly, with data chaining specified, the channel may prefetch the new CCW when up to 16 bytes remain to be transferred under control of the current CCW. In System/370, the restriction of 16 bytes is removed.

Appendix C. Lists of Instructions

The following four lists are of instructions arranged by name, mnemonic, operation code, and feature. Some models may offer instructions not appearing in the lists, 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 software uses where indication of invalid operation is required. It is improbable that these operation codes will ever be assigned to an instruction implemented in the CPU.

The listings in the Characteristics and Code columns mean:

A	Access exceptions
A_1	Addressing exception only
A_2	Addressing and translation-specification exceptions only
B	PER branch event
C	Condition code is set
CK	CPU-timer and clock-comparator feature
D	Data exception
DC	Direct-control feature
DF	Decimal-overflow exception
DK	Decimal-divide exception
DM	Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code
E	Exponent-overflow exception
EX	Execute exception
FK	Floating-point-divide exception
FP	Floating-point feature
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
MP	Multiprocessing feature
PD	Decimal feature
PK	PSW-key-handling feature
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
SW	Conditional-swapping feature
TR	Translation feature
U	Exponent-underflow exception
XP	Extended-precision floating-point feature
*	Bits 8-14 of the operation code are ignored
≠	Bits 8-15 of the operation code are ignored
\$	Causes serialization
$\$^1$	Causes serialization when the R_1 and R_2 fields contain all ones and all zeros, respectively.

Instructions Arranged by Name

Name	Mnemonic	Characteristics													Code	Page
ADD	AR	RR	C						IF				R		1A	117
ADD	A	RX	C			A			IF				R		5A	117
ADD DECIMAL	AP	SS	C	PD		A		D	DF					ST	FA	149
ADD HALFWORD	AH	RX	C			A			IF				R		4A	117
ADD LOGICAL	ALR	RR	C										R		1E	120
ADD LOGICAL	AL	RX	C			A							R		5E	120
ADD NORMALIZED (extended)	AXR	RR	C	XP			SP	U	E		LS				36	160
ADD NORMALIZED (long)	ADR	RR	C	FP			SP	U	E		LS				2A	160
ADD NORMALIZED (long)	AD	RX	C	FP		A	SP	U	E		LS				6A	160
ADD NORMALIZED (short)	AER	RR	C	FP			SP	U	E		LS				3A	160
ADD NORMALIZED (short)	AE	RX	C	FP		A	SP	U	E		LS				7A	160
ADD UNNORMALIZED (long)	AWR	RR	C	FP			SP		E		LS				2E	162
ADD UNNORMALIZED (long)	AW	RX	C	FP		A	SP		E		LS				6E	162
ADD UNNORMALIZED (short)	AUR	RR	C	FP			SP		E		LS				3E	162
ADD UNNORMALIZED (short)	AU	RX	C	FP		A	SP		E		LS				7E	162
AND	NR	RR	C										R		14	120
AND	N	RX	C			A							R		54	120
AND (character)	NC	SS	C			A								ST	D4	120
AND (immediate)	NI	SI	C			A								ST	94	120
BRANCH AND LINK	BALR	RR										B	R		05	121
BRANCH AND LINK	BAL	RX										B	R		45	121
BRANCH ON CONDITION	BCR	RR									$[1]	B			07	121
BRANCH ON CONDITION	BC	RX										B			47	121
BRANCH ON COUNT	BCTR	RR										B	R		06	122
BRANCH ON COUNT	BCT	RX										B	R		46	122
BRANCH ON INDEX HIGH	BXH	RS										B	R		86	122
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS										B	R		87	123
CLEAR I/O	CLRIO	S	C		M						$				9D01*	198
COMPARE	CR	RR	C												19	123
COMPARE	C	RX	C			A									59	123
COMPARE (long)	CDR	RR	C	FP			SP								29	163
COMPARE (long)	CD	RX	C	FP		A	SP								69	163
COMPARE (short)	CER	RR	C	FP			SP								39	163
COMPARE (short)	CE	RX	C	FP		A	SP								79	163
COMPARE AND SWAP	CS	RS	C	SW		A	SP				$		R	ST	BA	123
COMPARE DECIMAL	CP	SS	C	PD		A		D							F9	149
COMPARE DOUBLE AND SWAP	CDS	RS	C	SW		A	SP						R	ST	BB	124
COMPARE HALFWORD	CH	RX	C			A									49	125
COMPARE LOGICAL	CLR	RR	C												15	125
COMPARE LOGICAL	CL	RX	C			A									55	125
COMPARE LOGICAL (character)	CLC	SS	C			A									D5	125
COMPARE LOGICAL (immediate)	CLI	SI	C			A									95	125
COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS	C			A									BD	126
COMPARE LOGICAL LONG	CLCL	RR	C			A	SP				II		R		0F	126
CONVERT TO BINARY	CVB	RX				A		D		IK			R		4F	127
CONVERT TO DECIMAL	CVD	RX				A								ST	4E	128
DIAGNOSE					M	DM									83	103
DIVIDE	DR	RR					SP			IK			R		1D	128
DIVIDE	D	RY				A	SP			IK			R		5D	128
DIVIDE (long)	DDR	RR		FP			SP	U	E	FK					2D	163

Instructions Arranged by Name

Name	Mnemonic	Characteristics												Code	Page
DIVIDE (long)	DD	RX		FP		A	SP	U	E	FK				6D	163
DIVIDE (short)	DER	RR		FP			SP	U	E	FK				3D	163
DIVIDE (short)	DE	RX		FP		A	SP	U	E	FK				7D	163
DIVIDE DECIMAL	DP	SS		PD		A	SP	D		DK			ST	FD	149
EDIT	ED	SS	C	PD		A		D					ST	DE	150
EDIT AND MARK	EDMK	SS	C	PD		A		D				R	ST	DF	152
EXCLUSIVE OR	XR	RR	C									R		17	128
EXCLUSIVE OR	X	RX	C			A						R		57	129
EXCLUSIVE OR (character)	XC	SS	C			A							ST	D7	129
EXCLUSIVE OR (immediate)	XI	SI	C			A							ST	97	129
EXECUTE	EX	RX				A	SP				EX			44	129
HALT DEVICE	HDV	S	C		M						$			9E01*	199
HALT I/O	HIO	S	C		M						$			9E00*	202
HALVE (long)	HDR	RR		FP			SP	U						24	164
HALVE (short)	HER	RR		FP			SP	U						34	164
INSERT CHARACTER	IC	RX				A						R		43	130
INSERT CHARACTERS UNDER MASK	ICM	RS	C			A						R		BF	130
INSERT PSW KEY	IPK	S		PK	M							R		B20B	104
INSERT STORAGE KEY	ISK	RR			M	A_1	SP					R		09	105
LOAD	LR	RR										R		18	130
LOAD	L	RX				A						R		58	130
LOAD (long)	LDR	RR		FP			SP							28	165
LOAD (long)	LD	RX		FP		A	SP							68	165
LOAD (short)	LER	RR		FP			SP							38	165
LOAD (short)	LE	RX		FP		A	SP							78	165
LOAD ADDRESS	LA	RX										R		41	131
LOAD AND TEST	LTR	RR	C									R		12	131
LOAD AND TEST (long)	LTDR	RR	C	FP			SP							22	165
LOAD AND TEST (short)	LTER	RR	C	FP			SP							32	165
LOAD COMPLEMENT	LCR	RR	C						IF			R		13	131
LOAD COMPLEMENT (long)	LCDR	RR	C	FP			SP							23	166
LOAD COMPLEMENT (short)	LCER	RR	C	FP			SP							33	165
LOAD CONTROL	LCTL	RS			M	A	SP							B7	105
LOAD HALFWORD	LH	RX				A						R		48	131
LOAD MULTIPLE	LM	RS				A						R		98	132
LOAD NEGATIVE	LNR	RR	C									R		11	132
LOAD NEGATIVE (long)	LNDR	RR	C	FP			SP							21	166
LOAD NEGATIVE (short)	LNER	RR	C	FP			SP							31	166
LOAD POSITIVE	LPR	RR	C						IF			R		10	132
LOAD POSITIVE (long)	LPDR	RR	C	FP			SP							20	166
LOAD POSITIVE (short)	LPER	RR	C	FP			SP							30	166
LOAD PSW	LPSW	S	L		M	A	SP				$			82	105
LOAD REAL ADDRESS	LRA	RX	C	TR	M	A_2						R		B1	106
LOAD ROUNDED (extended to long)	LRDR	RR		XP			SP		E					25	167
LOAD ROUNDED (long to short)	LRER	RR		XP			SP		E					35	166
MONITOR CALL	MC	SI					SP				MO			AF	132
MOVE (character)	MVC	SS				A							ST	D2	133
MOVE (immediate)	MVI	SI				A							ST	92	133
MOVE LONG	MVCL	RR	C			A	SP				II	R	ST	0E	133
MOVE NUMERICS	MVN	SS				A							ST	D1	135

Instructions Arranged by Name

	Name	Mnemonic	Characteristics												Code	Page
	MOVE WITH OFFSET	MVO	SS				A							ST	F1	135
	MOVE ZONES	MVZ	SS				A							ST	D3	136
	MULTIPLY	MR	RR					SP					R		1C	136
	MULTIPLY	M	RX				A	SP					R		5C	136
	MULTIPLY (extended)	MXR	RR		XP			SP	U	E					26	167
	MULTIPLY (long)	MDR	RR		FP			SP	U	E					2C	167
	MULTIPLY (long)	MD	RX		FP		A	SP	U	E					6C	167
	MULTIPLY (long to extended)	MXDR	RR		XP			SP	U	E					27	167
	MULTIPLY (long to extended)	MXD	RX		XP		A	SP	U	E					67	167
	MULTIPLY (short to long)	MER	RR		FP			SP	U	E					3C	167
	MULTIPLY (short to long)	ME	RX		FP		A	SP	U	E					7C	167
	MULTIPLY DECIMAL	MP	SS		PD		A	SP	D					ST	FC	153
	MULTIPLY HALFWORD	MH	RX				A						R		4C	136
	OR	OR	RR	C									R		16	137
	OR	O	RX	C			A						R		56	137
	OR (character)	OC	SS	C			A							ST	D6	137
	OR (immediate)	OI	SI	C			A							ST	96	137
	PACK	PACK	SS				A							ST	F2	137
│	PURGE TLB	PTLB	S		TR	M					$				B20D	107
│	READ DIRECT	RDD	SI		DC	M	A				$			ST	85	107
	RESET REFERENCE BIT	RRB	S	C	TR	M	A_1								B213	107
	SET CLOCK	SCK	S	C		M	A	SP							B204	108
	SET CLOCK COMPARATOR	SCKC	S		CK	M	A	SP							B206	108
	SET CPU TIMER	SPT	S		CK	M	A	SP							B208	109
│	SET PREFIX	SPX	S		MP	M	A	SP			$				B210	109
	SET PROGRAM MASK	SPM	RR	L											04	138
	SET PSW KEY FROM ADDRESS	SPKA	S		PK	M									B20A	109
	SET STORAGE KEY	SSK	RR			M	A_1	SP							08	110
│	SET SYSTEM MASK	SSM	S			M	A	SP				SO			80	110
	SHIFT AND ROUND DECIMAL	SRP	SS	C	PD		A		D	DF				ST	F0	153
	SHIFT LEFT DOUBLE	SLDA	RS	C				SP		IF			R		8F	138
	SHIFT LEFT DOUBLE LOGICAL	SLDL	RS					SP					R		8D	139
	SHIFT LEFT SINGLE	SLA	RS	C						IF			R		8B	139
	SHIFT LEFT SINGLE LOGICAL	SLL	RS										R		89	139
	SHIFT RIGHT DOUBLE	SRDA	RS	C				SP					R		8E	140
	SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS					SP					R		8C	140
	SHIFT RIGHT SINGLE	SRA	RS	C									R		8A	140
	SHIFT RIGHT SINGLE LOGICAL	SRL	RS										R		88	141
│	SIGNAL PROCESSOR	SIGP	RS	C	MP	M					$		R		AE	110
│	START I/O	SIO	S	C		M					$				9C00*	204
│	START I/O FAST RELEASE	SIOF	S	C		M					$				9C01*	204
	STORE	ST	RX				A							ST	50	141
	STORE (long)	STD	RX		FP		A	SP						ST	60	169
	STORE (short)	STE	RX		FP		A	SP						ST	70	168
│	STORE CHANNEL ID	STIDC	S	C		M					$				B203	206
	STORE CHARACTER	STC	RX				A							ST	42	141
	STORE CHARACTERS UNDER MASK	STCM	RS				A							ST	BE	141
│	STORE CLOCK	STCK	S	C			A				$			ST	B205	141
	STORE CLOCK COMPARATOR	STCKC	S		CK	M	A	SP						ST	B207	111
	STORE CONTROL	STCTL	RS			M	A	SP						ST	B6	111

Instructions Arranged by Name

Name	Mnemonic	Characteristics											Code	Page
STORE CPU ADDRESS	STAP	S		MP	M	A	SP					ST	B212	112
STORE CPU ID	STIDP	S			M	A	SP					ST	B202	112
STORE CPU TIMER	STPT	S		CK	M	A	SP					ST	B209	113
STORE HALFWORD	STH	RX				A						ST	40	142
STORE MULTIPLE	STM	RS				A						ST	90	142
STORE PREFIX	STPX	S		MP	M	A	SP					ST	B211	113
STORE THEN AND SYSTEM MASK	STNSM	SI		TR	M	A						ST	AC	113
STORE THEN OR SYSTEM MASK	STOSM	SI		TR	M	A	SP					ST	AD	114
SUBTRACT	SR	RR	C						IF		R		1B	143
SUBTRACT	S	RX	C			A			IF		R		5B	143
SUBTRACT DECIMAL	SP	SS	C	PD		A		D	DF			ST	FB	154
SUBTRACT HALFWORD	SH	RX	C			A			IF		R		4B	143
SUBTRACT LOGICAL	SLR	RR	C								R		1F	143
SUBTRACT LOGICAL	SL	RX	C			A					R		5F	143
SUBTRACT NORMALIZED (extended)	SXR	RR	C	XP			SP	U	E	LS			37	169
SUBTRACT NORMALIZED (long)	SDR	RR	C	FP			SP	U	E	LS			2B	169
SUBTRACT NORMALIZED (long)	SD	RX	C	FP		A	SP	U	E	LS			6B	169
SUBTRACT NORMALIZED (short)	SER	RR	C	FP			SP	U	E	LS			3B	169
SUBTRACT NORMALIZED (short)	SE	RX	C	FP		A	SP	U	E	LS			7B	169
SUBTRACT UNNORMALIZED (long)	SWR	RR	C	FP			SP		E	LS			2F	170
SUBTRACT UNNORMALIZED (long)	SW	RX	C	FP		A	SP		E	LS			6F	170
SUBTRACT UNNORMALIZED (short)	SUR	RR	C	FP			SP		E	LS			3F	169
SUBTRACT UNNORMALIZED (short)	SU	RX	C	FP		A	SP		E	LS			7F	169
SUPERVISOR CALL	SVC	RR								$			0A	144
TEST AND SET	TS	S	C			A				$		ST	93	144
TEST CHANNEL	TCH	S	C		M					$			9F00≠	207
TEST I/O	TIO	S	C		M					$			9D00*	208
TEST UNDER MASK	TM	SI	C			A							91	145
TRANSLATE	TR	SS				A						ST	DC	145
TRANSLATE AND TEST	TRT	SS	C			A					R		DD	145
UNPACK	UNPK	SS				A						ST	F3	146
WRITE DIRECT	WRD	SI		DC	M	A				$			84	114
ZERO AND ADD	ZAP	SS	C	PD		A		D	DF			ST	F8	155

Instructions Arranged by Mnemonic

Mnemonic	Name	Characteristics													Code	Page
A	ADD	RX	C			A			IF				R		5A	117
AD	ADD NORMALIZED (long)	RX	C	FP		A	SP	U	E		LS				6A	160
ADR	ADD NORMALIZED (long)	RR	C	FP			SP	U	E		LS				2A	160
AE	ADD NORMALIZED (short)	RX	C	FP		A	SP	U	E		LS				7A	160
AER	ADD NORMALIZED (short)	RR	C	FP			SP	U	E		LS				3A	160
AH	ADD HALFWORD	RX	C			A			IF				R		4A	117
AL	ADD LOGICAL	RX	C			A							R		5E	120
ALR	ADD LOGICAL	RR	C										R		1E	120
AP	ADD DECIMAL	SS	C	PD		A		D	DF					ST	FA	149
AR	ADD	RR	C						IF				R		1A	117
AU	ADD UNNORMALIZED (short)	RX	C	FP		A	SP		E		LS				7E	162
AUR	ADD UNNORMALIZED (short)	RR	C	FP			SP		E		LS				3E	162
AW	ADD UNNORMALIZED (long)	RX	C	FP		A	SP		E		LS				6E	162
AWR	ADD UNNORMALIZED (long)	RR	C	FP			SP		E		LS				2E	162
AXR	ADD NORMALIZED (extended)	RR	C	XP			SP	U	E		LS				36	160
BAL	BRANCH AND LINK	RX										B	R		45	121
BALR	BRANCH AND LINK	RR										B	R		05	121
BC	BRANCH ON CONDITION	RX										B			47	121
BCR	BRANCH ON CONDITION	RR									^{1}	B			07	121
BCT	BRANCH ON COUNT	RX										B	R		46	122
BCTR	BRANCH ON COUNT	RR										B	R		06	122
BXH	BRANCH ON INDEX HIGH	RS										B	R		86	122
BXLE	BRANCH ON INDEX LOW OR EQUAL	RS										B	R		87	123
C	COMPARE	RX	C			A									59	123
CD	COMPARE (long)	RX	C	FP		A	SP								69	163
CDR	COMPARE (long)	RR	C	FP			SP								29	163
CDS	COMPARE DOUBLE AND SWAP	RS	C	SW		A	SP				$		R	ST	BB	124
CE	COMPARE (short)	RX	C	FP		A	SP								79	163
CER	COMPARE (short)	RR	C	FP			SP								39	163
CH	COMPARE HALFWORD	RX	C			A									49	125
CL	COMPARE LOGICAL	RX	C			A									55	125
CLC	COMPARE LOGICAL (character)	SS	C			A									D5	125
CLCL	COMPARE LOGICAL LONG	RR	C			A	SP				II		R		0F	126
CLI	COMPARE LOGICAL (immediate)	SI	C			A									95	125
CLM	COMPARE LOGICAL CHARACTERS UNDER MASK	RS	C			A									BD	126
CLR	COMPARE LOGICAL	RR	C												15	125
CLRIO	CLEAR I/O	S	C		M						$				9D01*	198
CP	COMPARE DECIMAL	SS	C	PD		A		D							F9	149
CR	COMPARE	RR	C												19	123
CS	COMPARE AND SWAP	RS	C	SW		A	SP				$		R	ST	BA	123
CVB	CONVERT TO BINARY	RX				A		D		IK			R		4F	127
CVD	CONVERT TO DECIMAL	RX				A								ST	4E	128
D	DIVIDE	RX				A	SP			IK			R		5D	128
DD	DIVIDE (long)	RX		FP		A	SP	U	E	FK					6D	163
DDR	DIVIDE (long)	RR		FP			SP	U	E	FK					2D	163
DE	DIVIDE (short)	RX		FP		A	SP	U	E	FK					7D	163
DER	DIVIDE (short)	RR		FP			SP	U	E	FK					3D	163
DP	DIVIDE DECIMAL	SS		PD		A	SP	D		DK				ST	FD	149
DR	DIVIDE	RR					SP			IK			R		1D	128
ED	EDIT	SS	C	PD		A		D						ST	DE	150

Instructions Arranged by Mnemonic

Mnemonic	Name	Characteristics											Code	Page
EDMK	EDIT AND MARK	SS	C	PD		A		D			R	ST	DF	152
EX	EXECUTE	RX				A	SP			EX			44	129
HDR	HALVE (long)	RR		FP			SP	U					24	164
HDV	HALT DEVICE	S	C		M					$			9E01*	199
HER	HALVE (short)	RR		FP			SP	U					34	164
HIO	HALT I/O	S	C		M					$			9E00*	202
IC	INSERT CHARACTER	RX				A					R		43	130
ICM	INSERT CHARACTERS UNDER MASK	RS	C			A					R		BF	130
IPK	INSERT PSW KEY	S		PK	M						R		B20B	104
ISK	INSERT STORAGE KEY	RR			M	A_1	SP				R		09	105
L	LOAD	RX				A					R		58	130
LA	LOAD ADDRESS	RX									R		41	131
LCDR	LOAD COMPLEMENT (long)	RR	C	FP			SP						23	166
LCER	LOAD COMPLEMENT (short)	RR	C	FP			SP						33	165
LCR	LOAD COMPLEMENT	RR	C						IF		R		13	131
LCTL	LOAD CONTROL	RS			M	A	SP						B7	105
LD	LOAD (long)	RX		FP		A	SP						68	165
LDR	LOAD (long)	RR		FP			SP						28	165
LE	LOAD (short)	RX		FP		A	SP						78	165
LER	LOAD (short)	RR		FP			SP						38	165
LH	LOAD HALFWORD	RX				A					R		48	131
LM	LOAD MULTIPLE	RS				A					R		98	132
LNDR	LOAD NEGATIVE (long)	RR	C	FP			SP						21	166
LNER	LOAD NEGATIVE (short)	RR	C	FP			SP						31	166
LNR	LOAD NEGATIVE	RR	C								R		11	132
LPDR	LOAD POSITIVE (long)	RR	C	FP			SP						20	166
LPER	LOAD POSITIVE (short)	RR	C	FP			SP						30	166
LPR	LOAD POSITIVE	RR	C						IF		R		10	132
LPSW	LOAD PSW	S	L		M	A	SP			$			82	105
LR	LOAD	RR									R		18	130
LRA	LOAD REAL ADDRESS	RX	C	TR	M	A_2					R		B1	106
LRDR	LOAD ROUNDED (extended to long)	RR		XP			SP		E				25	167
LRER	LOAD ROUNDED (long to short)	RR		XP			SP		E				35	166
LTDR	LOAD AND TEST (long)	RR	C	FP			SP						22	165
LTER	LOAD AND TEST (short)	RR	C	FP			SP						32	165
LTR	LOAD AND TEST	RR	C								R		12	131
M	MULTIPLY	RX				A	SP				R		5C	136
MC	MONITOR CALL	SI					SP			MO			AF	132
MD	MULTIPLY (long)	RX		FP		A	SP	U	E				6C	167
MDR	MULTIPLY (long)	RR		FP			SP	U	E				2C	167
ME	MULTIPLY (short to long)	RX		FP		A	SP	U	E				7C	167
MER	MULTIPLY (short to long)	RR		FP			SP	U	E				3C	167
MH	MULTIPLY HALFWORD	RX				A					R		4C	136
MP	MULTIPLY DECIMAL	SS		PD		A	SP	D				ST	FC	153
MR	MULTIPLY	RR					SP				R		1C	136
MVC	MOVE (character)	SS				A						ST	D2	133
MVCL	MOVE LONG	RR	C			A	SP			II	R	ST	0E	133
MVI	MOVE (immediate)	SI				A						ST	92	133
MVN	MOVE NUMERICS	SS				A						ST	D1	135
MVO	MOVE WITH OFFSET	SS				A						ST	F1	135

Instructions Arranged by Mnemonic

Mnemonic	Name	Characteristics												Code	Page
MVZ	MOVE ZONES	SS				A							ST	D3	136
MXD	MULTIPLY (long to extended)	RX		XP		A	SP	U	E					67	167
MXDR	MULTIPLY (long to extended)	RR		XP			SP	U	E					27	167
MXR	MULTIPLY (extended)	RR		XP			SP	U	E					26	167
N	AND	RX	C			A						R		54	120
NC	AND (character)	SS	C			A							ST	D4	120
NI	AND (immediate)	SI	C			A							ST	94	120
NR	AND	RR	C									R		14	120
O	OR	RX	C			A						R		56	137
OC	OR (character)	SS	C			A							ST	D6	137
OI	OR (immediate)	SI	C			A							ST	96	137
OR	OR	RR	C									R		16	137
PACK	PACK	SS				A							ST	F2	137
PTLB	PURGE TLB	S		TR	M					$				B20D	107
RDD	READ DIRECT	SI		DC	M	A				$			ST	85	107
RRB	RESET REFERENCE BIT	S	C	TR	M	A_1								B213	107
S	SUBTRACT	RX	C			A			IF			R		5B	143
SCK	SET CLOCK	S	C		M	A	SP							B204	108
SCKC	SET CLOCK COMPARATOR	S		CK	M	A	SP							B206	108
SD	SUBTRACT NORMALIZED (long)	RX	C	FP		A	SP	U	E	LS				6B	169
SDR	SUBTRACT NORMALIZED (long)	RR	C	FP			SP	U	E	LS				2B	169
SE	SUBTRACT NORMALIZED (short)	RX	C	FP		A	SP	U	E	LS				7B	169
SER	SUBTRACT NORMALIZED (short)	RR	C	FP			SP	U	E	LS				3B	169
SH	SUBTRACT HALFWORD	RX	C			A			IF			R		4B	143
SIGP	SIGNAL PROCESSOR	RS	C	MP	M							R		AE	110
SIO	START I/O	S	C		M					$				9C00*	204
SIOF	START I/O FAST RELEASE	S	C		M					$				9C01*	204
SL	SUBTRACT LOGICAL	RX	C			A						R		5F	143
SLA	SHIFT LEFT SINGLE	RS	C						IF			R		8B	139
SLDA	SHIFT LEFT DOUBLE	RS	C				SP		IF			R		8F	138
SLDL	SHIFT LEFT DOUBLE LOGICAL	RS					SP					R		8D	139
SLL	SHIFT LEFT SINGLE LOGICAL	RS										R		89	139
SLR	SUBTRACT LOGICAL	RR	C									R		1F	143
SP	SUBTRACT DECIMAL	SS	C	PD		A		D	DF				ST	FB	154
SPKA	SET PSW KEY FROM ADDRESS	S		PK	M									B20A	109
SPM	SET PROGRAM MASK	RR	L											04	138
SPT	SET CPU TIMER	S		CK	M	A	SP							B208	109
SPX	SET PREFIX	S		MP	M	A	SP			$				B210	109
SR	SUBTRACT	RR	C						IF			R		1B	143
SRA	SHIFT RIGHT SINGLE	RS	C									R		8A	140
SRDA	SHIFT RIGHT DOUBLE	RS	C				SP					R		8E	140
SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS					SP					R		8C	140
SRL	SHIFT RIGHT SINGLE LOGICAL	RS										R		88	141
SRP	SHIFT AND ROUND DECIMAL	SS	C	PD		A		D	DF				ST	F0	153
SSK	SET STORAGE KEY	RR			M	A_1	SP							08	110
SSM	SET SYSTEM MASK	S			M	A	SP				SO			80	110
ST	STORE	RX				A							ST	50	141
STAP	STORE CPU ADDRESS	S		MP	M	A	SP						ST	B212	112
STC	STORE CHARACTER	RX				A							ST	42	141
STCK	STORE CLOCK	S	C			A				$			ST	B205	141

Instructions Arranged by Mnemonic

Mnemonic	Name	Characteristics											Code	Page
STCKC	STORE CLOCK COMPARATOR	S		CK	M	A	SP					ST	B207	111
STCM	STORE CHARACTERS UNDER MASK	RS				A						ST	BE	141
STCTL	STORE CONTROL	RS			M	A	SP					ST	B6	111
STD	STORE (long)	RX		FP		A	SP					ST	60	169
STE	STORE (short)	RX		FP		A	SP					ST	70	168
STH	STORE HALFWORD	RX				A						ST	40	142
STIDC	STORE CHANNEL ID	S	C		M					$			B203	206
STIDP	STORE CPU ID	S			M	A	SP					ST	B202	112
STM	STORE MULTIPLE	RS				A						ST	90	142
STNSM	STORE THEN AND SYSTEM MASK	SI		TR	M	A						ST	AC	113
STOSM	STORE THEN OR SYSTEM MASK	SI		TR	M	A	SP					ST	AD	114
STPT	STORE CPU TIMER	S		CK	M	A	SP					ST	B209	113
STPX	STORE PREFIX	S		MP	M	A	SP					ST	B211	113
SU	SUBTRACT UNNORMALIZED (short)	RX	C	FP		A	SP		E	LS			7F	169
SUR	SUBTRACT UNNORMALIZED (short)	RR	C	FP			SP		E	LS			3F	169
SVC	SUPERVISOR CALL	RR								$			0A	144
SW	SUBTRACT UNNORMALIZED (long)	RX	C	FP		A	SP		E	LS			6F	170
SWR	SUBTRACT UNNORMALIZED (long)	RR	C	FP			SP		E	LS			2F	170
SXR	SUBTRACT NORMALIZED (extended)	RR	C	XP			SP	U	E	LS			37	169
TCH	TEST CHANNEL	S	C		M					$			9F00≠	207
TIO	TEST I/O	S	C		M					$			9D00*	208
TM	TEST UNDER MASK	SI	C			A							91	145
TR	TRANSLATE	SS				A						ST	DC	145
TRT	TRANSLATE AND TEST	SS	C			A					R		DD	145
TS	TEST AND SET	S	C			A				$		ST	93	144
UNPK	UNPACK	SS				A						ST	F3	146
WRD	WRITE DIRECT	SI		DC	M	A				$			84	114
X	EXCLUSIVE OR	RX	C			A					R		57	129
XC	EXCLUSIVE OR (character)	SS	C			A						ST	D7	129
XI	EXCLUSIVE OR (immediate)	SI	C			A						ST	97	129
XR	EXCLUSIVE OR	RR	C								R		17	128
ZAP	ZERO AND ADD	SS	C	PD		A		D	DF			ST	F8	155
	DIAGNOSE				M	DM							83	103

Instructions Arranged by Operation Code

Code	Name	Mnemonic	Characteristics													Page
04	SET PROGRAM MASK	SPM	RR	L												138
05	BRANCH AND LINK	BALR	RR										B	R		121
06	BRANCH ON COUNT	BCTR	RR										B	R		122
07	BRANCH ON CONDITION	BCR	RR									$[1]	B			121
08	SET STORAGE KEY	SSK	RR			M	A_1	SP								110
09	INSERT STORAGE KEY	ISK	RR			M	A_1	SP						R		105
0A	SUPERVISOR CALL	SVC	RR									$				144
0E	MOVE LONG	MVCL	RR	C			A	SP				II		R	ST	133
0F	COMPARE LOGICAL LONG	CLCL	RR	C			A	SP				II		R		126
10	LOAD POSITIVE	LPR	RR	C						IF				R		132
11	LOAD NEGATIVE	LNR	RR	C										R		132
12	LOAD AND TEST	LTR	RR	C										R		131
13	LOAD COMPLEMENT	LCR	RR	C						IF				R		131
14	AND	NR	RR	C										R		120
15	COMPARE LOGICAL	CLR	RR	C												125
16	OR	OR	RR	C										R		137
17	EXCLUSIVE OR	XR	RR	C										R		128
18	LOAD	LR	RR											R		130
19	COMPARE	CR	RR	C												123
1A	ADD	AR	RR	C						IF				R		117
1B	SUBTRACT	SR	RR	C						IF				R		143
1C	MULTIPLY	MR	RR					SP						R		136
1D	DIVIDE	DR	RR					SP			IK			R		128
1E	ADD LOGICAL	ALR	RR	C										R		120
1F	SUBTRACT LOGICAL	SLR	RR	C										R		143
20	LOAD POSITIVE (long)	LPDR	RR	C	FP			SP								166
21	LOAD NEGATIVE (long)	LNDR	RR	C	FP			SP								166
22	LOAD AND TEST (long)	LTDR	RR	C	FP			SP								165
23	LOAD COMPLEMENT (long)	LCDR	RR	C	FP			SP								166
24	HALVE (long)	HDR	RR		FP			SP	U							164
25	LOAD ROUNDED (extended to long)	LRDR	RR		XP			SP		E						167
26	MULTIPLY (extended)	MXR	RR		XP			SP	U	E						167
27	MULTIPLY (long to extended)	MXDR	RR		XP			SP	U	E						167
28	LOAD (long)	LDR	RR		FP			SP								165
29	COMPARE (long)	CDR	RR	C	FP			SP								163
2A	ADD NORMALIZED (long)	ADR	RR	C	FP			SP	U	E		LS				160
2B	SUBTRACT NORMALIZED (long)	SDR	RR	C	FP			SP	U	E		LS				169
2C	MULTIPLY (long)	MDR	RR		FP			SP	U	E						167
2D	DIVIDE (long)	DDR	RR		FP			SP	U	E	FK					163
2E	ADD UNNORMALIZED (long)	AWR	RR	C	FP			SP		E		LS				162
2F	SUBTRACT UNNORMALIZED (long)	SWR	RR	C	FP			SP		E		LS				170
30	LOAD POSITIVE (short)	LPER	RR	C	FP			SP								166
31	LOAD NEGATIVE (short)	LNER	RR	C	FP			SP								166
32	LOAD AND TEST (short)	LTER	RR	C	FP			SP								165
33	LOAD COMPLEMENT (short)	LCER	RR	C	FP			SP								165
34	HALVE (short)	HER	RR		FP			SP	U							164
35	LOAD ROUNDED (long to short)	LRER	RR		XP			SP		E						166
36	ADD NORMALIZED (extended)	AXR	RR	C	XP			SP	U	E		LS				160
37	SUBTRACT NORMALIZED (extended)	SXR	RR	C	XP			SP	U	E		LS				169
38	LOAD (short)	LER	RR		FP			SP								165

Instructions Arranged by Operation Code

Code	Name	Mnemonic	Characteristics												Page
39	COMPARE (short)	CER	RR	C	FP		SP								163
3A	ADD NORMALIZED (short)	AER	RR	C	FP		SP	U	E		LS				160
3B	SUBTRACT NORMALIZED (short)	SER	RR	C	FP		SP	U	E		LS				169
3C	MULTIPLY (short to long)	MER	RR		FP		SP	U	E						167
3D	DIVIDE (short)	DER	RR		FP		SP	U	E	FK					163
3E	ADD UNNORMALIZED (short)	AUR	RR	C	FP		SP		E		LS				162
3F	SUBTRACT UNNORMALIZED (short)	SUR	RR	C	FP		SP		E		LS				169
40	STORE HALFWORD	STH	RX			A								ST	142
41	LOAD ADDRESS	LA	RX										R		131
42	STORE CHARACTER	STC	RX			A								ST	141
43	INSERT CHARACTER	IC	RX			A							R		130
44	EXECUTE	EX	RX			A	SP				EX				129
45	BRANCH AND LINK	BAL	RX									B	R		121
46	BRANCH ON COUNT	BCT	RX									B	R		122
47	BRANCH ON CONDITION	BC	RX									B			121
48	LOAD HALFWORD	LH	RX			A							R		131
49	COMPARE HALFWORD	CH	RX	C		A									125
4A	ADD HALFWORD	AH	RX	C		A			IF				R		117
4B	SUBTRACT HALFWORD	SH	RX	C		A			IF				R		143
4C	MULTIPLY HALFWORD	MH	RX			A							R		136
4E	CONVERT TO DECIMAL	CVD	RX			A								ST	128
4F	CONVERT TO BINARY	CVB	RX			A		D		IK			R		127
50	STORE	ST	RX			A								ST	141
54	AND	N	RX	C		A							R		120
55	COMPARE LOGICAL	CL	RX	C		A									125
56	OR	O	RX	C		A							R		137
57	EXCLUSIVE OR	X	RX	C		A							R		129
58	LOAD	L	RX			A							R		130
59	COMPARE	C	RX	C		A									123
5A	ADD	A	RX	C		A			IF				R		117
5B	SUBTRACT	S	RX	C		A			IF				R		143
5C	MULTIPLY	M	RX			A	SP						R		136
5D	DIVIDE	D	RX			A	SP			IK			R		128
5E	ADD LOGICAL	AL	RX	C		A							R		120
5F	SUBTRACT LOGICAL	SL	RX	C		A							R		143
60	STORE (long)	STD	RX		FP	A	SP							ST	169
67	MULTIPLY (long to extended)	MXD	RX		XP	A	SP	U	E						167
68	LOAD (long)	LD	RX		FP	A	SP								165
69	COMPARE (long)	CD	RX	C	FP	A	SP								163
6A	ADD NORMALIZED (long)	AD	RX	C	FP	A	SP	U	E		LS				160
6B	SUBTRACT NORMALIZED (long)	SD	RX	C	FP	A	SP	U	E		LS				169
6C	MULTIPLY (long)	MD	RX		FP	A	SP	U	E						167
6D	DIVIDE (long)	DD	RX		FP	A	SP	U	E	FK					163
6E	ADD UNNORMALIZED (long)	AW	RX	C	FP	A	SP		E		LS				162
6F	SUBTRACT UNNORMALIZED (long)	SW	RX	C	FP	A	SP		E		LS				170
70	STORE (short)	STE	RX		FP	A	SP							ST	168
78	LOAD (short)	LE	RX		FP	A	SP								165
79	COMPARE (short)	CE	RX	C	FP	A	SP								163
7A	ADD NORMALIZED (short)	AE	RX	C	FP	A	SP	U	E		LS				160
7B	SUBTRACT NORMALIZED (short)	SE	RX	C	FP	A	SP	U	E		LS				169

Instructions Arranged by Operation Code

Code	Name	Mnemonic	Characteristics													Page
7C	MULTIPLY (short to long)	ME	RX		FP		A	SP	U	E						167
7D	DIVIDE (short)	DE	RX		FP		A	SP	U	E	FK					163
7E	ADD UNNORMALIZED (short)	AU	RX	C	FP		A	SP		E		LS				162
7F	SUBTRACT UNNORMALIZED (short)	SU	RX	C	FP		A	SP		E		LS				169
80	SET SYSTEM MASK	SSM	S			M	A	SP					SO			110
82	LOAD PSW	LPSW	S	L		M	A	SP				$				105
83	DIAGNOSE					M	DM									103
84	WRITE DIRECT	WRD	SI		DC	M	A					$				114
85	READ DIRECT	RDD	SI		DC	M	A					$			ST	107
86	BRANCH ON INDEX HIGH	BXH	RS										B	R		122
87	BRANCH ON INDEX LOW OR EQUAL	BXLE	RS										B	R		123
88	SHIFT RIGHT SINGLE LOGICAL	SRL	RS											R		141
89	SHIFT LEFT SINGLE LOGICAL	SLL	RS											R		139
8A	SHIFT RIGHT SINGLE	SRA	RS	C										R		139
8B	SHIFT LEFT SINGLE	SLA	RS	C						IF				R		139
8C	SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS					SP						R		140
8D	SHIFT LEFT DOUBLE LOGICAL	SLDL	RS					SP						R		139
8E	SHIFT RIGHT DOUBLE	SRDA	RS	C				SP						R		140
8F	SHIFT LEFT DOUBLE	SLDA	RS	C				SP		IF				R		138
90	STORE MULTIPLE	STM	RS				A								ST	142
91	TEST UNDER MASK	TM	SI	C			A									145
92	MOVE (immediate)	MVI	SI				A								ST	133
93	TEST AND SET	TS	S	C			A					$			ST	144
94	AND (immediate)	NI	SI	C			A								ST	120
95	COMPARE LOGICAL (immediate)	CLI	SI	C			A									125
96	OR (immediate)	OI	SI	C			A								ST	137
97	EXCLUSIVE OR (immediate)	XI	SI	C			A								ST	129
98	LOAD MULTIPLE	LM	RS				A							R		132
9C00*	START I/O	SIO	S	C		M						$				204
9C01*	START I/O FAST RELEASE	SIOF	S	C		M						$				204
9D00*	TEST I/O	TIO	S	C		M						$				208
9D01*	CLEAR I/O	CLRIO	S	C		M						$				198
9E00*	HALT I/O	HIO	S	C		M						$				202
9E01*	HALT DEVICE	HDV	S	C		M						$				199
9F00≠	TEST CHANNEL	TCH	S	C		M						$				207
AC	STORE THEN AND SYSTEM MASK	STNSM	SI		TR	M	A								ST	113
AD	STORE THEN OR SYSTEM MASK	STOSM	SI		TR	M	A	SP							ST	114
AE	SIGNAL PROCESSOR	SIGP	RS	C	MP	M						$		R		110
AF	MONITOR CALL	MC	SI					SP				MO				132
B1	LOAD REAL ADDRESS	LRA	RX	C	TR	M	A_2							R		106
B202	STORE CPU ID	STIDP	S			M	A	SP							ST	112
B203	STORE CHANNEL ID	STIDC	S	C		M						$				206
B204	SET CLOCK	SCK	S	C		M	A	SP								108
B205	STORE CLOCK	STCK	S	C			A					$			ST	141
B206	SET CLOCK COMPARATOR	SCKC	S		CK	M	A	SP								108
B207	STORE CLOCK COMPARATOR	STCKC	S		CK	M	A	SP							ST	111
B208	SET CPU TIMER	SPT	S		CK	M	A	SP								109
B209	STORE CPU TIMER	STPT	S		CK	M	A	SP							ST	113
B20A	SET PSW KEY FROM ADDRESS	SPKA	S		PK	M										109
B20B	INSERT PSW KEY	IPK	S		PK	M								R		104

Instructions Arranged by Operation Code

Code	Name	Mnemonic	Characteristics											Page
B20D	PURGE TLB	PTLB	S		TR	M								107
B210	SET PREFIX	SPX	S		MP	M	A	SP						109
B211	STORE PREFIX	STPX	S		MP	M	A	SP					ST	113
B212	STORE CPU ADDRESS	STAP	S		MP	M	A	SP					ST	112
B213	RESET REFERENCE BIT	RRB	S	C	TR	M	A_1							107
B6	STORE CONTROL	STCTL	RS			M	A	SP					ST	111
B7	LOAD CONTROL	LCTL	RS			M	A	SP						105
BA	COMPARE AND SWAP	CS	RS	C	SW		A	SP				R	ST	123
BB	COMPARE DOUBLE AND SWAP	CDS	RS	C	SW		A	SP				R	ST	124
BD	COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS	C			A							126
BE	STORE CHARACTERS UNDER MASK	STCM	RS				A						ST	141
BF	INSERT CHARACTERS UNDER MASK	ICM	RS	C			A					R		130
D1	MOVE NUMERICS	MVN	SS				A						ST	135
D2	MOVE (character)	MVC	SS				A						ST	133
D3	MOVE ZONES	MVZ	SS				A						ST	136
D4	AND (character)	NC	SS	C			A						ST	120
D5	COMPARE LOGICAL (character)	CLC	SS	C			A							125
D6	OR (character)	OC	SS	C			A						ST	137
D7	EXCLUSIVE OR (character)	XC	SS	C			A						ST	129
DC	TRANSLATE	TR	SS				A						ST	145
DD	TRANSLATE AND TEST	TRT	SS	C			A					R		145
DE	EDIT	ED	SS	C	PD		A		D				ST	150
DF	EDIT AND MARK	EDMK	SS	C	PD		A		D			R	ST	152
F0	SHIFT AND ROUND DECIMAL	SRP	SS	C	PD		A		D	DF			ST	153
F1	MOVE WITH OFFSET	MVO	SS				A						ST	135
F2	PACK	PACK	SS				A						ST	137
F3	UNPACK	UNPK	SS				A						ST	146
F8	ZERO AND ADD	ZAP	SS	C	PD		A		D	DF			ST	155
F9	COMPARE DECIMAL	CP	SS	C	PD		A		D					149
FA	ADD DECIMAL	AP	SS	C	PD		A		D	DF			ST	149
FB	SUBTRACT DECIMAL	SP	SS	C	PD		A		D	DF			ST	154
FC	MULTIPLY DECIMAL	MP	SS		PD		A	SP	D				ST	153
FD	DIVIDE DECIMAL	DP	SS		PD		A	SP	D		DK		ST	149

Instructions Arranged by Feature

Standard Instruction Set

Name	Mnemonic	Characteristics												Code	Page
ADD	AR	RR	C					IF				R		1A	117
ADD	A	RX	C		A			IF				R		5A	117
ADD HALFWORD	AH	RX	C		A			IF				R		4A	117
ADD LOGICAL	ALR	RR	C									R		1E	120
ADD LOGICAL	AL	RX	C		A							R		5E	120
AND	NR	RR	C									R		14	120
AND	N	RX	C		A							R		54	120
AND (character)	NC	SS	C		A								ST	D4	120
AND (immediate)	NI	SI	C		A								ST	94	120
BRANCH AND LINK	BALR	RR									B	R		05	121
BRANCH AND LINK	BAL	RX									B	R		45	121
BRANCH ON CONDITION	BCR	RR								$[1]	B			07	121
BRANCH ON CONDITION	BC	RX									B			47	121
BRANCH ON COUNT	BCTR	RR									B	R		06	122
BRANCH ON COUNT	BCT	RX									B	R		46	122
BRANCH ON INDEX HIGH	BXH	RS									B	R		86	122
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS									B	R		87	123
CLEAR I/O	CLRIO	S	C	M						$				9D01*	198
COMPARE	CR	RR	C											19	123
COMPARE	C	RX	C		A									59	123
COMPARE HALFWORD	CH	RX	C		A									49	125
COMPARE LOGICAL	CLR	RR	C											15	125
COMPARE LOGICAL	CL	RX	C		A									55	125
COMPARE LOGICAL (character)	CLC	SS	C		A									D5	125
COMPARE LOGICAL (immediate)	CLI	SI	C		A									95	125
COMPARE LOGICAL CHARACTERS UNDER MASK	CLM	RS	C		A									BD	126
COMPARE LOGICAL LONG	CLCL	RR	C		A	SP				II		R		0F	126
CONVERT TO BINARY	CVB	RX			A		D		IK			R		4F	127
CONVERT TO DECIMAL	CVD	RX			A								ST	4E	128
DIAGNOSE				M	DM									83	103
DIVIDE	DR	RR				SP			IK			R		1D	128
DIVIDE	D	RX			A	SP			IK			R		5D	128
EXCLUSIVE OR	XR	RR	C									R		17	128
EXCLUSIVE OR	X	RX	C		A							R		57	129
EXCLUSIVE OR (character)	XC	SS	C		A								ST	D7	129
EXCLUSIVE OR (immediate)	XI	SI	C		A								ST	97	129
EXECUTE	EX	RX			A	SP				EX				44	129
HALT DEVICE	HDV	S	C	M						$				9E01*	199
HALT I/O	HIO	S	C	M						$				9E00*	164
INSERT CHARACTER	IC	RX			A							R		43	130
INSERT CHARACTERS UNDER MASK	ICM	RS	C		A							R		BF	130
INSERT STORAGE KEY	ISK	RR		M	A_1	SP						R		09	105
LOAD	LR	RR										R		18	130
LOAD	L	RX			A							R		58	130
LOAD ADDRESS	LA	RX										R		41	131
LOAD AND TEST	LTR	RR	C									R		12	131
LOAD COMPLEMENT	LCR	RR	C					IF				R		13	131
LOAD CONTROL	LCTL	RS		M	A	SP								B7	105
LOAD HALFWORD	LH	RX			A							R		48	131
LOAD MULTIPLE	LM	RS			A							R		98	132

Instructions Arranged by Feature

Standard Instruction Set (continued)

Name	Mnemonic	Characteristics										Code	Page
LOAD NEGATIVE	LNR	RR	C							R		11	132
LOAD POSITIVE	LPR	RR	C				IF			R		10	132
LOAD PSW	LPSW	S	L	M	A	SP		$				82	105
MONITOR CALL	MC	SI				SP		MO				AF	132
MOVE (character)	MVC	SS			A						ST	D2	133
MOVE (immediate)	MVI	SI			A						ST	92	133
MOVE LONG	MVCL	RR	C		A	SP				R	ST	0E	133
MOVE NUMERICS	MVN	SS			A						ST	D1	135
MOVE WITH OFFSET	MVO	SS			A						ST	F1	135
MOVE ZONES	MVZ	SS			A						ST	D3	136
MULTIPLY	MR	RR				SP				R		1C	136
MULTIPLY	M	RX			A	SP				R		5C	136
MULTIPLY HALFWORD	MH	RX			A					R		4C	136
OR	OR	RR	C							R		16	137
OR	O	RX	C		A					R		56	137
OR (character)	OC	SS	C		A						ST	D6	137
OR (immediate)	OI	SI	C		A						ST	96	137
PACK	PACK	SS			A						ST	F2	137
SET CLOCK	SCK	S	C	M	A	SP						B204	108
SET PROGRAM MASK	SPM	RR	L									04	138
SET STORAGE KEY	SSK	RR		M	A_1	SP						08	110
SET SYSTEM MASK	SSM	S		M	A	SP			SO			80	110
SHIFT LEFT DOUBLE	SLDA	RS	C			SP	IF			R		8F	138
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS				SP				R		8D	139
SHIFT LEFT SINGLE	SLA	RS	C				IF			R		8B	139
SHIFT LEFT SINGLE LOGICAL	SLL	RS								R		89	139
SHIFT RIGHT DOUBLE	SRDA	RS	C			SP				R		8E	140
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS				SP				R		8C	140
SHIFT RIGHT SINGLE	SRA	RS	C							R		8A	140
SHIFT RIGHT SINGLE LOGICAL	SRL	RS								R		88	141
START I/O	SIO	S	C	M				$				9C00*	204
START I/O FAST RELEASE	SIOF	S	C	M				$				9C01*	204
STORE	ST	RX			A						ST	50	141
STORE CHANNEL ID	STIDC	S	C	M				$				B203	206
STORE CHARACTER	STC	RX			A						ST	42	141
STORE CHARACTERS UNDER MASK	STCM	RS			A						ST	BE	141
STORE CLOCK	STCK	S	C		A			$			ST	B205	141
STORE CONTROL	STCTL	RS		M	A	SP					ST	B6	111
STORE CPU ID	STIDP	S		M	A	SP					ST	B202	112
STORE HALFWORD	STH	RX			A						ST	40	142
STORE MULTIPLE	STM	RS			A						ST	90	142
SUBTRACT	SR	RR	C				IF			R		1B	143
SUBTRACT	S	RX	C		A		IF			R		5B	143
SUBTRACT HALFWORD	SH	RX	C		A		IF			R		4B	143
SUBTRACT LOGICAL	SLR	RR	C							R		1F	143
SUBTRACT LOGICAL	SL	RX	C		A					R		5F	143
SUPERVISOR CALL	SVC	RR										0A	144
TEST AND SET	TS	S	C		A						ST	93	144
TEST CHANNEL	TCH	S	C	M								9F00≠	207
TEST I/O	TIO	S	C	M								9D00*	208
TEST UNDER MASK	TM	SI	C		A							91	145
TRANSLATE	TR	SS			A						ST	DC	145
TRANSLATE AND TEST	TRT	SS	C		A					R		DD	145
UNPACK	UNPK	SS			A						ST	F3	146

Instructions Arranged by Feature

Decimal-Feature Instructions

Name	Mnemonic	Characteristics										Code	Page
ADD DECIMAL	AP	SS	C	PD	A		D	DF			ST	FA	149
COMPARE DECIMAL	CP	SS	C	PD	A		D					F9	149
DIVIDE DECIMAL	DP	SS		PD	A	SP	D		DK		ST	FD	149
EDIT	ED	SS	C	PD	A		D				ST	DE	150
EDIT AND MARK	EDMK	SS	C	PD	A		D			R	ST	DF	152
MULTIPLY DECIMAL	MP	SS		PD	A	SP	D				ST	FC	153
SHIFT AND ROUND DECIMAL	SRP	SS	C	PD	A		D	DF			ST	F0	153
SUBTRACT DECIMAL	SP	SS	C	PD	A		D	DF			ST	FB	154
ZERO AND ADD	ZAP	SS	C	PD	A		D	DF			ST	F8	155

Instructions Arranged by Feature

Floating-Point Feature Instructions

Name	Mnemonic	Characteristics										Code	Page
ADD NORMALIZED (long)	ADR	RR	C	FP		SP	U	E		LS		2A	160
ADD NORMALIZED (long)	AD	RX	C	FP	A	SP	U	E		LS		6A	160
ADD NORMALIZED (short)	AER	RR	C	FP		SP	U	E		LS		3A	160
ADD NORMALIZED (short)	AE	RX	C	FP	A	SP	U	E		LS		7A	160
ADD UNNORMALIZED (long)	AWR	RR	C	FP		SP		E		LS		2E	162
ADD UNNORMALIZED (long)	AW	RX	C	FP	A	SP		E		LS		6E	162
ADD UNNORMALIZED (short)	AUR	RR	C	FP		SP		E		LS		3E	162
ADD UNNORMALIZED (short)	AU	RX	C	FP	A	SP		E		LS		7E	162
COMPARE (long)	CDR	RR	C	FP		SP						29	163
COMPARE (long)	CD	RX	C	FP	A	SP						69	163
COMPARE (short)	CER	RR	C	FP		SP						39	163
COMPARE (short)	CE	RX	C	FP	A	SP						79	163
DIVIDE (long)	DDR	RR		FP		SP	U	E	FK			2D	163
DIVIDE (long)	DD	RX		FP	A	SP	U	E	FK			6D	163
DIVIDE (short)	DER	RR		FP		SP	U	E	FK			3D	163
DIVIDE (short)	DE	RX		FP	A	SP	U	E	FK			7D	163
HALVE (long)	HDR	RR		FP		SP	U					24	164
HALVE (short)	HER	RR		FP		SP	U					34	164
LOAD (long)	LDR	RR		FP		SP						28	165
LOAD (long)	LD	RX		FP	A	SP						68	165
LOAD (short)	LER	RR		FP		SP						38	165
LOAD (short)	LE	RX		FP	A	SP						78	165
LOAD AND TEST (long)	LTDR	RR	C	FP		SP						22	165
LOAD AND TEST (short)	LTER	RR	C	FP		SP						32	165
LOAD COMPLEMENT (long)	LCDR	RR	C	FP		SP						23	166
LOAD COMPLEMENT (short)	LCER	RR	C	FP		SP						33	165
LOAD NEGATIVE (long)	LNDR	RR	C	FP		SP						21	166
LOAD NEGATIVE (short)	LNER	RR	C	FP		SP						31	166
LOAD POSITIVE (long)	LPDR	RR	C	FP		SP						20	166
LOAD POSITIVE (short)	LPER	RR	C	FP		SP						30	166
MULTIPLY (long)	MDR	RR		FP		SP	U	E				2C	167
MULTIPLY (long)	MD	RX		FP	A	SP	U	E				6C	167
MULTIPLY (short to long)	MER	RR		FP		SP	U	E				3C	167
MULTIPLY (short to long)	ME	RX		FP	A	SP	U	E				7C	167
STORE (long)	STD	RX		FP	A	SP					ST	60	169
STORE (short)	STE	RX		FP	A	SP					ST	70	168
SUBTRACT NORMALIZED (long)	SDR	RR	C	FP		SP	U	E		LS		2B	169
SUBTRACT NORMALIZED (long)	SD	RX	C	FP	A	SP	U	E		LS		6B	169
SUBTRACT NORMALIZED (short)	SER	RR	C	FP		SP	U	E		LS		3B	169
SUBTRACT NORMALIZED (short)	SE	RX	C	FP	A	SP	U	E		LS		7B	169
SUBTRACT UNNORMALIZED (long)	SWR	RR	C	FP		SP		E		LS		2F	170
SUBTRACT UNNORMALIZED (long)	SW	RX	C	FP	A	SP		E		LS		6F	170
SUBTRACT UNNORMALIZED (short)	SUR	RR	C	FP		SP		E		LS		3F	169
SUBTRACT UNNORMALIZED (short)	SU	RX	C	FP	A	SP		E		LS		7F	169

Instructions Arranged by Feature

CPU Timer and Clock Comparator Feature Instructions

Name	Mnemonic	Characteristics											Code	Page
SET CLOCK COMPARATOR	SCKC	S		CK	M	A	SP						B206	108
SET CPU TIMER	SPT	S		CK	M	A	SP						B208	109
STORE CLOCK COMPARATOR	STCKC	S		CK	M	A	SP					ST	B207	111
STORE CPU TIMER	STPT	S		CK	M	A	SP					ST	B209	113

Direct-Control Feature Instructions

Name	Mnemonic	Characteristics											Code	Page
READ DIRECT	RDD	SI		DC	M	A				$		ST	85	107
WRITE DIRECT	WRD	SI		DC	M	A				$			84	114

Extended-Precision Floating-Point Feature Instructions

Name	Mnemonic	Characteristics											Code	Page
ADD NORMALIZED (extended)	AXR	RR	C	XP			SP	U	E	LS			36	160
LOAD ROUNDED (extended to long)	LRDR	RR		XP			SP		E				25	167
LOAD ROUNDED (long to short)	LRER	RR		XP			SP		E				35	166
MULTIPLY (extended)	MXR	RR		XP			SP	U	E				26	167
MULTIPLY (long to extended)	MXDR	RR		XP			SP	U	E				27	167
MULTIPLY (long to extended)	MXD	RX		XP		A	SP	U	E				67	167
SUBTRACT NORMALIZED (extended)	SXR	RR	C	XP			SP	U	E	LS			37	169

Translation-Feature Instructions

Name	Mnemonic	Characteristics											Code	Page
LOAD REAL ADDRESS	LRA	RX	C	TR	M	A_2					R		B1	106
PURGE TLB	PTLB	S		TR	M					$			B20D	107
RESET REFERENCE BIT	RRB	S	C	TR	M	A_1							B213	107
STORE THEN AND SYSTEM MASK	STNSM	SI		TR	M	A						ST	AC	113
STORE THEN OR SYSTEM MASK	STOSM	SI		TR	M	A	SP					ST	AD	114

Multiprocessing-Feature Instructions

Name	Mnemonic	Characteristics											Code	Page
SET PREFIX	SPX	S		MP	M	A	SP						B210	109
SIGNAL PROCESSOR	SIGP	RS	C	MP	M						R		AE	110
STORE CPU ADDRESS	STAP	S		MP	M	A	SP					ST	B212	112
STORE PREFIX	STPX	S		MP	M	A	SP					ST	B211	113

Conditional-Swapping Feature Instructions

Name	Mnemonic	Characteristics											Code	Page
COMPARE AND SWAP	CS	RS	C	SW		A	SP				R	ST	BA	123
COMPARE DOUBLE AND SWAP	CDS	RS	C	SW		A	SP				R	ST	BB	124

PSW Key-Handling Feature Instructions

Name	Mnemonic	Characteristics											Code	Page
INSERT PSW KEY	IPK	S		PK	M						R		B20B	105
SET PSW KEY FROM ADDRESS	SPKA	S		PK	M								B20A	109

Program Status Word

Channel Masks 0-5	I O	E	Key	0	M	W	P	Interruption Code	
0	6		8	12				16	31

ILC	CC	Program Mask	Instruction Address	
32	34	36	40	63

PSW Format in BC Mode

0	R	0	0	0	T	I O	E	Key	1	M	W	P	0 0	CC	Program Mask	0 0 0 0 0 0 0 0	
0								8	12				16	18	20	24	31

0 0 0 0 0 0 0 0	Instruction Address	
32	40	63

PSW Format in EC Mode

Assignment of Control Register Fields

Word	Bits	Name of Field	Associated With	Initial Value
0	0	Block-Multiplexing Control	Block-Multiplexing	0
0	1	SSM-Suppression Control	SSM Suppression	0
0	2	TOD Clock Sync Control	Multiprocessing	0
0	8-9	Page-Size Control	Dynamic Addr. Translation	0
0	10	Unassigned, must be zero	Dynamic Addr. Translation	0
0	11-12	Segment-Size Control	Dynamic Addr. Translation	0
0	16	Malfunction-Alert Mask	Multiprocessing	0
0	17	Emergency-Signal Mask	Multiprocessing	0
0	18	External-Call Mask	Multiprocessing	0
0	19	TOD-Clock-Sync-Check Mask	Multiprocessing	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
1	0-7	Segment-Table Length	Dynamic Addr. Translation	0
1	8-25	Segment-Table Address	Dynamic Addr. Translation	0
2	0-31	Channel Masks	Channels	1
8	16-31	Monitor Masks	Monitoring	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[1] 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	Check-Stop Control	Machine-Check Handling	1
14	1	Synchronous-MCEL[2] Control	Machine-Check Handling	1
14	2	I/O-Extended-Logout Control	I/O Extended Logout	0
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
14	8	Asynchronous-MCEL Control	Machine-Check Handling	0
14	9	Asynchronous-Fixed-Log Control	Machine-Check Handling	0
15	8-28	MCEL Address	Machine-Check Handling	512[3]

Explanation:

The fields not listed are unassigned.

Except for bit 10 of control register 0, the initial value of unassigned register positions is unpredictable.

[1] PER means program-event recording.

[2] MCEL means machine-check extended logout.

[3] Bit 22 is set to one, with all other bits set to zero, thus yielding a decimal byte address of 512.

Assigned Locations in Real Main Storage

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	Processor Address	External-Interruption Code		
88	136	0000000000000	ILC	0	Superv.-Call-Irptn. Code
8C	140	0000000000000	ILC	0	Program-Interruption Code
90	144	00000000	Translation-Exception Address		
94	148	00000000	Monitor Cl.#	PER C.	000000000000
98	152	00000000	PER Address		
9C	156	00000000	Monitor Code		
A0	160				
A4	164				
A8	168	Channel ID			
AC	172		IOEL Address		
B0	176	Limited Channel Logout			
B4	180				
B8	184		00000000	I/O Address	

Hex	Dec		
BC	188		
C0	192		
C4	196		
C8	200		
CC	204		
D0	208		
D4	212		
D8	216	Machine-Check CPU-Timer Save Area	
DC	220		
E0	224	Machine-Check 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	Region Code	
100	256	Fixed Logout Area	
104	260		
108	264		
10C	268		
154	340		
158	344		
15C	348		
160	352	Machine-Check Floating-Point Register Save Area	
164	356		
168	360		
16C	364		
170	368		
174	372		
178	376		
17C	380		
180	384	Machine-Check General-Register Save Area	
184	388		
188	392		
18C	396		
1B4	436		
1B8	440		
1BC	444		
1C0	448	Machine-Check Control-Register Save Area	
1C4	452		
1C8	456		
1CC	460		
1F4	500		
1F8	504		
1FC	508		

Assigned Locations in Absolute Main Storage

Hex	Dec	
0	0	Initial Program Loading PSW
4	4	
8	8	Initial Program Loading CCW1
C	12	
10	16	Initial Program Loading CCW2
14	20	
18	24	
1C	28	
20	32	
24	36	
28	40	
2C	44	
30	48	
34	52	
38	56	
3C	60	
40	64	
44	68	
48	72	
4C	76	
50	80	
54	84	
58	88	
5C	92	
60	96	
64	100	
68	104	
6C	108	
70	112	
74	116	
78	120	
7C	124	
80	128	
84	132	
88	136	
8C	140	
90	144	
94	148	
98	152	
9C	156	
A0	160	
A4	164	
A8	168	
AC	172	
B0	176	
B4	180	
B8	184	
BC	188	

Hex	Dec	
C0	192	
C4	196	
C8	200	
CC	204	
D0	208	
D4	212	
D8	216	Store-Status CPU Timer Save Area
DC	220	
E0	224	Store-Status Clock-Comparator Save Area
E4	228	
E8	232	
EC	236	
F0	240	
F4	244	
F8	248	
FC	252	
100	256	Store-Status PSW Save Area
104	260	
108	264	Store-Status Prefix Save Area
10C	268	Store-Status Model-Dependent Feature Area
110	272	
		≈
158	344	
15C	348	
160	352	Store-Status Floating-Point Register Save Area
164	356	
168	360	
16C	364	
170	368	
174	372	
178	376	
17C	380	
180	384	Store-Status General-Register Save Area
184	388	
188	392	
18C	396	
		≈
1B4	436	
1B8	440	
1BC	444	
1C0	448	Store-Status Control-Register Save Area
1C4	452	
1C8	456	
1CC	460	
		≈
1F4	500	
1F8	504	
1FC	508	

Appendix E. Condition-Code Settings

	Condition Code			
Instruction	0	1	2	3
General Instructions				
ADD (and ADD HALFWORD)	zero	< zero	> zero	overflow
ADD LOGICAL	zero, no carry	not zero, no carry	zero, carry	not zero, carry
AND	zero	not zero	–	–
COMPARE (and COMPARE HALFWORD)	equal	low	high	–
COMPARE AND SWAP	equal	not equal	–	–
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	–
EXCLUSIVE OR	zero	not zero	–	–
INSERT CHARACTERS UNDER MASK	zero	1st bit one	1st bit zero	–
LOAD AND TEST	zero	< zero	> zero	–
LOAD COMPLEMENT	zero	< zero	> zero	overflow
LOAD NEGATIVE	zero	< zero	–	–
LOAD POSITIVE	zero	–	> zero	overflow
MOVE LONG	count equal	count low	count high	destr. overlap
OR	zero	not zero	–	–
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	–
STORE CLOCK	set	not set	error	not operational
SUBTRACT (and SUBTRACT HALFWORD)	zero	< zero	> zero	overflow
SUBTRACT LOGICAL	–	not zero, no carry	zero, carry	not zero, carry
TEST AND SET	zero	one	–	–
TEST UNDER MASK	zero	mixed	–	ones
TRANSLATE AND TEST	zero	incomplete	complete	–
Decimal Instructions				
ADD DECIMAL	zero	< zero	> zero	overflow
COMPARE DECIMAL	equal	low	high	–
EDIT	zero	< zero	> zero	–
EDIT AND MARK	zero	< zero	> zero	–
SHIFT AND ROUND DECIMAL	zero	< zero	> zero	overflow
SUBTRACT DECIMAL	zero	< zero	> zero	overflow
ZERO AND ADD	zero	< zero	> zero	overflow
Floating-Point Instructions				
ADD NORMALIZED	zero	< zero	> zero	–
ADD UNNORMALIZED	zero	< zero	> zero	–
COMPARE	equal	low	high	–
LOAD AND TEST	zero	< zero	> zero	–
LOAD COMPLEMENT	zero	< zero	> zero	–
LOAD NEGATIVE	zero	< zero	–	–
LOAD POSITIVE	zero	–	> zero	–
SUBTRACT NORMALIZED	zero	< zero	> zero	–
SUBTRACT UNNORMALIZED	zero	< zero	> zero	–

Condition-Code Settings (Part 1 of 2)

	Condition Code			
Instruction	0	1	2	3
Input/Output Instructions				
CLEAR I/O	no operation in progress	CSW stored	channel busy	not operational
HALT DEVICE	interruption pending, or busy	CSW stored	channel working	not operational
HALT I/O	interruption pending	CSW stored	burst op. stopped	not operational
START I/O	successful	CSW stored	busy	not operational
START I/O FAST RELEASE	successful	CSW stored	busy	not operational
STORE CHANNEL ID	ID stored	CSW stored	busy	not operational
TEST CHANNEL	available	interruption pending	burst mode	not operational
TEST I/O	available	CSW stored	busy	not operational
System Control Instructions				
LOAD REAL ADDRESS	translation available	ST entry invalid	PT entry invalid	length violation
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
SET CLOCK	set	secure	–	not operational
SIGNAL PROCESSOR	order code accepted	status stored	busy	not operational

Explanation:

> zero	Result is greater than zero.
high	First operand compares high.
< zero	Result is less than zero.
low	First operand compares low.

The condition code may also be changed by LOAD PSW, SET PROGRAM MASK, and DIAGNOSE, and by an interruption.

Condition-Code Settings (Part 2 of 2)

Appendix F. 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	0.01562 5
128	7	0.00781 25
256	8	0.00390 625
512	9	0.00195 3125
1,024	10	0.00097 65625
2,048	11	0.00048 82812 5
4,096	12	0.00024 41406 25
8,192	13	0.00012 20703 125
16,384	14	0.00006 10351 5625
32,768	15	0.00003 05175 78125
65,536	16	0.00001 52587 89062 5
131,072	17	0.00000 76293 94531 25
262,144	18	0.00000 38146 97265 625
524,288	19	0.00000 19073 48632 8125
1,048,576	20	0.00000 09536 74316 40625
2,097,152	21	0.00000 04768 37158 20312 5
4,194,304	22	0.00000 02384 18579 10156 25
8,388,608	23	0.00000 01192 09289 55078 125
16,777,216	24	0.00000 00596 04644 77539 0625
33,554,432	25	0.00000 00298 02322 38769 53125
67,108,864	26	0.00000 00149 01161 19384 76562 5
134,217,728	27	0.00000 00074 50580 59692 38281 25
268,435,456	28	0.00000 00037 25290 29846 19140 625
536,870,912	29	0.00000 00018 62645 14923 09570 3125
1,073,741,824	30	0.00000 00009 31322 57461 54785 15625
2,147,483,648	31	0.00000 00004 65661 28730 77392 57812 5
4,294,967,296	32	0.00000 00002 32830 64365 38696 28906 25
8,589,934,592	33	0.00000 00001 16415 32182 69348 14453 125
17,179,869,184	34	0.00000 00000 58207 66091 34674 07226 5625
34,359,738,368	35	0.00000 00000 29103 83045 67337 03613 28125
68,719,476,736	36	0.00000 00000 14551 91522 83668 51806 64062 5
137,438,953,472	37	0.00000 00000 07275 95761 41834 25903 32031 25
274,877,906,944	38	0.00000 00000 03637 97880 70917 12951 66015 625
549,755,813,888	39	0.00000 00000 01818 98940 35458 56475 83007 8125
1,099,511,627,776	40	0.00000 00000 00909 49470 17729 28237 91503 90625
2,199,023,255,552	41	0.00000 00000 00454 74735 08864 64118 95751 95312 5
4,398,046,511,104	42	0.00000 00000 00227 37367 54432 32059 47875 97656 25
8,796,093,022,208	43	0.00000 00000 00113 68683 77216 16029 73937 98828 125
17,592,186,044,416	44	0.00000 00000 00056 84341 88608 08014 86968 99414 0625
35,184,372,088,832	45	0.00000 00000 00028 42170 94304 04007 43484 49707 03125
70,368,744,177,664	46	0.00000 00000 00014 21085 47152 02003 71742 24853 51562 5
140,737,488,355,328	47	0.00000 00000 00007 10542 73576 01001 85871 12426 75781 25
281,474,976,710,656	48	0.00000 00000 00003 55271 36788 00500 92935 56213 37890 625
562,949,953,421,312	49	0.00000 00000 00001 77635 68394 00250 46467 78106 68945 3125
1,125,899,906,842,624	50	0.00000 00000 00000 88817 84197 00125 23233 89053 34472 65625
2,251,799,813,685,248	51	0.00000 00000 00000 44408 92098 50062 61616 94526 67236 32812 5
4,503,599,627,370,496	52	0.00000 00000 00000 22204 46049 25031 30808 47263 33618 16406 25
9,007,199,254,740,992	53	0.00000 00000 00000 11102 23024 62515 65404 23631 66809 08203 125
18,014,398,509,481,984	54	0.00000 00000 00000 05551 11512 31257 82702 11815 83404 54101 5625
36,028,797,018,963,968	55	0.00000 00000 00000 02775 55756 15628 91351 05907 91702 27050 78125
72,057,594,037,927,936	56	0.00000 00000 00000 01387 77878 07814 45675 52953 95851 13525 39062 5
144,115,188,075,855,872	57	0.00000 00000 00000 00693 88939 03907 22837 76476 97925 56762 69531 25
288,230,376,151,711,744	58	0.00000 00000 00000 00346 94469 51953 61418 88238 48962 78381 34765 625
576,460,752,303,423,488	59	0.00000 00000 00000 00173 47234 75976 80709 44119 24481 39190 67382 8125
1,152,921,504,606,846,976	60	0.00000 00000 00000 00086 73617 37988 40354 72059 62240 69595 33691 40625
2,305,843,009,213,693,952	61	0.00000 00000 00000 00043 36808 68994 20177 36029 81120 34797 66845 70312 5
4,611,686,018,427,387,904	62	0.00000 00000 00000 00021 68404 34497 10088 68014 90560 17398 83422 85156 25
9,223,372,036,854,775,808	63	0.00000 00000 00000 00010 84202 17248 55044 34007 45280 08699 41711 42578 125
18,446,744,073,709,551,616	64	0.00000 00000 00000 00005 42101 08624 27522 17003 72640 04349 70855 71289 0625

Powers of 2 (Part 1 of 2)

18,446,744,073,709,551,616	64
36,893,488,147,419,103,232	65
73,786,976,294,838,206,464	66
147,573,952,589,676,412,928	67
295,147,905,179,352,825,856	68
590,295,810,358,705,651,712	69
1,180,591,620,717,411,303,424	70
2,361,183,241,434,822,606,848	71
4,722,366,482,869,645,213,696	72
9,444,732,965,739,290,427,392	73
18,889,465,931,478,580,854,784	74
37,778,931,862,957,161,709,568	75
75,557,863,725,914,323,419,136	76
151,115,727,451,828,646,838,272	77
302,231,454,903,657,293,676,544	78
604,462,909,807,314,587,353,088	79
1,208,925,819,614,629,174,706,176	80
2,417,851,639,229,258,349,412,352	81
4,835,703,278,458,516,698,824,704	82
9,671,406,556,917,033,397,649,408	83
19,342,813,113,834,066,795,298,816	84
38,685,626,227,668,133,590,597,632	85
77,371,252,455,336,267,181,195,264	86
154,742,504,910,672,534,362,390,528	87
309,485,009,821,345,068,724,781,056	88
618,970,019,642,690,137,449,562,112	89
1,237,940,039,285,380,274,899,124,224	90
2,475,880,078,570,760,549,798,248,448	91
4,951,760,157,141,521,099,596,496,896	92
9,903,520,314,283,042,199,192,993,792	93
19,807,040,628,566,084,398,385,987,584	94
39,614,081,257,132,168,796,771,975,168	95
79,228,162,514,264,337,593,543,950,336	96
158,456,325,028,528,675,187,087,900,672	97
316,912,650,057,057,350,374,175,801,344	98
633,825,300,114,114,700,748,351,602,688	99
1,267,650,600,228,229,401,496,703,205,376	100
2,535,301,200,456,458,802,993,406,410,752	101
5,070,602,400,912,917,605,986,812,821,504	102
10,141,204,801,825,835,211,973,625,643,008	103
20,282,409,603,651,670,423,947,251,286,016	104
40,564,819,207,303,340,847,894,502,572,032	105
81,129,638,414,606,681,695,789,005,144,064	106
162,259,276,829,213,363,391,578,010,288,128	107
324,518,553,658,426,726,783,156,020,576,256	108
649,037,107,316,853,453,566,312,041,152,512	109
1,298,074,214,633,706,907,132,624,082,305,024	110
2,596,148,429,267,413,814,265,248,164,610,048	111
5,192,296,858,534,827,628,530,496,329,220,096	112
10,384,593,717,069,655,257,060,992,658,440,192	113
20,769,187,434,139,310,514,121,985,316,880,384	114
41,538,374,868,278,621,028,243,970,633,760,768	115
83,076,749,736,557,242,056,487,941,267,521,536	116
166,153,499,473,114,484,112,975,882,535,043,072	117
332,306,998,946,228,968,225,951,765,070,086,144	118
664,613,997,892,457,936,451,903,530,140,172,288	119
1,329,227,995,784,915,872,903,807,060,280,344,576	120
2,658,455,991,569,831,745,807,614,120,560,689,152	121
5,316,911,983,139,663,491,615,228,241,121,378,304	122
10,633,823,966,279,326,983,230,456,482,242,756,608	123
21,267,647,932,558,653,966,460,912,964,485,513,216	124
42,535,295,865,117,307,932,921,825,928,971,026,432	125
85,070,591,730,234,615,865,843,651,857,942,052,864	126
170,141,183,460,469,231,731,687,303,715,884,105,728	127
340,282,366,920,938,463,463,374,607,431,768,211,456	128

Powers of 2 (Part 2 of 2)

Appendix G. 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 Integers

HALFWORD								HALFWORD							
BYTE				BYTE				BYTE				BYTE			
BITS: 0123		4567		0123		4567		0123		4567		0123		4567	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
8		7		6		5		4		3		2		1	

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position.
4. Add the numbers selected from the table to form the decimal number.

EXAMPLE	
Conversion of Hexadecimal Value	D34
1. D	3328
2. 3	48
3. 4	4
4. Decimal	3380

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.
   (b) Record the hexadecimal of the column containing the selected number.
   (c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1(c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder).
3. Using the remainder from step 2 repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine terms to form the hexadecimal number.

EXAMPLE	
Conversion of Decimal Value	3380
1. D	-3328
	52
2. 3	-48
	4
3. 4	-4
4. Hexadecimal	D34

To convert integer numbers greater than the capacity of table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D34_{16} = 3380_{10}$

$$\begin{array}{rr} D = & 13 \\ & \underline{\times 16} \\ & 208 \\ 3 = & \underline{+\ 3} \\ & 211 \\ & \underline{\times 16} \\ & 3376 \\ 4 = & \underline{+4} \\ & 3380 \end{array}$$

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

$$\begin{array}{lll} 16 \,\underline{|\,3380} & \searrow & \text{remainder} \\ 16 \,\underline{|\,211} & \searrow & 4 \;\uparrow \\ 16 \,\underline{|\,13} & \searrow & 3 \\ & & D \end{array}$$

$3380_{10} = D34_{16}$

POWERS OF 16 TABLE

Example: $268{,}435{,}456_{10} = (2.68435456 \times 10^8)_{10} = 1000\ 0000_{16} = (10^7)_{16}$

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10 = A
17 592 186 044 416	11 = B
281 474 976 710 656	12 = C
4 503 599 627 370 496	13 = D
72 057 594 037 927 936	14 = E
1 152 921 504 606 846 976	15 = F

Decimal Values

Conversion Table: Hexadecimal and Decimal Fractions

HALFWORD													
BYTE					BYTE								
BITS 0123		4567			0123				4567				
Hex	Decimal	Hex	Decimal		Hex	Decimal			Hex	Decimal Equivalent			
.0	.0000	.00	.0000	0000	.000	.0000	0000	0000	.0000	.0000	0000	0000	0000
.1	.0625	.01	.0039	0625	.001	.0002	4414	0625	.0001	.0000	1525	8789	0625
.2	.1250	.02	.0078	1250	.002	.0004	8828	1250	.0002	.0000	3051	7578	1250
.3	.1875	.03	.0117	1875	.003	.0007	3242	1875	.0003	.0000	4577	6367	1875
.4	.2500	.04	.0156	2500	.004	.0009	7656	2500	.0004	.0000	6103	5156	2500
.5	.3125	.05	.0195	3125	.005	.0012	2070	3125	.0005	.0000	7629	3945	3125
.6	.3750	.06	.0234	3750	.006	.0014	6484	3750	.0006	.0000	9155	2734	3750
.7	.4375	.07	.0273	4375	.007	.0017	0898	4375	.0007	.0001	0681	1523	4375
.8	.5000	.08	.0312	5000	.008	.0019	5312	5000	.0008	.0001	2207	0312	5000
.9	.5625	.09	.0351	5625	.009	.0021	9726	5625	.0009	.0001	3732	9101	5625
.A	.6250	.0A	.0390	6250	.00A	.0024	4140	6250	.000A	.0001	5258	7890	6250
.B	.6875	.0B	.0429	6875	.00B	.0026	8554	6875	.000B	.0001	6784	6679	6875
.C	.7500	.0C	.0468	7500	.00C	.0029	2968	7500	.000C	.0001	8310	5468	7500
.D	.8125	.0D	.0507	8125	.00D	.0031	7382	8125	.000D	.0001	9836	4257	8125
.E	.8750	.0E	.0546	8750	.00E	.0034	1796	8750	.000E	.0002	1362	3046	8750
.F	.9375	.0F	.0585	9375	.00F	.0036	6210	9375	.000F	.0002	2888	1835	9375
1		2			3				4				

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

Find .A in position 1	.6250
Find .0B in position 2	.0429 6875
Find .00C in position 3	.0029 2968 7500
.ABC Hex is equal to	.6708 9843 7500

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	
	-.0039 0625	= .01
3. Find .0009 7656 2500	.0010 9375 0000	
	-.0009 7656 2500	= .004
4. Find .0001 0681 1523 4375	.0001 1718 7500 0000	
	-.0001 0681 1523 4375	= .0007
	.0000 1037 5976 5625	= .2147 Hex

5. .13 Decimal is approximately equal to ⟶ .2147 Hex

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: $.8A7 = .540771_{10}$

$$8A7_{16} = 2215_{10} \qquad 16^3 = 4096 \qquad \begin{array}{r} .540771 \\ 4096\overline{)2215.000000} \end{array}$$

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

$$\begin{array}{lr} & .5408 \\ & \underline{\text{x}16} \\ 8 \longleftarrow & \boxed{8}.6528 \\ & \underline{\text{x}16} \\ A \longleftarrow & \boxed{10}.4448 \\ & \underline{\text{x}16} \\ \downarrow 7 \longleftarrow & \boxed{7}.1168 \end{array}$$

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 x 4 = 08, F x 2 = 1E

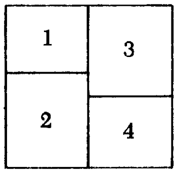
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Appendix H. EBCDIC Chart

Extended Binary-Coded-Decimal Interchange Code (EBCDIC)

The 256-position EBCDIC table, outlined by the heavy black lines, shows the graphic characters and control character representations for EBCDIC. The bit-position numbers, bit patterns, hexadecimal representations and card hole patterns for these and other possible EBCDIC characters are also shown.

To find the card hole patterns for most characters, partition the 256-position table into four blocks as follows:



Block 1: Zone punches at top of table; digit punches at left
Block 2: Zone punches at bottom of table; digit punches at left
Block 3: Zone punches at top of table; digit punches at right
Block 4: Zone punches at bottom of table; digit punches at right

Fifteen positions in the table are exceptions to the above arrangement. These positions are indicated by small numbers in the upper right corners of their boxes in the table. The card hole patterns for these positions are given at the bottom of the table. Bit-position numbers, bit patterns, and hexadecimal representations for these positions are found in the usual manner.

Following are some examples of the use of the EBCDIC chart:

Character	Type	Bit Pattern	Hex	Hole Pattern	
				Zone Punches	Digit Punches
PF	Control Character	00 00 0100	04	12 - 9	- 4
%	Special Graphic	01 10 1100	6C	0	- 8 - 4
R	Upper Case	11 01 1001	D9	11	- 9
a	Lower Case	10 00 0001	81	12 - 0	- 1
	Control Character, function not yet assigned	00 11 0000	30	12 - 11 - 0 - 9	- 8 - 1

Bit Positions
01 23 4567

			00				01				10				11				Bit Positions 0,1
			00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	Bit Positions 2,3
			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	First Hexadecimal Digit
			12				12	12		12	12	12		12	12				Zone Punches
				11				11	11	11		11	11	11		11			
					0		0		0	0	0		0	0			0		
			9	9	9	9	9	9	9	9									
Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	Digit Punches																	Digit Punches
0000	0	8-1	① NUL	② DLE	DS ③	④	SP ⑤	& ⑥	- ⑦	⑧					{ ⑨	} ⑩	\ ⑪	0 ⑫	8-1
0001	1	1	SOH	DC1	SOS				/ ⑬		a	j	~		A	J	⑭	1	1
0010	2	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2	2
0011	3	3	ETX	TM							c	l	t		C	L	T	3	3
0100	4	4	PF	RES	BYP	PN					d	m	u		D	M	U	4	4
0101	5	5	HT	NL	LF	RS					e	n	v		E	N	V	5	5
0110	6	6	LC	BS	ETB	UC					f	o	w		F	O	W	6	6
0111	7	7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7	7
1000	8	8	GE	CAN							h	q	y		H	Q	Y	8	8
1001	9	8-1	RLF	EM						`	i	r	z		I	R	Z	9	9
1010	A	8-2	SMM	CC	SM		¢	!	¦ ⑮	:								|	8-2
1011	B	8-3	VT	CU1	CU2	CU3	.	$	,	#									8-3
1100	C	8-4	FF	IFS		DC4	<	*	%	@					⑀		⑁		8-4
1101	D	8-5	CR	IGS	ENQ	NAK	(	)	_	'									8-5
1110	E	8-6	SO	IRS	ACK		+	;	>	=					⑂				8-6
1111	F	8-7	SI	IUS	BEL	SUB	|	¬	?	"								EO	8-7
			12				12				12	12		12	12	12		12	Zone Punches
				11				11				11	11	11		11	11	11	
					0				0		0		0	0	0		0	0	
			9	9	9	9									9	9	9	9	

Card Hole Patterns

①	12-0-9-8-1	⑤	No Punches	⑨	12-0	⑬	0-1
②	12-11-9-8-1	⑥	12	⑩	11-0	⑭	11-0-9-1
③	11-0-9-8-1	⑦	11	⑪	0-8-2	⑮	12-11
④	12-11-0-9-8-1	⑧	12-11-0	⑫	0		

Control Character Representations

ACK	Acknowledge
BEL	Bell
BS	Backspace
BYP	Bypass
CAN	Cancel
CC	Cursor Control
CR	Carriage Return
CU1	Customer Use 1
CU2	Customer Use 2
CU3	Customer Use 3
DC1	Device Control 1
DC2	Device Control 2
DC4	Device Control 4
DEL	Delete
DLE	Data Link Escape
DS	Digit Select
EM	End of Medium
ENQ	Enquiry
EO	Eight Ones
EOT	End of Transmission
ESC	Escape
ETB	End of Transmission Block
ETX	End of Text
FF	Form Feed
FS	Field Separator
GE	Graphic Escape
HT	Horizontal Tab
IFS	Interchange File Separator
IGS	Interchange Group Separator
IL	Idle
IRS	Interchange Record Separator
IUS	Interchange Unit Separator
LC	Lower Case
LF	Line Feed
NAK	Negative Acknowledge
NL	New Line
NUL	Null
PF	Punch Off
PN	Punch On
RES	Restore
RLF	Reverse Line Feed
RS	Reader Stop
SI	Shift In
SM	Set Mode
SMM	Start of Manual Message
SO	Shift Out
SOH	Start of Heading
SOS	Start of Significance
SP	Space
STX	Start of Text
SUB	Substitute
SYN	Synchronous Idle
TM	Tape Mark
UC	Upper Case
VT	Vertical Tab

Special Graphic Characters

¢	Cent Sign
.	Period, Decimal Point
<	Less-than Sign
(	Left Parenthesis
+	Plus Sign
|	Logical OR
&	Ampersand
!	Exclamation Point
$	Dollar Sign
*	Asterisk
)	Right Parenthesis
;	Semicolon
¬	Logical NOT
-	Minus Sign, Hyphen
/	Slash
¦	Vertical Line
,	Comma
%	Percent
_	Underscore
>	Greater-than Sign
?	Question Mark
`	Grave Accent
:	Colon
#	Number Sign
@	At Sign
'	Prime, Apostrophe
=	Equal Sign
"	Quotation Mark
~	Tilde
{	Opening Brace
⑀	Hook
⑂	Fork
}	Closing Brace
\	Reverse Slant
⑁	Chair
|	Long Vertical Mark

Appendix I. Number Representation and Instruction-Use Examples

Number Representation

FIXED-POINT WITH TWO'S COMPLEMENT

A fixed-point number is a signed value, recorded as a binary integer. It is called fixed-point because the programmer determines the fixed positioning of the radix point.

Fixed-point operands may be recorded in halfword (16-bit) or word (32-bit) lengths. In both lengths, the first bit position (0) holds the sign of the number, with the remaining bit positions (1-15 for halfwords and 1-31 for fullwords) used to designate the magnitude of the number.

Positive fixed-point numbers are represented in true binary form with a zero sign bit. Negative fixed-point numbers are represented 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 fixed-point numbers are formed in two's-complement notation by inverting each bit of the positive binary number and adding one. For example, the true binary form of the decimal value (+26) is made negative (–26) in the following manner:

```
         S  Integer
+26      0 000 0000 0001 1010
Invert   1 111 1111 1110 0101
Add 1                       1
         ---------------------
–26      1 111 1111 1110 0110 (Two's-complement
                                form)
```

This is equivalent to subtracting the number

```
  0000 0000 0001 1010
  from
1 0000 0000 0000 0000
```

The following addition examples illustrate two's-complement arithmetic. Only eight bit positions are used. All negative numbers are in two's-complement form.

```
1.   +57 = 0011 1001
     +35 = 0010 0011
     ---------------
     +92 = 0101 1100

2.   +57 = 0011 1001
     –35 = 1101 1101    No overflow.
     ---------------
     +22 = 0001 0110    Ignore carry–carry into high-order
                        position and carry out.

3.   +35 = 0010 0011
     –57 = 1100 0111
     ---------------
     –22 = 1110 1010    Sign change only; no carry

4.   –57 = 1100 0111
     –35 = 1101 1101    No overflow.
     ---------------
     –92 = 1010 0100    Ignore carry–carry into high-order
                        position and carry out.

5.   –57 = 1100 0111
     –92 = 1010 0100
     ----------------
    –149 = *0110 1011   *Overflow–no carry into high-order
                         position but carry out.

6.   +57 = 0011 1001
     +92 = 0101 1100
     ----------------
     149 = *1001 0101   *Overflow–carry into high-order
                         position, no carry out.
```

The presence or absence of an overflow condition may be recognized by the condition of the carries.

- There is no overflow:
  a. If there is a carry into the high-order bit position and also a carry out (examples 2 and 4).
  b. If there is no carry into the high-order bit position and no carry out (examples 1 and 3).
- There is an overflow:
  a. If there is no carry into the high-order position but there is a carry out (example 5).
  b. If there is a carry into the high-order position but no carry out (example 6).

The following are 16-bit fixed-point numbers. The first is the largest 16-bit positive number and the last, the largest 16-bit negative number.

Number		*Decimal*		*S*	*Integer*
$2^{15} - 1$	=	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 32-bit fixed-point numbers. The first is the largest positive number that can be represented by 32 bits, and the last is the largest negative number 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

FLOATING POINT

Floating-point arithmetic simplifies the programming of computations in which the range of values used varies widely. It is called floating point because the radix-point placement, or scaling, is automatically maintained by the machine.

The key to floating-point data representation is the separation of the significant digits of a number from the size (scale) of the number. Thus, the number is expressed as a fraction times a power of 16.

A floating-point number has two associated sets of values. One set represents the significant digits of the number and is called the fraction. The second set specifies the power (exponent) to which 16 is raised and indicates the location of the binary point of the number.

The two numbers (the fraction and exponent) are recorded in a single word, a doubleword, or two doublewords.

Since each of these two numbers is signed, some method must be employed to express two signs in an area that provides for a single sign. This is accomplished by having the fraction sign use the sign associated with the word (or doubleword) and 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. 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 the range covered by the magnitude (M) of a floating-point number is:

$$16^{-65} \leqslant M < 16^{63}$$

or more precisely:

In the short format:

$$16^{-65} \leqslant M \leqslant (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \leqslant M \leqslant (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \leqslant M \leqslant (1 - 16^{-28}) \times 16^{63}$$

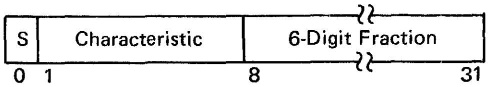
In decimal terms:

16^{-65} is approximately equal to 5.4×10^{-79}

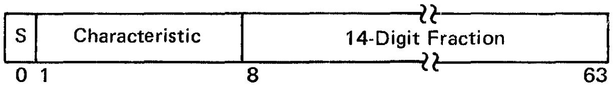
16^{63} is approximately equal to 7.2×10^{75}

Floating-point data in System/370 may be recorded in short, long, or extended formats. Each format uses a sign bit in bit position 0, followed by a characteristic in bit positions 1-7. Short floating-point operands contain the fraction in bit positions 8-31; long operands have the fraction in bit positions 8-63; and extended operands have the fraction in bit positions 8-63 and 72-127.

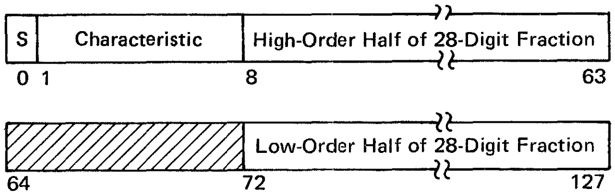
Short Floating-Point Number



Long Floating-Point Number



Extended Floating-Point Number


S
Characteristic
High-Order Half of 28-Digit Fraction
0
1
8
63
Low-Order Half of 28-Digit Fraction
64
72
127

The sign of the fraction is indicated by a zero or one bit in bit position 0 to denote a positive or negative fraction, respectively.

Within a given fraction length (6, 14, or 28 digits), a floating-point operation provides 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.

CONVERSION EXAMPLE

Convert the decimal number 149.25 to a short-precision floating-point operand. (Appendix G provides tables for the conversion of hexadecimal and decimal integers and fractions.)

1. The number is decomposed into decimal integer and a decimal fraction:

   149.25 = 149 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.

   base + exponent = characteristic
   64 + 2 = 66 = 1000010

6. The fraction is converted to binary and grouped hexadecimally.

   0.954_{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.

S	*Char*	*Fraction*
0	1000010	1001 0101 0100 0000 0000 0000

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	= $+1/16 \times 16^{1}$	= 0	100 0001	0001 0000 0000 0000 0000 0000
0.5	= $+8/16 \times 16^{0}$	= 0	100 0000	1000 0000 0000 0000 0000 0000
1/64	= $+4/16 \times 16^{-1}$	= 0	011 1111	0100 0000 0000 0000 0000 0000
0.0	= $+0 \times 16^{-64}$	= 0	000 0000	0000 0000 0000 0000 0000 0000
−15.0	= $-15/16 \times 16^{1}$	= 1	100 0001	1111 0000 0000 0000 0000 0000
5.4×10^{-79}	$\cong +1/16 \times 16^{-64}$	= 0	000 0000	0001 0000 0000 0000 0000 0000
7.2×10^{75}	$\cong (1 - 16^{-6}) \times 16^{63}$	= 0	111 1111	1111 1111 1111 1111 1111 1111

Instruction-Use Examples

The following examples illustrate the use of many System/370 instructions. Before studying one of these examples, the reader should first consult the instruction description in this manual for the particular instruction of interest to him.

Please note that this publication, and the instruction-use examples, are written principally for assembly-language programmers, to be used in conjunction with the appropriate assembly-language manuals.

For clarity, and for ease in programming, each example in this section presents the instruction both as it is written in an assembly-language statement and as it appears when assembled in storage (machine format).

Machine Format

As a rule, 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.

Assembly-Language Format

In assembly-language statements, registers, lengths, and masks are all presented in decimal, but displacements may be in hexadecimal or decimal. (A hexadecimal displacement is indicated by X'n', where n can range from 000-FFF.) Immediate operands are normally shown in hexadecimal. 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 System/370 assembly 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 assembly-language length. The assembly 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 assembly-language statements, a symbolic address is represented as a mnemonic term written in all capitals, such as FLAGS, which is used to denote the address of a storage location used to contain data or program-control information. When symbolic addresses are used, the assembler supplies actual base and displacement values according to the USING and DROP assembler instructions.

When symbolic addresses are used in the example, the values for base and displacement are not shown in the assembly-language format or in the machine-language format. For assembly-language formats, the letter S in the labels that designate instruction fields 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 (*).

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_{10}
Storage locations 2000-2001 contain FF FE = -2_{10}
Register 12 contains 00 00 18 00.
Register 13 contains 00 00 01 50.

The format of the required instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4A	5	D	C	6B0

Assembler Format

Op Code	$R_1, D_2 (X_2, B_2)$
AH	5, X'6B0'(13,12)

After the instruction is executed, register 5 contains 00 00 00 17 = 23_{10}.

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 in System/370, each pair of bits is handled separately; there is no connection from one bit position to another.

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 $0100\ 0011_2$. To set the rightmost bit of this byte to zero without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

Op Code	I_2	B_1	D_1
94	FE	8	001

Assembler Format

Op Code	D_1 (B_1), I_2
NI	1(8),X'FE'

When this instruction is executed, the byte in storage is ANDed with the immediate byte:

Location 4891	$0100\ 0011_2$
Immediate byte	$1111\ 1110_2$
Result:	$0100\ 0010_2$

The resulting byte, with bit 7 set to zero, is stored in location 4891. Condition code 2 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 a BAL instruction. (PSW bits 40-63 will contain 00 00 CA after execution of BAL)

The format of the BAL instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
45	2	0	5	010

Assembler Format

Op Code	R_1, D_2 (X_2, B_2)
BAL	2,X'10'(0,5)

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_{10}, provided that register 2 has not meanwhile been disturbed.

The BALR instruction with the R_2 field equal to zero may be used to load a register for use as a base register. For example, in the assembly language the sequence of statements:

```
BALR    15,0
USING   *,15
```

tells the assembly 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.)

At any time, the condition code may be preserved for future inspection with BALR 1,0. Bits 2 and 3 of the register (R_1) contain the condition code.

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 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:

Machine Format

Op Code	M_1	X_2	B_2	D_2
47	C	B	A	050

Assembler Format

Op Code	M_1, D_2 (X_2, B_2)
BC	12,X'50'(11,10)

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 assembly-language program:

```
        .
        .
        .
LUPE  AR  8,1
        .
        .
        .
BACK  BCT  6,LUPE
        .
        .
        .
```

where register 6 contains 00 00 00 03 and the address of LUPE is 6826. Assume that, in order to address this location, register 10 is used as a base register and contains 00 00 68 00.

The format of the BCT instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
46	6	0	A	026

Assembler Format

Op Code	R_1, D_2 (X_2, B_2)
BCT	6,X'26'(0,10)

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_{10} = the index
Register 6 contains 00 00 00 02 = 2_{10} = the increment
Register 7 contains 00 00 00 AA = 170_{10} = the comparand
Register 10 contains 00 00 71 30 = the branch address

The format of the instruction is:

Machine Format

Op Code	R_1	R_3	B_2	D_2
86	4	6	A	000

Assembler Format

Op Code	R_1, R_3, D_2 (B_2)
BXH	4, 6, 0(10)

When the instruction is executed: first, the contents of register 6 are added to register 4; second, the sum is compared with the contents of register 7; and third, the decision to branch is made. After execution:

Register 4 contains 00 00 00 8C = 140_{10}
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 assembly-language routine illustrates how this feaure may be used to search a table:

Table

Two Bytes	Two 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_{16} 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_{10}. 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 halfword in storage with the contents of a register. For example, assume that:

Register 44 contains FF FF 80 00 = $-32{,}768_{10}$
Register 13 contains 00 01 60 50
Storage locations 16080-16081 contain 8000 = $-32{,}768_{10}$

When the instruction

Machine Format

Op Code	R_1	X_2	B_2	D_2
49	4	0	D	030

Assembler Format

Op Code	$R_1, D_2 (X_2, B_2)$
CH	4,X'30'(0,13)

is executed, the contents of locations 16080-16081 are fetched, expanded to 32 bits (the sign bit is propagated to the left), and compared with the contents of register 4. Because the two numbers are equal, condition code 0 is set.

Compare Logical (CL, CLR, CLI, CLC)

The COMPARE LOGICAL instructions differ from the algebraic instructions (C, CR) in that all quantities are handled as if unsigned.

Compare Logical (CLR)

Assume that:

Register 1 contains 00 00 00 01
Register 2 contains FF FF FF FF

Execution of the instruction

Machine Format

Op Code	R_1	R_2
15	1	2

Assembler Format

Op Code	R_1, R_2
CLR	1,2

sets condition code 1. A condition code 1 indicates that the first operand is lower than the second. However, if an *algebraic* compare instruction had been executed, condition code 2 would have been set, indicating that the first operand is higher. During algebraic comparison, the contents of register 1 are interpreted as + 1 and the contents of register 2 as – 1. During logical comparison, the leftmost byte of register 2 is compared with the leftmost byte of register 1; each byte is interpreted as a binary number. In this case:

Leftmost byte of register 1: $0000\ 0000_2 = 0_{10}$
Leftmost byte of register 2: $1111\ 1111_2 = 255_{10}$

If the leftmost bytes are equal, the next two bytes are compared, etc., until either an inequality is discovered or the contents of the registers are exhausted.

Compare Logical Immediate (CLI)

The CLI instruction logically compares a byte from the instruction stream with a byte from storage. For example, assume that:

Register 10 contains 00 00 17 00
Storage location 1703 contains 7E

Execution of the instruction

Machine Format

Op Code	I_2	B_1	D_1
95	AF	A	003

Assembler Format

Op Code	$D_1 (B_1), I_2$
CLI	3(10),X'AF'

sets condition code 1, indicating that the first operand (the quantity in main storage) is lower than the second (immediate) operand.

Compare Logical Characters (CLC)

The COMPARE LOGICAL CHARACTERS instruction can be used to perform the logical 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 6 contains 00 00 18 80
Register 7 contains 00 00 19 00

Execution of the instruction

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D5	0B	6	006	7	000

Assembler Format

Op Code	$D_1 (L, B_1), D_2 (B_2)$
CLC	6(12,6),0(7)

sets condition code 2, indicating that the contents of field 1 are higher in value than the contents of field 2.

Because CLC compares bytes on an unsigned binary 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 selected bytes of a word contained in a general register to a contiguous field of bytes in main storage. The M_3 field of the CLM instruction is a four-bit mask that selects zero through four bytes from a general register, each mask bit corresponding left to right with a selected register byte. In the comparison, the selected register bytes are treated as a contiguous field, and the operands are considered as binary unsigned quantities, with all codes valid. The operation proceeds left to right. For example, assume that:

Three bytes starting at storage location 10200 contain F0 BC 7B
Register 12 contains 10 000
Register 6 contains F0 BC 5C 7B

Execution of the instruction

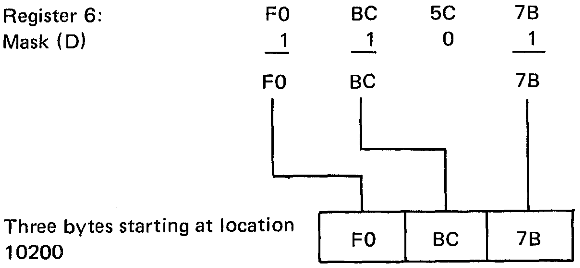
Machine Format

Op Code	R_1	M_3	B_2	D_2
BD	6	D	C	200

Assembler Format

Op Code	$R_1, M_3, D_2 (B_2)$
CLM	6,B'1101',X'200'(12)

produces the following result:



Result: Condition code 0 is set (selected bytes are equal, or mask is zero). Register 6 and bytes in main storage are unchanged.

Other condition codes would indicate:

Condition Code:

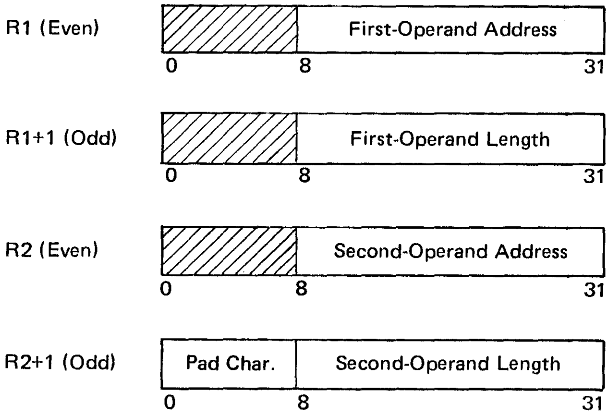
1 Selected field of first operand (register contents) is less than second operand (storage locations)

2 Selected field of first operand is greater than second operand

3 –

Compare Logical Long (CLCL)

The CLCL instruction is used to logically compare two operands in main storage. Each operand can be up to 16,777,215 bytes in length. Two pairs of even-odd general registers are used to locate the operands and to control the execution of the CLCL instruction, which can be interrupted in progress. The first register of each pair must be an even register, and it is used to contain 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 character which is used to logically extend a shorter operand to the same length as a longer operand. The following illustrates the assignment of registers for CLCL:



The following instructions set up two register pairs to control a text-string comparison. For example, assume:

Operand 1
Address: 20800 (hex)
Length: 100 (dec)

Pad Character
Address: 20003 (hex)
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 the start of the first operand
LA	5,100	Set register 5 to the length of first operand
LA	8,X'A00'(12)	Point register 8 to the start of second operand
LA	9,132	Set register 9 to the length of second operand
ICM	9,B'1000',3(12)	Insert padding character (blank) into byte 0 of register 9.

The register pair 4-5 is now covering 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.

The register pair 8-9 covers the second operand with bits 8-31 of register 8 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 logically pad the shorter operand. In this example, the blank padding character is used in the first operand, after the 100th character, to compare with the remaining characters in the second operand.

With the 4-5 and 8-9 register pairs thus set up, the format of the CLCL instruction is:

Machine Format

Op Code	R_1	R_2
0F	4	8

Assembler Format

Op Code	R_1, R_2
CLCL	4, 8

When this instruction is executed, the comparison starts at the high-order end of both operands and proceeds to the right. The operation ends as soon as an inequality is detected or the end of the longest operand is reached.

If this CLCL instruction is interrupted after 60 bytes are successfully compared, the operand lengths in registers 5 and 9 are decremented to X'28' and X'48', respectively, and the operand locations in registers 4 and 8 are incremented to X'2083C' and X'20A3C'. When the CLCL instruction is reexecuted, the comparison begins at the point of interruption.

If the instruction is interrupted after 110 bytes are successfully compared, the operand lengths in registers 5 and 9 are decremented to 0 and X'16', respectively, and the operand locations in registers 4 and 8 are incremented to X'2086E' and X'20A6E'.

When the comparison ends, the condition code indicates the result. The condition code settings are as follows:

Condition Code:

0	Operands are equal, or both field lengths are zero
1	First operand is low
2	First operand is high
3	–

When the operands are unequal, the address fields of registers 4 and 8 can be used to locate the bytes that caused the mismatch. The byte count fields in registers 5 and 9 can be used to determine how far the comparison progressed successfully.

Convert to Binary (CVB)

The CONVERT TO BINARY instruction converts an eight-byte, signed, packed-decimal number into a signed binary number 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 fixed-point arithmetic. For example, assume:

Storage locations 7608-760F contain a positive packed-decimal number, 00 00 00 00 00 25 59 4C
The contents of register 7 are not significant
Register 13 contains 00 00 76 00

The format of the conversion instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4F	7	0	D	008

Assembler Format

Op Code	$R_1, D_2 (X_2, B_2)$
CVB	7,8(0,13)

After the instruction is executed, register 7 contains 00 00 63 FA = $+25{,}594_{10}$.

Convert to Decimal (CVD)

The CONVERT TO DECIMAL instruction performs functions exactly opposite to those of the CONVERT TO BINARY instruction. CVD converts a binary number in a register to packed decimal and stores the result in a doubleword. For example, assume:

Register 1 contains 00 00 0F 0F = 3855_{10}
Register 13 contains 00 00 76 00
PSW bit 12 = 0 (EBCDIC mode)

The format of the instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4E	1	0	D	008

Assembler Format

Op Code	$R_1, D_2 (X_2, B_2)$
CVD	1,8(0,13)

After the instruction is executed, locations 7608-760F contain 00 00 00 00 00 03 85 5C.

The plus sign generated is the standard EBCDIC plus sign, 1100_2.

Divide (D, DR)

The DIVIDE instruction divides a 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 D7 = 2270_{10} = the dividend
Storage locations 3554-3557 contain 00 00 00 32 = 50_{10} = the divisor

Register 6 does not contain all zeros
The initial contents of register 7 are not significant
Register 8 contains 00 00 35 50

The following assembly language statements load the registers properly and perform the divide operation:

Statement		*Comments*
L	6,0(0,8)	Places 00 00 08 D7 into register 6
SRDA	6,32(0)	Shifts 00 00 08 D7 into register 7 Register 6 is filled with zeros (sign bits)
D	6,4(0,8)	Performs the division

The machine format of the preceding DIVIDE instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
5D	6	0	8	004

After the foregoing instructions are executed:

Register 6 contains 00 00 00 14 = 20_{10} = the remainder
Register 7 contains 00 00 00 2D = 45_{10} = the quotient

Note that if the dividend had not been first placed in register 6 and shifted into register 7, register 6 would not have been filled with the proper sign bits (zeros in this example), and the DIVIDE instruction would not have given the expected results.

Exclusive OR (X, XR, XI, XC)

When the Boolean operation EXCLUSIVE OR is applied to two bits, the result is one when one, and only one, of the two bits is one; otherwise, the result is zero. When two bytes are EXCLUSIVE ORed in System/370, each pair of bits is handled separately; there is no connection from one bit position to another.

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_2. To set the leftmost bit to one and the rightmost bit to zero without affecting any of the other bits, the following instruction can be used (assume that register 9 contains 00 00 80 80):

Machine Format

Op Code	I_2	B_1	D_1
97	81	9	002

Assembler Format

Op Code	D_1 (B_1), I_2
XI	2(9),X'81'

When the instruction is executed, the byte in storage is EXCLUSIVE ORed with the immediate byte:

Location 8082:	0110 1001_2
Immediate byte:	1000 0001_2
Result:	1110 1000_2

The resulting byte with the leftmost and rightmost bits inverted is stored in location 8082. Condition code 1 is set.

Exclusive OR (XC)

The EXCLUSIVE OR (XC) instruction can be used to change the contents of two areas in storage without the use of an intermediate storage area. For example, assume that two words are in storage:

Word 1

358			35B
00	00	17	90

Word 2

360			363
00	00	14	01

Execution of the instruction (assume that register 7 contains 00 00 03 58):

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D7	03	7	000	7	008

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
XC	0(4,7),8(7)

EXCLUSIVE ORs word 1 with word 2 as follows:

Word 1: 0000 0000 0000 0000 0001 0111 1001 0000_2
= 00 00 17 90
Word 2: 0000 0000 0000 0000 0001 0100 0000 0001_2
= 00 00 14 01

Result: 0000 0000 0000 0000 0000 0011 1001 0001_2
= 00 00 03 91

The result replaces the former contents of word 1.

Now, execution of the instruction

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D7	03	7	008	7	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
XC	8(4,7),0(7)

produces the following result:

Word 1: 0000 0000 0000 0000 0000 0011 1001 0001_2
= 00 00 03 91
Word 2: 0000 0000 0000 0000 0001 0100 0000 0001_2
= 00 00 14 01

Result: 0000 0000 0000 0000 0001 0111 1001 0000_2
= 00 00 17 90

The result of this operation replaces the former contents of word 2. Word 2 now contains the original value of word 1.

Lastly, execution of the instruction

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D7	03	7	000	7	008

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
XC	0(4,7),8(7)

produces the following result:

Word 1: 0000 0000 0000 0000 0000 0011 1001 0001_2
= 00 00 03 91

Word 2: 0000 0000 0000 0000 0001 0111 1001 0000_2
= 00 00 17 90

Result: 0000 0000 0000 0000 0001 0100 0000 0001_2
= 00 00 14 01

The result of this operation replaces the former contents of word 1. Word 1 now contains the original value of word 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* instruction in main storage to be executed out of sequence without actually branching to the object instruction. 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 located at address 3820, with a format as follows:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	00	C	003	D	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
MVC	3(1,12),0(13)

where register 12 contains 00 00 89 13 and register 13 contains 00 00 90 A0.

Further assume that at storage address 5000, the following EXECUTE instruction is located:

Machine Format

Op Code	R_1	X_2	B_2	D_2
44	1	0	A	000

Assembler Format

Op Code	R_1, D_2 (X_2, B_2)
EX	1,0(0,10)

where register 10 contains 00 00 38 20 and register 1 contains 00 0F F0 03.

When the instruction at 5000 is executed, bits 24-31 of register 1 are ORed with bits 8-15 of the instruction at 3820:

Bits 8-15:	0000 0000_2 = 00
Bits 24-31:	0000 0011_2 = 03
Result:	0000 0011_2 = 03

causing the instruction at 3820 to be executed *as if it originally were:*

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	03	C	003	D	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
MVC	3(4,12),0(13)

However, after execution:

Register 1 is unchanged
The instruction at 3820 is unchanged
The contents of the four bytes starting at location 90A0 have been moved to the four bytes starting at location 8916
The CPU next executes the instruction at address 5004 (PSW bits 40-63 contain 00 50 04)

Load (L, LR)

The LOAD instructions place, unchanged, the contents of a word in storage or of a register into another register. For example, assume that the four bytes starting with location 21004 (a fullword boundary) are to be loaded into register 10. Initially:

Register 5 contains 00 02 00 00
Register 6 contains 00 00 10 04
The contents of register 10 are not significant
Storage locations 21004-21007 contain 00 00 AB CD

To load register 10, the RX form of the instruction can be used:

Machine Format

Op Code	R_1	X_2	B_2	D_2
58	A	5	6	000

Assembler Format

Op Code	R_1, D_2 (X_2, B_2)
L	10,0(5,6)

After the instruction is executed, register 10 contains 00 00 AB CD.

Load Address (LA)

The LOAD ADDRESS instruction provides a convenient way to place a nonnegative number ≤ 4095_{10} in a register without first defining the number as a constant and then using it as an operand. For example, assume that the number 2048_{10} is to be placed in register 1. One instruction that will do this is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
41	1	0	0	800

Assembler Format

Op Code	$R_1, D_2 (X_2, B_2)$
LA	1,2048(0,0)

As indicated in the programming note in the instruction description, the LOAD ADDRESS instruction can also be used to increment a register by an amount ≤ 4095_{10} specified in the D_2 field. For example, assume that register 5 contains 00 12 34 56.

The instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
41	5	0	5	00A

Assembler Format

Op Code	$R_1, D_2 (X_2, B_2)$
LA	5,10(0,5)

adds 10 (decimal) to the contents of register 5 as follows:

Register 5 (old):	00 12 34 56
D2 field:	00 00 00 0A
Register 5 (new):	00 12 34 60

Load Halfword (LH)

The LOAD HALFWORD instruction places unchanged the contents of a halfword in storage into the right half of a register. The left half of the register is replaced by zeros or ones to reflect the sign (leftmost bit) of the halfword.

For example, assume that the two bytes in storage locations 1802-1803 are to be loaded into register 6. Also assume:

Register 6 contains 7F 12 34 56
Register 14 contains 00 00 18 02
Locations 1802-1803 contain 00 20

The instruction required to load the register is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
48	6	0	E	000

Assembler Format

Op Code	$R_1, D_2 (X_2, B_2)$
LH	6,0(0,14)

After the instruction is executed, register 6 contains 00 00 00 20. If 1802-1803 contained a negative number, for example, A7 B6, the sign bit would again be propagated to the left, giving FF FF A7 B6 as the final result in register 6.

Move (MVI)

The MOVE (immediate) instruction can place one byte of information from the instruction stream into any designated location. For example, if the instruction

Machine Format

Op Code	I_2	B_1	D_1
92	FA	0	055

Assembler Format

Op Code	$D_1 (B_1), I_2$
MVI	85(0),X'FA'

is executed, bits 8-15 of the instruction ($1111\ 1010_2$) are copied in storage location 85_{10}.

Move (MVC)

The MVC instruction can be used to move a data field from one 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:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	07	1	000	2	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
MVC	0(8,1),0(2)

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.

As indicated in the programming note in the MOVE instruction description, MVC can be used to propagate one character through a field by starting the first-operand field one byte 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):

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	07	B	001	B	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
MVC	1(8,11),0(11)

Because 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 Numerics (MVN)

To illustrate the operation of the MOVE NUMERICS instruction, assume that the following two fields are in storage:

Field 1

7090 7097

C1	C2	C3	C4	C5	C6	C7	C8

Field 2

7041 7049

F0	F1	F2	F3	F4	F5	F6	F7	F8

Also assume:

Register 14 contains 00 00 70 90
Register 15 contains 00 00 70 40

After the instruction

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D1	03	F	001	E	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
MVN	1(4,15),0(14)

is executed, field 2 becomes:

7041 7049

F1	F2	F3	F4	F4	F5	F6	F7	F8

The numeric portions of locations 7090-7093 have been stored in the numeric portions of locations 7041-7044. The contents of locations 7090-7097 and 7045-7049 are unchanged.

Move with Offset (MVO)

Assume that the unsigned three-byte field in storage locations 4500-4502 is to be moved to locations 5600-5603 and given the sign of the one-byte field located at 5603. Also assume:

Register 12 contains 00 00 56 00
Register 15 contains 00 00 45 00
Storage locations 5600-5603 contain 77 88 99 0C
Storage locations 4500-4502 contain 12 34 56

After the instruction

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F1	3	2	C	000	F	000

Assembler Format

Op Code	D_1 (L_1, B_1), D_2 (L_2, B_2)
MVO	0(4,12),0(3,15)

is executed, storage locations 5600-5603 contain 01 23 45 6C. Note that the second operand was extended with one high-order zero to fill out the first-operand field.

Move Zones (MVZ)

The MOVE ZONES instruction, similarly to MVC and MVN, can 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 high-order 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:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D3	04	F	001	F	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
MVZ	1(5,15),0(15)

propagates the zone from the byte at address 800 through 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_{10} = 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_{10} = the multiplier

The instruction required for performing the multiplication is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
5C	4	B	C	150

Assembler Format

Op Code	R_1, D_2 (X_2, B_2)
M	4,X'150'(11,12)

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 = $20{,}174_{10}$

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_{10}. The instruction

Machine Format

Op Code	R_1	R_2
1C	6	7

Assembler Format

Op Code	R_1, R_2
MR	6,7

multiplies the number in register 7 by itself:

The product, 00 00 00 00 00 00 01 00 = 256_{10}, 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_{10} = the multiplicand
Register 14 contains 00 00 01 00
Register 15 contains 00 00 20 00
Storage locations 2102-2103 contain FF D9 = −39 = the multiplier

The instruction

Machine Format

Op Code	R_1	X_2	B_2	D_2
4C	B	E	F	002

Assembler Format

Op Code	R_1, D_2 (X_2, B_2)
MH	11,2(14,15)

multiplies the two numbers. The product, FF FF FC CD = -819_{10}, replaces the original contents of register 11.

Only the 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 in System/370, each pair of bits is handled separately; there is no connection from one bit position to another.

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 $0100\ 0010_2$. To set the rightmost bit of this byte to one without affecting the other bits, the following instruction can be used (assume that register 8 contains 00 00 48 90):

Machine Format

Op Code	I_2	B_1	D_1
96	01	8	001

Assembler Format

Op Code	D_1 (B_1), I_2
OI	1(8),X'01'

When this instruction is executed, the byte in storage is ORed with the immediate byte:

Location 4891:	0100 0010$_2$
Immediate byte:	0000 0001$_2$
Result:	0100 0011$_2$

The resulting byte with bit 7 set to one is stored in location 4891. Condition code 1 is set.

Pack (PACK)

Assume that storage locations 1000-1004 contain the following zoned-decimal field that is to be converted to a packed-decimal field and left in the same location:

	1000				1004
Zoned Field	F1	F2	F3	F4	C5

Also assume that register 12 contains 00 00 10 00. After the instruction

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F2	4	4	C	000	C	000

Assembler Format

Op Code	D_1 (L_1, B_1), D_2 (L_2, B_2)
PACK	0(5,12),0(5,12)

is executed, the field in locations 1000-1004 is in the packed-decimal format:

	1000				1004
Packed Field	00	00	12	34	5C

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_1 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$_2$

the instruction

Machine Format

Op Code	R_1	R_3	B_2	D_2
8F	2	▨	0	01F

Assembler Format

Op Code	R_1, D_2 (B_2)
SLDA	2,31(0)

results in registers 2 and 3 both being left-shifted 31 bit positions, so that their new contents are:

7F 6E 5D 4C 00 00 00 00 =
0111 1111 0110 1110 0101 1101 0100 1100
0000 0000 0000 0000 0000 0000 0000 0000$_2$

In this case, a significant bit is shifted out of position 1, and a fixed-point overflow interruption occurs (unless PSW bit 36 equals zero).

Shift Left Single (SLA)

Because the sign bit remains unchanged during an SLA operation, 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$_2$

then the instruction

Machine Format

Op Code	R_1	R_3	B_2	D_2
8B	2	▨	0	008

Assembler Format

Op Code	R_1, D_2 (B_2)
SLA	2,8(0)

results in register 2 being shifted left eight bit positions so that its new contents are:

7F 0A 72 00 = 0111 1111 0000 1010 0111 0010 0000 0000$_2$

If a left shift of nine places had been specified, a significant bit would have been shifted out of position 1, and a fixed-point overflow interruption might have occurred (unless PSW bit 36 equaled zero).

Note that register 0 does not participate in the operation and that the contents of the R_3 field are ignored.

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 when it is written as:

Machine Format

Op Code	R_1	R_3	B_2	D_2
90	E	1	6	050

Assembler Format

Op Code	$R_1, R_3, D_2 (B_2)$
STM	14,1,X'50'(6)

After the instruction is executed:

Locations 4050-4053 contain	00 00 25 63
Locations 4054-4057 contain	00 01 27 36
Locations 4058-405B contain	12 43 00 62
Locations 405C-405F contain	73 26 12 57

Test Under Mask (TM)

The TEST UNDER MASK instruction examines specific bits within a byte and sets the condition code according to what it finds. For example, assume that:

Storage location 9999 contains FB
Register 9 contains 00 00 99 90

Execution of the instruction

Machine Format

Op Code	I_2	B_1	D_1
91	C3	9	009

Assembler Format

Op Code	$D_1 (B_1), I_2$
TM	9(9),X'C3'

produces the following result:

FB	= $1111\ 1011_2$
Mask (C3)	= $1100\ 0011_2$
Result	= $11xx\ xx11_2$

Condition code 3 is set: all selected bits are ones.

If location 9999 had contained B9, the result would have been:

B9	= $1011\ 1001_2$
Mask (C3)	= $1100\ 0011_2$
Result	= $10xx\ xx01_2$

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	= $0011\ 1100_2$
Mask (C3)	= $1100\ 0011_2$
Result	= $00xx\ xx00_2$

Condition code 0 is set; all selected bits are zeros.

Note: Storage location 9999 remains unchanged.

Translate (TR)

With the TRANSLATE instruction, System/370 can translate data from any 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 code into which you are translating (the function bytes). The table must be in order, not by the binary values of the characters it contains, but by the binary sequence of the characters of the original code (the argument bytes). For example, note in the following table that the characters are in the normal EBCDIC collating sequence.

Translate Table:

100F

1000																
1010																
1020																
1030																
1040	b											.		(	+	
1050	&											$	*	)		
1060	-	/										,	%			
1070												#	@	'	=	
1080		a	b	c	d	e	f	g	h	i						
1090		j	k	l	m	n	o	p	q	r						
10A0			s	t	u	v	w	x	y	z						
10B0																
10C0		A	B	C	D	E	F	G	H	I						
10D0		J	K	L	M	N	O	P	Q	R						
10E0			S	T	U	V	W	X	Y	Z						
10F0	0	1	2	3	4	5	6	7	8	9						

10FF

Notes:

1. The underscores are used to indicate the ASCII representations of the EBCDIC characters shown.
2. If the character codes in the statement being translated occupy a range smaller than 00 through FF_{16}, a table of fewer than 256 bytes can be used.

Now, assume that starting at storage location 2100 there is a sequence of 20_{10} EBCDIC characters to be translated to ASCII:

Locations 2100-2113: JOHNbJONESb257bW.b95

Also assume:

Register 12 contains 00 00 21 00
Register 15 contains 00 00 10 00

As the instruction

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DC	13	C	000	F	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
TR	0(20,12),0(15)

is executed, the binary value of each argument byte is added to the starting address of the table, and the resulting address is used to fetch a function byte:

Table starting address:	1000
First argument byte (J):	D1
Address of function byte:	10D1

Because the table is arranged so that every EBCDIC character is replaced by the corresponding ASCII character, the result is:

Locations 2100-2113: JOHNbJONESb257bW.b95

Note: To verify that this example is correct, find in appendix H the hexadecimal values for the remaining EBCDIC characters and add them to the starting address of the table (1000). The sums should be the addresses within the table of the corresponding ASCII characters.

Translate and Test (TRT)

The TRANSLATE AND TEST instruction is used to scan a data field (the argument bytes) for characters with a special meaning. To indicate which characters have special meaning, first set up a table similar to the one used for the TRANSLATE instruction. (See the preceding example.) Once again the table must be in order by the binary sequence of the code of the argument bytes. This time, however, put zeros in the table to indicate characters without any special meaning and nonzero values to indicate characters with special meaning.

This example deals with EBCDIC characters; the characters with special meaning in the argument field are a selected set of punctuation marks. The Translate-and-Test Table was set up accordingly.

Translate-and-Test Table:

200F

2000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2040	00	00	00	00	00	00	00	00	00	00	00	00	10	20	25	00
2050	90	00	00	00	00	00	00	00	00	00	00	30	35	40	45	00
2060	80	85	00	00	00	00	00	00	00	00	00	50	55	00	00	00
2070	00	00	00	00	00	00	00	00	00	00	00	60	65	70	75	00
2080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

20FF

Note: If the character codes in the statement being translated occupy a range smaller than 00 through FF_{16}, a table of fewer than 256 bytes can be used.

Now, assume that starting at storage location 3000 you have the following sequence of 30_{10} EBCDIC characters:

Locations 3000-301D: bbbbbUNPKbbbbbPROUT(9),WORD(5)

Also assume:

Register 1 contains 00 00 2F FF
Register 2 contains 00 00 00 00
Register 15 contains 00 00 20 00

As the instruction

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DD	1D	1	001	F	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
TRT	1(30,1),0(15)

is executed, the value of the first argument byte, a blank, is added to the starting address of the table to produce the address of the function byte to be examined:

Table starting address	2000
First argument byte (blank)	40
Address of function byte	2040

Because zeros were originally placed in storage location 2040, no special action occurs, and the operation continues with the second argument byte. The operation will thus continue until it reaches the symbol ((left parenthesis) in location 3013. When this symbol is reached, its value is added to the starting address of the table, as usual:

Table starting address	2000
Argument byte (left parenthesis)	4D
Address of function byte	204D

Because location 204D contains a nonzero value, the following actions occur:

1. The address of the argument byte, 003013, is placed in the low-order 24 bits of register 1.
2. The function byte, 20, is placed in the low-order eight bits of register 2.
3. Condition code 1 is set (scan not completed).

In general, TRANSLATE AND TEST is 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. In the example, after the first execution of TRT, 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 (301D) 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. (Because the length code in the machine format is one less than the total number of bytes in the field, one must be subtracted from the computed length.) The second-operand address of the EXECUTE instruction points to the TRANSLATE AND TEST instruction, which must now appear in the following format:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DD	00	1	001	F	000

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
TRT	1(0,1),0(15)

Now the entire argument field can be scanned, stopping to examine those characters of special interest, without having to modify any of the instructions already written. After a stop is made to examine a character, only a new length need be computed before continuing the scan.

Unpack (UNPK)

Assume that storage locations 2501-2503 contain a signed, packed-decimal field 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 2501-2503 contain 12 34 5D
The initial contents of storage locations 1000-1004 are not significant

After the instruction

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F3	4	2	C	000	D	001

Assembler Format

Op Code	D_1 (L_1, B_1), D_2 (L_2, B_2)
UNPK	0(5,12),1(3,13)

is executed, storage locations 1000-1004 contain F1 F2 F3 F4 D5.

DECIMAL INSTRUCTIONS

Add Decimal (AP)

Assume that the signed, packed-decimal field at storage locations 500-503 is to be added to the signed, packed-decimal field at locations 2000-2002. Also assume:

Register 12 contains 00 00 20 00
Register 13 contains 00 00 04 FD
Storage locations 2000-2002 contain 38 46 0D (a negative number)
Storage locations 500-503 contain 01 12 34 5C (a positive number)

After the instruction

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
FA	2	3	C	000	D	003

Assembler Format

Op Code	D_1 (L_1, B_1), D_2 (L_2, B_2)
AP	0(3,12),3(4,13)

is executed, storage locations 2000-2002 contain 73 88 5C; condition code 2 is set to indicate that the sum is positive. Note that:

1. Although the second-operand field is larger than the first-operand field, no overflow interruption occurs because the result can be entirely contained within the first-operand field.
2. Because the two numbers had different signs, they were in effect subtracted.

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 location 500-503. Also assume:

Register 12 contains 00 00 06 00
Register 13 contains 00 00 04 00
Storage locations 700-703 contain 17 25 35 6D
Storage locations 500-503 contain 06 72 14 2D

After the instruction

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F9	3	3	C	100	D	100

Assembler Format

Op Code	D_1 (L_1, B_1), D_2 (L_2, B_2)
CP	X'100'(4,12),X'100'(4,13)

is executed, condition code 1 is set, indicating that the first operand (the contents of locations 700-703) is lower than the second.

Divide Decimal (DP)

Assume that the signed, packed-decimal field at storage locations 2000-2004 (the dividend) is to be divided by the signed, packed-decimal field at locations 3000-3001 (the divisor). Also assume:

Instruction address: 005000
Location 5000 contains: FD 41 C 000 D 000
DP 0(5,12),0(2,13)
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 at location 5000 is executed, the dividend field is entirely replaced by the signed quotient and remainder fields, as follows:

	2000				2004
Locations 2000-2004	38	46	0D	01	8C
	Quotient			Remainder	

Notes:

1. Because the signs of the dividend and divisor are different, the quotient receives a negative sign.
2. The remainder receives the sign of the dividend and the length of the divisor.
3. If an attempt is made to divide the dividend by the one-byte field at location 3001, the quotient will be too long to fit within the four bytes allotted to it. A decimal-divide exception exists, causing a program interruption.

Edit (ED)

Because the decimal-feature instructions operate only on packed-decimal data, it is necessary to convert the data to the zoned format before a legible report can be printed. Moreover, if the report is to be useful to a great many people, certain punctuation marks, such as commas and decimal points, should be inserted in appropriate places. The highly flexible EDIT instruction performs these two functions in a single instruction execution.

This example shows step-by-step one way in which the EDIT instruction can be used. The field to be edited (the source) is four bytes long; it is edited against a pattern 13 bytes long. The following symbols are used:

Symbol	*Meaning*
b (Hexadecimal 40)	Blank character
((Hexadecimal 21)	Significance starter
d (Hexadecimal 20)	Digit selector

Assume that the source and pattern fields are:

Source

1200			1203
02	57	42	6C
			+

Pattern

1000												100C
40	20	20	6B	20	20	21	4B	20	20	40	C3	D9
b	d	d	,	d	d	(	.	d	d	b	C	R

Execution of the instruction (assume that register 12 contains 00 00 10 00)

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DE	0C	C	000	C	200

Assembler Format

Op Code	D_1 (L, B_1), D_2 (B_2)
ED	0(13,12),X'200'(12)

alters the pattern field as follows:

Pattern	*Digit*	*Significance Indicator Before/After*	*Rule*	*Location 1000-100C*
b		off/off	leave(1)	bdd,dd(.ddbCR
d	0	off/off	fill	bbd,dd(.ddbCR
d	2	off/on(2)	digit	bb2,dd(.ddbCR
,		on/on	leave	same
d	5	on/on	digit	bb2,5d(.ddbCR
d	7	on/on	digit	bb2,57(.ddbCR
(	4	on/on	digit	bb2,574.ddbCR
.		on/on	leave	same
d	2	on/on	digit	bb2,574.2dbCR
d	6+	on/off(3)	digit	bb2,574.26bCR
b		off/off	fill	same
C		off/off	fill	bb2,574.26bbR
R		off/off	fill	bb2,574.26bbb

Notes:

(1) This character becomes the fill character.
(2) First nonzero decimal source digit turns on significance indicator.
(3) Plus sign in the four low-order bits of the byte turns off significance indicator.

Thus, after the instruction is executed, the source is unchanged, and the pattern is as follows:

Pattern

1000												100C
40	40	F2	6B	F5	F7	F4	4B	F2	F6	40	40	40
b	b	2	,	5	7	4	.	2	6	b	b	b

Condition code = 2: result is greater than zero.

When printed, this new pattern field appears as:

2,574.26

If the number in the source field is changed to 00 00 02 6D, a negative number, and the original pattern is used, the edited result this time is:

Pattern

1000												100C
40	40	40	40	40	40	40	4B	F2	F6	40	C3	D9
b	b	b	b	b	b	b	.	2	6	b	C	R

Condition code = 1: result is less than zero.

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 CR symbol is also preserved.

Edit and Mark (EDMK)

After an EDIT AND MARK operation, a symbol (such as a dollar sign) can be inserted at the appropriate position in the edited result. Usually a currency symbol is inserted to the immediate left of the first significant digit in the amount; however, if a decimal point appears in an amount less than one, the currency symbol must be inserted to the immediate left of the decimal point. A typical operation would leave no blank between the currency symbol and the amount, thus protecting against one form of alteration when the result is printed on a check.

If significance is not forced by the significance starter, the EDIT AND MARK operation inserts into general register 1 an address one more than the address at which a currency symbol would normally be inserted. After one is subtracted from the value in general register 1 (for example, by using a BRANCH ON COUNT instruction with R_1 set to one and R_2 set to zero), a MOVE instruction (MVI) may be used to position the symbol in main storage.

Machine Format

Op Code	I_2	B_1	D_1
92	5B	1	000

Assembler Format

Op Code	D_1 (B_1), I_2
MVI	0(1),C'$'

If significance is forced, general register 1 remains unchanged. Therefore, the address of the character following the significance starter should be placed in the register before the EDIT AND MARK instruction is performed.

Multiply Decimal (MP)

Assume that the signed, packed-decimal field in storage locations 1202-1204 (the multiplicand) is to be multiplied by the signed, packed-decimal field in locations 500-501 (the multiplier).

	1202		1204
Multiplicand	38	46	0D

	500	501
Multiplier	32	1D

Because there is a total of eight significant digits in the multiplier and multiplicand, a field at least five bytes in length must be reserved for the signed result. As indicated in the programming note for MULTIPLY DECIMAL, a ZERO AND ADD into a larger field can provide the required space. If it is assumed

Register 4 contains 00 00 12 00
Register 6 contains 00 00 05 00

then execution of the assembler instruction

ZAP X'100'(5,4),2(3,4)

sets up a new multiplicand in storage locations 1300-1304 as follows.

	1300				1304
Multiplicand (new)	00	00	38	46	0D

Now, after the instruction

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
FC	4	1	4	100	6	000

Assembler Format

Op Code	D_1 (L_1, B_1), D_2 (L_2, B_2)
MP	X'100'(5,4),0(2,6)

is executed, storage locations 1300-1304 contain the product 01 23 45 66 0C.

Shift and Round Decimal (SRP)

The SRP instruction can be used for shifting decimal fields in main storage. When the field 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, and its contents are shown in "FIELD1 (before)" below. The following SRP instruction performs the above operation:

Machine Format

Op Code	L_1	I_3	S_1	B_2	D_2
F0	5	0	****	0	003

Assembler Format

Op Code	S_1 (L_1), S_2, I_3
SRP	FIELD1(6),3,0

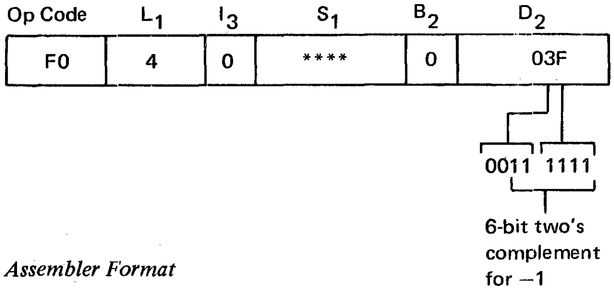
FIELD1 (before): 00 01 23 45 67 8C
FIELD1 (after): 12 34 56 78 00 0C

The second-operand address in this instruction specifies the shift amount (three places) completely in the D_2 field. The rounding factor, I_3, is not used in left shift, but it must be a valid decimal digit.

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 SRP instruction performs this operation:

Machine Format



Assembler Format

Op Code	S_1 (L_1), S_2, I_3
SRP	FIELD2(5),64 −1,0

FIELD2 (before): 01 23 45 67 8C
FIELD2 (after): 00 12 34 56 7C

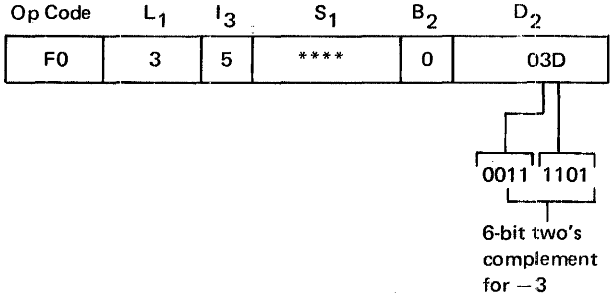
The second-operand address specifies the shift amount (one place) completely in the D2 field. In the SRP instruction, shifts to the right are specified by negative shift values, which are represented as a six-bit value in two's-complement form.

The six-bit two's complement of a number, n, can be represented as 64 − n. In this example, a right shift of one is represented as 64 −1.

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 1,000 and rounding to the nearest whole number. FIELD3 is four bytes in length.

Machine Format



Assembler Format

Op Code	S_1 (L_1), S_2, I_3
SRP	FIELD3(4),64 −3,5

FIELD3 (before): 12 39 60 0C
FIELD3 (after): 00 01 24 0C

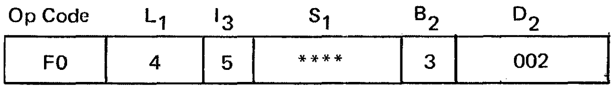
The shift amount (three places) is specified in the D_2 field. The I_3 field specifies the rounding factor of 5. The rounding factor is added to the last digit shifted out (which is a 6) and the carry, if any, is propagated to the left. Since 5 + 6 in decimal totals one, plus a carry, a carry is propagated in the above example, and, as a result, 1239.6 becomes 1240.

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 field by 10 raised to the power specified by the shift value.

In this example, the SRP instruction is used to adjust a decimal field by a variable scale factor contained in a general register. Main storage location FIELD4 contains a decimal integer (the decimal point is implied to be on the right). FIELD4 is five bytes in length. Register 3 contains a fixed-point binary value that is the scale factor of FIELD4 (the power of 10 by which FIELD4 is multiplied). The following SRP instruction adjusts FIELD4 so that the implied decimal point retains the two nearest decimal places (requiring the implied decimal point to shift two places to the left, under control of the D_2 field) and multiplies FIELD4 by the variable power of 10 contained in register 3:

Machine Format



Assembler Format

Op Code	S_1 (L_1), D_2 (B_2), I_3
SRP	FIELD4(5),2(3),5

FIELD4 (before): 00 00 00 12 7C

Case 1: Register 3 contains 00 00 00 00

If the scale factor is zero, FIELD4 represents $127 \times 10^0 = 127$

FIELD4 (after): 00 00 12 70 0C

The implied decimal point is now shifted two places to the left.

Case 2: Register 3 contains 00 00 00 03

If the scale factor is 3, FIELD4 represents $127 \times 10^3 = 127\ 000$

FIELD4 (after): 01 27 00 00 0C

The implied decimal point is now shifted two places to the left.

Case 3: Register 3 contains FF FF FF FD
If the scale factor is –3, FIELD4 represents
$127 \times 10^{-3} = 0.127$
FIELD4 (after): 00 00 00 01 3C
The implied decimal point is now shifted two places to the left; because the shift was to the right. FIELD4 is rounded to the nearest two decimal places.

In the preceding cases, the implied decimal is shifted two places to the left, under control of the D_2 field in the SRP instruction. The shifting is controlled by the address that is resolved when the contents of the base register (GR 3) are added to the displacement D_2, effecting the multiplication of FIELD4 by a variable power of 10.

Zero and Add (ZAP)

Assume that the signed, packed-decimal field at storage locations 4500-4502 is to be moved to locations 4000-4004 with four leading zeros in the result field. Also assume:

Register 9 contains 00 00 40 00
Storage locations 4000-4004 contain 12 34 56 78 90
Storage locations 4500-4502 contain 38 46 0D

After the instruction

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F8	4	2	9	000	9	500

Assembler Format

Op Code	D_1 (L_1, B_1), D_2 (L_2, B_2)
ZAP	0(5,9),X'500'(3,9)

is executed, the storage locations 4000-4004 contain 00 00 38 46 0D; condition code 1 is set to indicate a negative result. Note that because the first operand is not checked for valid sign and digit codes, it may contain any combination of hexadecimal digits.

FLOATING-POINT INSTRUCTIONS

In this section, the abbreviations FPR0, FPR2, FPR4, and FPR6 stand for floating-point registers 0, 2, 4, and 6, respectively.

Add Normalized (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:

FPR6 contains 43 08 21 00 00 00 00 00 = 82.1_{16}
= approximately 130.06_{10}
Storage locations 2000-2007 contain 41 12 34 56 00 00 00 00
= 1.234561_{16} = approximately 1.13_{10} (normalized)
Register 13 contains 00 00 20 00

The instruction

Machine Format

Op Code	R_1	X_2	B_2	D_2
7A	6	0	D	000

Assembler Format

Op Code	R_1, D_2 (X_2, B_2)
AE	6,0(0,13)

can be used to perform the short-precision addition of the two operands. In this example, the instruction operates as follows:

The characteristics of the two numbers are compared. Since the number in storage has a characteristic that is smaller by 2, it is right-shifted after fetching until the characters agree. The two numbers are then added:

		Guard Digit
FPR6:	43 08 21 00	
Shifted number from storage:	43 00 12 34	5
Intermediate sum:	43 08 33 34	5

Because the intermediate sum is unnormalized, it is left-shifted to form the normalized floating-point number 42 83 33 45 (= 83.3345_{16} = 131.2_{10}). This number replaces the high-order portion of FPR6. The low-order portion of FPR6 and the contents of storage locations 2000-2007 are unchanged.

If the long-precision instruction AD is used, the result in FPR6 will be 42 83 33 45 60 00 00 00. Note that, in this case, the use of the long-precision instruction provides one additional hexadecimal digit of precision.

Add Unnormalized (AU, AUR, AW, AWR)

The ADD UNNORMALIZED instructions operate identically to the ADD NORMALIZED instructions, except that the final result is not normalized when ADD UNNORMALIZED is used. For example, using the same operands as in the example for ADD NORMALIZED, when the short-precision instruction

Machine Format

Op Code	R_1	X_2	B_2	D_2
7E	6	0	D	000

Assembler Format

Op Code	$R_1, D_2 (X_2, B_2)$
AU	6,0(0,13)

is executed, the two numbers are added as follows:

					Guard Digit
FPR6:	43	08	21	00	
Shifted number from storage:	43	00	12	34	5
Sum:	43	08	33	34	5

The guard digit participates in the addition but is discarded. The unnormalized sum replaces the high-order portion of FPR6.

If the result in FPR6 is converted to a normalized number (42 83 33 40 00 00 00 00) and is compared to the result in FPR6 when ADD NORMALIZED was used (42 83 33 45 00 00 00 00), it is apparent in this case that the use of ADD NORMALIZED (with the retention of the guard digit) has preserved some additional significance in the result.

Compare (CE, CER, CD, CDR)

Assume that FPR4 contains 43 00 00 00 00 00 00 00 (= 0), and FPR6 contains 34 12 34 56 78 9A BC DE (a positive number). The contents of the two registers are to be compared with the following long-precision instruction:

Machine Format

Op Code	R_1	R_2
29	4	6

Assembler Format

Op Code	R_1, R_2
CDR	4,6

When this instruction is executed, the number with the smaller characteristic is taken from the register and right-shifted until 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 comparèd, more than two numbers may compare equally if one of the numbers is unnormalized. For example, the unnormalized floating-point number 41 00 12 34 56 78 9A BC compares equally with all numbers of the form 3F 12 34 56 78 9A BC 0X (X represents any hexadecimal number). 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.

Note, however, that when two normalized floating-point numbers are compared, the relationship between numbers that compare equally is unique: each digit in one number must be identical to the corresponding digit in the other number.

MULTIPROCESSING EXAMPLES

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, control words, and other common storage areas.

Setting a Single Bit

In a multiprocessing system, two central processors have access to the same main storage, and both can fetch, modify, and store data in the same locations (such as in a system control block). In this configuration, if the OR (immediate) instruction is used to modify storage, such as when setting a flag bit, program logic errors may occur.

Example of a Program Failure Using OR Immediate

Assume that two independently processing programs wish 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*
	X'00'	Fetch FLAGS X'00'
Fetch FLAGS X'00'	X'00'	
	X'00'	OR X'80' into X'00'
OR X'01' into X'00'	X'00'	
	X'80'	Store X'80' into FLAGS
Store X'01' 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 program executing on CPU B now has invalid data.

The COMPARE AND SWAP (CS) instruction is included in System/370 to overcome this and similar problems. The CS instruction first checks the value of a storage location and then modifies it only if it is the same as the program expects; normally, this would be a previously fetched value. If the location is not what the program expects, then the location is not modified, but rather 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 CS execution, no other CPU can access the subject storage location.

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 modify 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
         BNE   RETRY       If new flags not stored, try again
```

The format of the CS instruction is:

Machine Format

Op Code	R_1	R_3	S_1
BA	5	4	****

Assembler Format

Op Code	R_1, R_3, S_1
CS	5,4,WORD

The CS instruction compares the first operand (register 5 containing the original flag values) to the second operand (word) while storage access to the subject location is not permitted to any CPU other than the one executing the CS instruction.

If the compare 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 example 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.

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 a register. The original counter is then moved into another register to provide a modifiable copy, and a register (containing an increment to the counter) is added to the modifiable copy to provide the updated counter value. The CS instruction is then used to ensure a valid store of the counter. The following instruction sequence performs this procedure:

```
     LA  6,1        Put increment (1) in register
     L   5,CNTR     Get original counter value
LOOP LR  4,5        Set up copy to modify
     AR  4,6        Update counter in register
     CS  5,4,CNTR   Update counter in storage
     BNE LOOP       If original value changed, update new
                    value

CNTR (before): 00 00 00 10
CNTR (after):  00 00 00 11
```

The program updating the counter byte then checks the result by examining the condition code. Condition code 0 indicates a successful update, and the program can proceed. If the counter byte 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 control byte value into register 5 and set the condition code to 1, indicating an unsuccessful update. The program would then have to update the new counter value in register 5 and retry the CS instruction, retesting the condition code, and retrying, until a successful update is completed.

The following shows two CPUs, A and B, executing the above program simultaneously. That is, both CPUs desire to add one to CNTR.

CPU A			*CPU B*		*Comments*
Reg4	*Reg5*	*CNTR*	*Reg4*	*Reg5*	
		16			
16	16	16			CPU A loads Reg5 and Reg4 from CNTR
16	16	16	16	16	CPU B loads Reg5 and Reg4 from CNTR
		16	17	16	CPU B adds 1 to Reg4
17	16	16			CPU A adds 1 to Reg4
17	16	17			CPU A executes CS; successful match store
		17	17	17	CPU B executes CS; no match, Reg5 changed
			18	17	CPU B loads Reg5 into Reg4 and adds 1 to Reg4
		18	18	17	CPU B executes CS; successful match store

EXAMPLES OF THE USE OF COMPARE AND SWAP

The following examples of the use of the COMPARE AND SWAP instruction illustrate the applications for which the instruction is intended. It is important to note that these are examples of functions that can be performed by programs running enabled or by programs that are running on a shared-main-storage multiprocessor.

Interlocked Single-Word, or Smaller, Serially Reusable Resource (SRR)

The following routine allows a program to modify the contents of a storage location while running enabled, even though the possibility exists that another CPU may simultaneously update the same location and even though the routine may be interrupted by another program that updates the location.

SRR UPDATE Routine

Initial conditions:

General register (GR) 2 contains the address of the word to be updated.

```
           L     3,0(2)   FETCH THE WORD
TRYAGN     LR    4,3      MOVE IT TO GR4
[ANY INSTRUCTION]         MODIFY GR4 (e.g., add a constant,
                          set a bit, etc.)
           CS    3,4,0(2) STORE THE NEW VALUE IF THE
                          WORD HAS NOT CHANGED
           BNE   TRYAGN
```

The branch to TRYAGN is different from "bit-spinning" in that the branch will be taken only if some other program modifies the update location. If a number of CPUs simultaneously attempt to modify one location, one CPU will fall through the loop, another will loop once, and so on until all CPUs have succeeded.

Bypassing POST

The following routine allows the SVC "POST" as used in OS/VS to be bypassed whenever the corresponding WAIT has not yet been issued, provided that the supervisor WAIT and POST routines use COMPARE AND SWAP to manipulate event control blocks (ECBs).

BYPASS POST Routine

Initial conditions:

GR1 contains the address of the ECB.

GR0 contains the POST code.

```
HSPOST L     3,0(1)    GR3 = CONTENTS OF ECB
       LTR   3,3       ECB MARKED 'WAITING'
       BM    PSVC      YES, ISSUE AN SVC
       CS    3,0,0(1)  NO, STORE POST CODE
       BE    EXIT      CONTINUE
PSVC   SVC   POST      ECB ADDRESS IS IN GR1,
                       POST CODE IN GR0
EXIT   [ANY INSTRUCTION]
```

A corresponding bypass WAIT function, using TM, is in use at present.

The following routine may be used in place of the previous HSPOST routine if the ECB is assumed to contain zeros when it is not marked "WAITING."

```
HSPOST SR    0,0
       CS    0,2,0(1)
       BE    EXIT
       SVC   POST
EXIT   [ANY INSTRUCTION]
```

Lock/Unlock

When SRRs larger than a doubleword are to be updated, it is usually necessary to provide special interlocks to ensure that a single program at a time updates the SRR. In general, updating a list, or even scanning a list, cannot be safely accomplished without first "freezing" the list. However, the COMPARE AND SWAP instructions can be used in certain restricted situations to perform queuing and list manipulation. Of prime importance is the capability to perform the lock/unlock functions and to provide sufficient queuing to resolve contentions, either in a LIFO or FIFO manner. The lock/unlock functions can then be used as the interlock mechanism for updating an SRR of any complexity.

The lock/unlock functions are based on the use of a header associated with the SRR. The header is the common starting point for determining the states of the SRR, either free or in use, and is also used for queuing requests when contentions occur. Contentions are resolved using WAIT and POST. Although the examples do not show it, it is expected that the BYPASS WAIT and BYPASS POST would be used. The general programming technique requires that the program that encounters a locked SRR must "leave a mark on the wall" indicating the address of an ECB on which it will WAIT. The program "unlocking" sees the mark and posts the ECB, thus permitting the waiting program to continue. In the two examples given, all programs using a particular SRR must use either the LIFO queuing scheme or the FIFO scheme; the two cannot be mixed. When more complex queuing is required, it is suggested that the queue for the SRR be locked using one of the two methods shown.

Lock/Unlock with LIFO Queuing for Contentions

The header consists of a word, which can contain zero, a positive value, or a negative value.

- A zero value indicates that the SRR is free.
- A negative value indicates that the SRR is in use but no additional programs are waiting for the SRR.
- A positive value indicates that the SRR is in use and that one or more additional programs are waiting for the SRR. Each waiting program is identified by an element in a chained list. The positive value in the header is the address of the element most recently added to the list.

Each element consists of two words. The first word is used as an ECB; the second word is used as a pointer to the next element in the list. A negative value in a pointer indicates that the element is the last element in the list. The element is required only if the program finds the SRR locked and desires to be placed in the list.

The following chart describes the action taken for LIFO LOCK and LIFO UNLOCK routines.

	Action		
Function	*Header Contains Zero*	*Header Contains Positive Value*	*Header Contains Negative Value*
LIFO LOCK (The incoming element is at location A)	SRR is free. Set the header to a negative value. Use the SRR.	SRR is in use. Store the contents of the header into location A +4. Store the address A into the header. WAIT; the ECB is at location A.	
LIFO UNLOCK	Error	Someone is waiting for the SRR. Move the pointer from the "last in" element into the header. POST; the ECB is in the "last in" element.	The list is empty. Store zeros into the header. The SRR is free.

The following routines allow enabled code to perform the actions described in the previous chart.

LIFO LOCK Routine:

Initial conditions:

GR1 contains the address of the incoming element.

GR2 contains the address of the header.

```
LLOCK    SR   3,3       GR3 = 0
         ST   3,0(1)    INITIALIZE THE ECB
         LNR  0,1       GR0 = A NEGATIVE VALUE
TRYAGN   CS   3,0,0(2)  SET THE HEADER TO A NEGATIVE
                        VALUE IF THE HEADER CON-
                        TAINS ZEROS
         BE   USE       DID THE HEADER CONTAIN
                        ZEROS?
         ST   3,4(1)    NO, STORE THE VALUE OF THE
                        HEADER INTO THE POINTER IN
                        THE INCOMING ELEMENT
         CS   3,1,0(2)  STORE THE ADDRESS OF THE
                        INCOMING ELEMENT INTO THE
                        HEADER
         LA   3,0(0)    GR3 = 0
         BNE  TRYAGNDID THE HEADER GET UPDATED?
         WAIT (1)       YES, WAIT FOR THE RESOURCE;
                        THE ECB IS IN THE INCOMING
                        ELEMENT
USE      [ANY INSTRUCTION]
```

LIFO UNLOCK Routine:

Initial conditions:

GR2 contains the address of the header.

```
LUNLK  L     1,0(2)    GR1 = THE CONTENTS OF THE
                       HEADER
A      LTR   1,1       DOES THE HEADER CONTAIN A
       BM    B         NEGATIVE VALUE?
       L     0,4(1)    NO, LOAD THE POINTER FROM THE
       CS    1,0,0(2)  "LAST IN" ELEMENT AND STORE IT
                       IN THE HEADER
       BNE   A         DID THE HEADER GET UPDATED?
       POST  (1)       YES, POST THE "LAST IN" ELEMENT
       B     EXIT      CONTINUE
B      SR    0,0       THE HEADER CONTAINS A NEGATIVE
       CS    1,0,0(2)  VALUE; FREE THE HEADER AND
       BNE   A         CONTINUE
EXIT   [ANY INSTRUCTION]
```

Note that the L 1,0(2) instruction at location LUNLK would have to be CS 1,1,0(2) if it were not for the rule that a fullword fetch starting on a word boundary must fetch the word such that if another CPU changes the word being fetched, either the entire new or the entire old value of the word, and not a combination of the two, is obtained.

LOCK/UNLOCK with FIFO Queuing for Contentions

Both a header and a free element are associated with the SRR. Each program using the SRR must provide an element regardless of whether contention occurs. The element provided by the program becomes the new free element, and the old free element becomes the program's new current element. The free element is initialized to contain a posted ECB. In the example, the element need be only a single word. In some cases, the element could be made larger to include a reverse pointer to the previous element.

The following chart describes the action taken for FIFO LOCK and FIFO UNLOCK routines.

Function	*Action*
FIFO LOCK (The incoming element is at location A)	Store the address A into the header. WAIT; the ECB is at the location addressed by the old contents of the header.
FIFO UNLOCK	POST; the ECB is at the location that you specified as an element when you locked.

The following routines allow enabled code to perform the actions described in the previous chart.

FIFO LOCK Routine:

Initial conditions:

PNTRS is a doubleword containing two pointers. The first word is a pointer to the current element owned by this program. The second word is a pointer to the previous element owned by this program.

GR3 contains the address of the header.

```
FLOCK    L     2,PNTRS    GR2 = ADDRESS OF THE
                          CURRENT ELEMENT
         SR    1,1        GR1 = ZERO
         ST    1,0(2)     INITIALIZE THE ECB
         L     1,0(3)     GR1 = CONTENTS OF THE
                          HEADER, ADDRESS OF THE
                          OLD ELEMENT
TRYAGN   CS    1,2,0(3)   STORE THE ADDRESS OF THE
         BNE   TRYAGN     CURRENT ELEMENT INTO THE
                          HEADER, CURRENT ELEMENT
                          BECOMES NEW FREE ELEMENT
         STM   1,2PNTRS   SAVE THE ADDRESSES IN GR1
                          AND GR2 FOR FUTURE USE,
                          CURRENT ELEMENT BECOMES
                          PREVIOUS ELEMENT, OLD
                          FREE ELEMENT BECOMES
                          CURRENT ELEMENT
         WAIT  (1)        GR1 CONTAINS THE ADDRESS
                          OF THE ECB
USE      [ANY INSTRUCTION]
```

FIFO UNLOCK Routine:

Initial conditions:

PNTRS is the same location used in the FIFO LOCK routine.

```
FUNLK      L      1,PNTRS+4  GR1 CONTAINS THE ADDRESS
           POST (1)          OF THE PREVIOUS ELEMENT,
                             THAT IS, THE ELEMENT THAT
                             WAS ADDED IN THE FLOCK
                             ROUTINE.
CONTINUE  [ANY INSTRUCTION]
```

Free-Pool-List Manipulation

It is anticipated that a program will need to add and delete items from a free list without using the lock/unlock routines. This is especially likely since the lock/unlock routines require storage elements for queuing and may require working storage. The lock/unlock routines discussed previously allow simultaneous "lockers" but permit only one "unlocker" at a time. In such a situation, multiple additions and a single deletion to the list may all occur simultaneously, but multiple deletions cannot occur at the same time. In the case of a chain of pointers containing free storage buffers, multiple deletions along with additions can occur simultaneously. In this case, the removal cannot be done using the CS instruction without a certain degree of exposure.

Consider a chained list of the type used in the LIFO lock/unlock example. Assume that the first two elements are at locations A and B, respectively. If one program attempted to remove the first element and was interrupted between the fourth and fifth instructions of the LUNLK routine, the list could be changed so that elements A and C are the first two elements when the interrupted program resumes execution. The CS instruction would then succeed in storing the value B into the header, thereby destroying the list.

The probability of the occurrence of such list destruction can be reduced to *near* zero by appending to the header a counter that indicates the number of times elements have been added to the list. The use of a 32-bit counter guarantees that the list will not be destroyed unless the following events occur, in this exact sequence:

1. An unlocker is interrupted between the fetch of the pointer from the first element and the update of the header.
2. The list is manipulated, including the deletion of the element referenced in step 1, and exactly $2^{32} - 1$ additions to the list are performed. Note that this takes on the order of days to perform in any practical situation.
3. The element referenced in step 1 is added to the list.
4. The unlocker interrupted in step 1 resumes execution.

The routines ADD TO FREE LIST and DELETE FROM FREE LIST use such a counter in order to allow multiple, simultaneous additions and removals at the head of a chain of pointers.

The list consists of a doubleword header and a chain of elements. The first word of the header contains a pointer to the first element in the list. The second word of the header contains a 32-bit counter indicating the number of additions that have been made to the list. Each element contains a pointer to the next element in the list. A zero value indicates the end of the list.

The following chart describes the free-pool-list manipulation.

	Action	
Function	*Header=0, Count*	*Header=A, Count*
ADD TO LIST (The incoming element is at location A)	Store the first word of the header into location A. Store the address A into the first word of the header. Decrement the second word of the header by one.	
DELETE FROM LIST	The list is empty.	Set the first word of the header to the value of the contents of location A. Use element A.

The following routines allow enabled code to perform the free-pool-list manipulation described in the chart.

ADD TO FREE LIST Routine:

Initial conditions:

GR2 contains the address of the element to be added.
GR4 contains the address of the header.

```
ADDQ    LM     0,1,0(4)  GR0,GR1 = CONTENTS OF THE
                         HEADER
TRYAGN ST      0,0(2)    POINT THE NEW ELEMENT TO
                         THE TOP OF THE LIST
        LR     3,1       MOVE THE COUNT TO GR3
        BCTR   3,0       DECREMENT THE COUNT
        CDS    0,2,0(4)  UPDATE THE HEADER
        BNE    TRYAGN
```

DELETE FROM FREE LIST Routine:

Initial conditions:

GR4 contains the address of the header.

```
DELETQ  LM     2,3,0(4)  GR2,GR3 = CONTENTS OF THE
                         HEADER
TRYAGN LTR     2,2       IS THE LIST EMPTY?
        BZ     EMPTY     YES, GET HELP
        L      0,0(2)    NO, GR0 = THE POINTER FROM
                         THE FIRST ELEMENT
        LR     1,3       MOVE THE COUNT TO GR1
        CDS    2,0,0(4)  UPDATE THE HEADER
        BNE    TRYAGN
USE   [ANY INSTRUCTION]  THE ADDRESS OF THE REMOVED
                         ELEMENT IS IN GR2
```

Note that the LM instructions at locations ADDQ and DELETQ would have to be CDS instructions if it were not for the rule that a doubleword fetch starting on a doubleword boundary must fetch the doubleword such that if another CPU changes the doubleword being fetched, either the entire new or the entire old value of the doubleword, and not a combination of the two, is obtained.

Index

Where more than one page-reference is given, major references appear first.

absolute address 95, 14
absolute main storage 92
access control bits (in key in storage) 38
access exception 80
  handling of (table) 81
  priority of 83
  recognition of 80
  recognition of (table) 83
access to main storage, right of 38
accesses (references), sequence of main storage 23
active, state of address translation table entry 65
adapter, channel-to-channel 186
ADD (A, AR) instruction 117
ADD DECIMAL (AP) instruction 149
  example 305
ADD HALFWORD (AH) instruction 117
  example 291
ADD LOGICAL (AL, ALR) instruction 120
ADD NORMALIZED (ADR, AD, AER, AE) instruction 160
  example 309
ADD NORMALIZED (AXR) instruction 160
ADD UNNORMALIZED (AWR, AW, AUR, AU) instruction 162
  example 309
address
  absolute 95, 14
  base 21
  branch 22
  channel/device 192
  failing-storage (*see* failing-storage address)
  invalid 77
  logical 14, 58
  of channel command word, in CSW 229
  real 95, 14, 57
  translation (*see* dynamic address translation)
  virtual 57
address arithmetic (generation) 21
address-compare controls 244
address generation 21
address identification, CPU 101
addresses
  handling of 63
  translated (*see* dynamic address translation)
  types of 62
addressing
  capability 14
  information in a register 20
  limitations of 14
  main storage 14
  wraparound 14
addressing exception 76, 15
  during address translation 61, 62, 81
  summary table 77
addressing, I/O
  channel 191
  device 191
  nonexistent or protected areas 213
alert condition (machine-check interruption) 175
  degradation 179
  warning 179
alteration mask for program-event recording 40
alteration of an instruction by EXECUTE 129
AND (NR, N, NI, NC) instruction 120
  example 292
arithmetic (*see* decimal instructions; floating-point-instructions; general instructions)
assembly language, symbolic operand designations for System/370 (*see* individual instruction descriptions)
assigned main storage locations 90
  absolute 91
  real 90
asynchronous fixed logout (*see* machine-check fixed logout)
asynchronous fixed logout control bit (in control register 14) 182
asynchronous machine-check logout (*see* machine-check extended logout and machine-check fixed logout)
asynchronous MCEL control bit (in control register 14) 182
attached, state of address translation table entry 65
attachment of I/O devices 186
attention (I/O unit status condition) 229, 239
availability, System/370 facilities for achieving 11
available (I/O system state) 193

B field of an instruction 20, 21
backed-up bit (machine-check interruption code) 179
base address (in operand designation) 21
basic-control mode 31
  PSW format 33
  (*see also* extended-control mode)
BC mode (*see* basic-control mode)
binary notation, excess-64 159
bit, check 14
bits in a byte 14
block-concurrent references to storage 27
block of data, I/O
  definition 210
  self-describing 214
block-multiplexer channel 188
block-multiplexing-control bit 189
block-multiplexing mode bit (in control register 0) 189
blocking of data (I/O operations) 210
boundaries in main storage, integral 15
branch address 22
BRANCH AND LINK (BAL, BALR) instruction 121
  example 292
BRANCH ON CONDITION (BC, BCR) instruction 121
  example 292
BRANCH ON COUNT (BCT, BCTR) instruction 122
  example 293
BRANCH ON INDEX HIGH (BXH) instruction 122
  example 293
BRANCH ON INDEX LOW OR EQUAL (BXLE) instruction 123
  example 293
branch, successful (program event) 42
branching, general description of 22
burst mode 187
bus out check (sense data) 220

busy (I/O unit status condition) 230, 238
byte (definition) 14
byte index field 58
byte-interleave mode 187
byte-multiplexer channel 188
byte-oriented-operand feature 15

cache 13
CAI (channel available interruption) 227
CAW (channel address word) 210, 190
CBC (checking block code) 171
CBC, invalid
  handling of
    in keys in storage 173
    in registers 173
    in storage 172
CCW (channel command word) 189, 211
  address in CAW 210
  address in CSW 229
  command code 212
  definition 211
CCW1 (IPL) in absolute main storage 91
CCW2 (IPL) in absolute main storage 91
central processing unit (*see* CPU)
chain command (CC) flag (in CCW) 211
chain data (CD) flag (in CCW) 211
chaining check (channel status condition) 235, 237
chaining, command 215, 279
chaining data 213, 191
change bit 57, 67
channel 187
  address 192
  burst mode 187
  byte-interleave mode 187
  commands 217
  compatibility of operation 191
  control check (status condition) 235, 237
  data check (status condition) 235, 237
  description of 17
  equipment error 196
  identification (ID) 240, 206
  indirect data addressing (CIDA) 216
  mask, BC mode (*see* extended control mode) 33
  masks (in control register 2) 88
  modes of operation 187
  not operational (state of I/O system) 194
  program, termination of (or conclusion of) 222
  subchannel 188
  types 188
  working (state of I/O system) 194
channel address validity flag 241
channel address word (CAW) 190, 210
channel available interruption (CAI) 227
channel command word (*see* CCW)
channel end (I/O unit status condition) 231, 238
channel ID 240, 92, 206
channel logout 240
channel status conditions 233
  chaining check 235
  channel control check 235
  channel data check 235
  incorrect length 233
  interface control check 235
  program check 234
  program-controlled interruption 233
  protection check 234
channel status word (CSW) 228
channel-to-channel adapter 186
characteristic in floating-point operand 158
check (or checking) bit 171, 14
check-stop indication 245
  control bit (in control register 14) 181
  status bit 99
check-stop state 31
checking block 171
  code (*see* CBC)
class number, monitor 39
classes of interruptions (*see* interruption classes)
CLEAR I/O (CLRIO) instruction 198
clock
  interval timer as a real-time 49
  time-of-day (TOD) 46
clock comparator 47
  interruption 48, 87
  interruption submask 88
  mask bit (in control register 0) 87
  priority of interruption 86
  valid bit in machine-check interruption code 180
  save area (machine-check extended interruption information, timing facilities) (*see* timing facilities) 177
code
  command 217
  condition (*see* condition code in PSW)
  instruction length 71, 34, 72
  interruption, in BC mode PSW 34, 72
  monitor 39
  operation 19
  PER (program-event recording) 40
codes, decimal sign, and zone 148
command
  control 219
  I/O 210, 217
  read 218
  read backward 218
  retry 221
  sense 219
  transfer in channel 221
  write 218
command address, I/O 236
  in CAW 210
  in CSW 229, 235
command address/key validity flag 241
command chaining 215, 191
command code (in CCW) 211
command, immediate (*see* immediate operation) 222
command reject (sense data) 220
command retry 221
communications area, I/O 239
COMPARE (C, CR) instruction 123
COMPARE (CDR, CD, CER, CE) instruction 163
  example 310
COMPARE AND SWAP (CS) instruction 123
  example 310
COMPARE DECIMAL (CP) instruction 149
  example 305
COMPARE DOUBLE AND SWAP (CDS) instruction 124
COMPARE HALFWORD (CH) instruction 125
  example 294
COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) instruction 126
  example 295

- COMPARE LOGICAL (CLR, CL, CLI, CLC) instruction 125
  - example 294
- COMPARE LOGICAL LONG (CLCL) instruction 126
  - example 295
- compatibility and compatibility limitations in System/370 2
- compatibility of I/O operations 191
- completion (method of ending instruction execution) 74
- conceptual sequence (order) in instruction execution 23
- concluding (termination of) I/O
  - at operation initiation 222
  - by HALT I/O or HALT DEVICE 224
  - due to equipment malfunction 226
  - of data transfer 223
  - of I/O operations 222
- concurrent within a block, storage references 25
- condition code (CC)
  - deferred (DCC), in CSW 229
  - setting for I/O instructions 195
  - setting, summary of (*see* Appendix E)
- condition code in PSW 22
  - BC mode 34
  - EC mode 34
  - program mask and condition code validity bit 180
- conditions
  - determining response to orders 98
  - external interruption 84
  - I/O interruption 226, 88
  - precluding interpretation of an order code 98
  - program interruption 75
- configuration controls 245
- console device (of the system console) 243
- console, system 243, 18
- control
  - check-stop 181
  - logout 181
  - panel, system 243
  - store status 54
  - system 29
- control command, I/O 219
- control panel, system 243
- control register 36, 16
  - field and bit assignments 37
  - machine check controls 181, 183
  - save area (machine-check extended interruption information) 177
  - valid bit (machine-check interruption code) 181
- control unit 187
  - address in device address 192
  - attachment in system 187
  - description of 18
  - functions 187
  - selection 192
- control unit end (I/O unit status condition) 230
- CONVERT TO BINARY (CVB) instruction 127
  - example 296
- CONVERT TO DECIMAL (CVD) instruction 128
  - example 296
- count 236
  - in CCW 211
  - in CSW 238
- counter, instruction (*see* instruction address in PSW)
- coupled general registers 16
- CPU (central processing unit)
  - address identification 101
  - general description 15
  - initialization reset 51
  - power-on reset 53
  - reset 50
  - reset order 98
  - retry 172
  - signaling and response 97
- CPU states 30
  - problem/supervisor 30
  - stopped/operating 30
  - wait/running 30
- CPU timer 48
  - interruption 48, 88
  - interruption submask 88
  - mask bit (in control register 0) 88
  - priority of interruption 86
  - save area (machine-check extended interruption information timing facilities) (*see* timing facilities) 177
  - valid bit in machine-check interruption code 180
- CSW (channel status word) 228
- current PSW 16, 22
- customer-engineer-control section (of the system control panel) 248

- D field of an instruction 20
- damage condition 175
- DAT (*see* dynamic address translation)
- data
  - address (in CCW) 211
  - block, I/O (definition) 210
  - chaining (in I/O operations) 213, 191
  - check (sense data) 220
  - exception 78
  - format
    - decimal instruction 147
    - fixed-point numbers 116
    - floating-point instruction 157
    - general instruction 1,16
  - formats 14
  - prefetching and buffering of in channel 212
  - transfer (I/O)
    - concluding of 223
    - modes of 187
- decimal
  - data format 147
  - divide exception 78
  - number representation 148
  - operands 147
  - overflow exception 78
  - packed 147
  - sign codes 147, 148
  - zoned 147
- decision making by BRANCH ON CONDITION instruction 22
- deferred condition code (DCC) 229
- degradation
  - machine-check interruption condition 179, 175
  - report mask bit 182
- delayed bit, machine-check interruption code 180
- deletion, unit 172
- destructive overlap (in MOVE LONG) 133
- detect field (in limited channel logout) 240

- device
  - addressing, I/O 192
    - validity flag 241
  - description 18
  - end (I/O unit status condition) 232
  - error, I/O 197
  - general information 186
  - not operational (I/O system state) 193
  - working (I/O system state) 193
  - (*see also* I/O devices and control units)
- DIAGNOSE instruction 103
- digit, decimal 147
- digit selector in editing 150
- direct-control facility 46
- disabling, enabling interruptions 70
- displacement (in operand designation) 20
- display-and-enter controls 245
- DIVIDE (DDR, DD, DER, DE) instruction 163
- DIVIDE (DR, D) instruction 128
  - example 296
- DIVIDE DECIMAL (DP) instruction 149
  - example 306
- doubleword (definition) 14
- doubleword-concurrent fetch 27
- dynamic address translation 57
  - addresses translated 62
  - addressing exception during 61, 68
  - control 58
  - control register 0 58
  - control register 1 59
  - exceptions 68, 81
    - page-translation exception 79, 62
    - segment-translation exception 79, 61
    - translation-specification exception 79, 62
  - formats, summary 68
  - page invalid bit 60
  - page size 59
  - page table
    - address 60
    - entry 60
    - entry fetch sequence 24
    - length 60
    - lookup 61
  - process 60
  - segment invalid bit 60
  - segment size 59
  - segment table
    - address 59
    - entry 59
    - entry fetch sequence 24
    - length 59
    - lookup 61
  - states of translation-table entries 65
  - table entries
    - active 66
    - attached 65
    - valid 65
  - tables 59
    - modification of 66
  - translation lookaside buffer 65

- EBCDIC chart (*see* Appendix H)
- EDIT AND MARK (EDMK) instruction 152
  - example 307
- EDIT (ED) instruction 150
  - example 306
- emergency-pull switch 245
- emergency signal
  - external interruption 87
  - order 97
- enable-system-clear key 245
- enabling and disabling interruptions 70
- ending of instruction execution, methods of 74
- equipment check (sense data) 220
  - status bit 99
- error
  - program, handling of 75
  - state of time-of-day clock 46
  - storage
    - corrected bit 180
    - key in storage 180
    - uncorrected bit 180
- error checking and correction (ECC), redundancy correction 172
- event mask for program event recording 40
- events, interruption causing 80
- exception conditions
  - during decimal operations 149
  - during EXECUTE operations 129
  - during fixed-point operations 116
  - during floating-point operations 161
- exceptions associated with PSW 35
- excess-64 binary notation 157
- EXCLUSIVE OR (XR, X, XI, XC) instruction 128
  - example 297
- EXECUTE (EX) instruction 129
  - example 298
  - exceptions during execution 129
- execute exception 76
- execution, program 19
- exigent machine-check interruption conditions
  - definition of 175
  - handling of (interruption action) 175, 177
- explicit (address) translation 60
- exponent
  - in a floating-point number 157
  - overflow exception 78
  - underflow exception 79
- extended control mode (EC) 32
  - PSW format 34
- extended-floating-point number 158
- extended logout (*see* machine-check extended logout)
- extended logout pointer, I/O 240
- external-call
  - external interruption 87
  - order 97
  - pending, status bit 99
- external damage, machine-check interruption condition 179, 175
  - report mask 182
- external interruption 84
  - clock comparator 87
  - CPU timer 88
  - emergency signal 87
  - external call 88
  - external signal 86
  - identification in main storage 90
  - interrupt key 86
  - interval timer 86
  - malfunction alert 86

external interruption *(continued)*
  mask (BC mode) 34
  mask (EC mode) 34
  time-of-day clock sync check 88
external signal, interruption 87
  mask bit (in control register 0) 87

failing-storage address
  in machine-check extended interruption information 177
  in machine-check interruption code validity bits 180
features, System/370 (*see* Appendix A)
fetch protection bit (*see also* storage protection) 38
fetch reference, storage operand 25
field (*see* instruction format)
field validity flags (in limited channel logout) 241
fill character 151
fixed-length operands 14
fixed logout (*see* machine-check fixed logout)
fixed-point
  divide exception 78
  exceptions 78
  number representation 116
  overflow exception 78
flag in CCW
  as defined for each type of command 217
  chain command 211
  chain data 211
  program-controlled interruption (PCI) 211
  skip 211
  suppress-length indication (SLI) 211
floating-point
  divide exception 161, 79
  instructions 157, 290
    data format 157
    examples 309
    exceptions 159, 79
  number representation 159
  register 16
  register valid bit (in machine-check interruption code) 180
  register save area (machine-check extended interruption information) 178
format
  data 14
  dynamic address translation 68
  information 14
  instruction 20
  I/O instruction 197
  summary (*see* Appendix D)
  word 14
formation of the real address 62
forming the operand address 21
fraction in floating-point operands 157
functions that differ from System/360 (*see* Appendix B)

general instructions 116
  data formats 116
  representation of fixed-point numbers 116
general-purpose design of System/370 9
general register 16
  coupled 16
  save area (machine-check extended interruption information) 178
  valid bit (machine-check interruption code) 180
general-register-alteration program event 43
guard digit, floating-point 158

halfword (definition) 14
HALT DEVICE (HDV) instruction 199
HALT I/O (HIO) instruction 202
HALVE (HDR, HER) instruction 164
handling of access exceptions 81
handling of (storage) addresses 63
hardware checkpoint (in CPU retry) 172
hardware instruction retry (*see* CPU retry)
hexadecimal tables (*see* Appendix G)

I field in an instruction 19
IDAW (indirect data address word) 216
identification of source of interruption 70
identity of storage control unit (SCU), in limited channel logout 240
immediate operand 20, 19
immediate operation (I/O) 222
IMPL (initial microprogram load) controls 245
implicit (address) translation 60
implied field length of operands 14
inadvertent resetting of sense data (*see* unit check programming note) 233
incorrect length, channel status condition 233
index (in operand designation) 20
indirect data address (IDA) 190, 216
  flag (in CCW) 211
  word (IDAW) 216
information
  formats 14
  positioning 15
initial-CPU-reset order 98
initial-microprogram-load (IMPL)
  controls 245
  order 98
initial program load (IPL) 54
initial-program-reset order 97
input/output
  address, in limited channel logout 242
  commands 217
  communications area (IOCA) 239
  device 186
    addressing 192
    attachment of 186
    error 197
  devices and control units 186
  error alert, in limited channel logout 241
  extended logout control bit (in control register 14) 182
  extended logout pointer 240
  general description 17
  instructions 197
  interface 17
  interruption 226, 73
    channel available (CAI) 227
    channel mask 88
    conditions 226
    priority of 227
    program-controlled (PCI) 233
  mask
    BC mode (*see* extended control mode) 33
    EC mode 34
  operations 185
    blocking of data 210
    chaining 213
    conclusion (termination) due to equipment malfunction 226

input/output *(continued)*
operations *(continued)*
conclusion of 222
initiation 210
termination by HALT I/O, HALT DEVICE 224
selective reset 195
system operation 189
system reset 194, 51
input/output status condition 229, 239
input/output system states 192
INSERT CHARACTER (IC) instruction 130
INSERT CHARACTERS UNDER MASK (ICM) instruction 130
INSERT PSW KEY (IPK) instruction 104
INSERT STORAGE KEY (ISK) instruction 105
instruction
address in PSW
BC mode 34
EC mode 35
validity bit 180
address, updated 22
address validity bit (machine-check interruption code) 180
B field 20
counter (instruction address portion of current PSW) 35
D field 20
decimal 147
exception handling, I/O 210
execution 22, 74
conceptual sequence (order) of 22
fetching 23
fetching program event 42
field, zero value in X or B 22
fixed-point (*see* general instructions)
floating-point 157
format 20
basic 20
I/O 197
general 115
I field 19
input/output 197
length code (ILC) in PSW
in BC mode PSW 34
meaning 71
logical (*see* general instructions)
operand 19
operation 20
privileged 30
R field 19
sets and features 9
system control 103
use examples 291
X field 20
instruction processing damage (machine-check interruption condition) 178, 175
instructions (*see* Appendix C for listings)
instructions
interruptible 73
offered by some models, but not listed in this manual 76
integral boundaries in main storage 15
interface
address validity flag 241
control check, channel status condition 235
I/O 17
interlocked update storage reference 26
interlocks between logical and real storage references 63
interlocks between storage references 64
interpretation of order code, conditions precluding 98

interrupt key 245, 86
interruption 86
mask bit (in control register 0) 86
priority of interruption 86
interruptible instructions 75
interruption (to program execution) 70
classes 70
clock comparator 47
code, BC mode PSW 34
CPU timer 48
enabling and disabling 70
external 84
general description 22
instruction length code use 71
I/O (input/output) 88
machine-check 75
new PSW 90, 23
old PSW 90, 23
point of (machine check) 176
point of (occurrence of) 74
priorities 89
program 75
program-controlled 215
purpose 70
restart 88
source identification 70
supervisor-call 84
interruption action 70
machine check 175
table 72
interruption classes
external 84
input/output 88
machine check 75
program 75
supervisor-call 84
interruption, machine check 175
conditions 175
extended information 177
interruption code 178
interruption pending (I/O system state) 192
in channel 194
in device 193
in subchannel 194
interruptions, multiple PCI (*see* programming notes) 221
interval timer 49
external interruption 49, 86
mask bit (in control register 0) 86
priority of interruption 86
updating 49
intervention required (sense data) 220
invalid address 76
invalid CBC
definition 172
handling of
in keys in storage 173
in registers 173
in storage 172
invalid (I/O programming) 234
invalid order status bit 100
I/O (*see* input/output)
I/O interface 17
IOCA (input/output communications area) 240
IOEL (input/output extended logout)
address in main storage 91

IOEL (input/output extended logout) *(continued)*
  control register bit 182
  pointer 240
IPL (initial program load) 54
IPL CCW1, CCW2, in absolute main storage 91
IPL PSW in absolute main storage 91

key in storage error uncorrected bit (machine-check interruption code) 180
key, protection, in CSW 228, 236
key, storage 38
  accesses 24

length of operand 19
  immediate operands 20
  register operands 20
  storage operands 20
limited channel logout 240
  detect field 240
  field validity flags 241
  in main storage 91
  I/O address 242
  I/O error alert 241
  sequence code 241
  source field 240
  storage control unit (SCU) identity 240
  type of termination 241
load
  indicator 245
  key 245
  state 31
  unit-address controls 246
LOAD (LDR, LD, LER, LE) instruction 165
LOAD (LR, L) instruction 130
  example 298
LOAD ADDRESS (LA) instruction 131
  example 299
LOAD AND TEST (LTDR, LTER) instruction 165
LOAD AND TEST (LTR) instruction 131
LOAD COMPLEMENT (LCDR, LCER) instruction 165
LOAD COMPLEMENT (LCR) instruction 131
LOAD CONTROL (LCTL) instruction 105
LOAD HALFWORD (LH) instruction 131
  example 299
LOAD MULTIPLE (LM) instruction 132
LOAD NEGATIVE (LNDR, LNER) instruction 166
LOAD NEGATIVE (LNR) instruction 132
LOAD POSITIVE (LPDR, LPER) instruction 166
LOAD POSITIVE (LPR) instruction 132
LOAD PSW (LPSW) instruction 105
LOAD REAL ADDRESS (LRA) instruction 106
LOAD ROUNDED (LRDR, LRER) instruction 166
loading of initial program information 54
logical storage
  address 14, 58
  address translation 58
  addressing 58
logout
  asynchronous/synchronous 177
  extended/fixed 177
  main storage, permanently assigned locations 92
  pending (LOP), in CSW 228
  (*see also* machine-check extended logout and machine-check fixed logout)
logout control 181

long block (in I/O) 233
long floating-point number 158

machine-check
  code 75, 178
  detection 171
  handling 172
machine-check control register
  bits (chart) 183
  subclass masks 182
  subclass masks summary 183
machine-check extended interruption information 177
  register save area 178
machine-check extended logout (MCEL)
  address in control register 181
  asynchronous, control of 182
  asynchronous, definition 177
  control, summary chart 183
  length in machine-check interruption code 181
  maximum length, in CPU ID 112
  synchronous, control of 181
  synchronous, definition 177
  valid bit (machine-check interruption code) 180
machine-check fixed logout
  area 177
  asynchronous, control of 182
  asynchronous, definition 177
  control, summary chart 183
  synchronous, definition 177
machine-check interruption 75
  action 175
  code 178
  code in main storage 91
  code validity bits 180
  mask, BC mode 34
  mask, EC mode 35
  point of 176
machine-check logout, synchronous/asynchronous 177
  control (chart) 183
  extended (*see* machine-check extended logout)
  fixed (*see* machine-check fixed logout)
machine-check mask
  BC mode 34
  EC mode 35
  subclass masks 183
  summary chart 183
machine-check save areas (machine-check extended interruption information) 177
machine errors, handling of 172
main storage
  accesses, sequence of 23
    actual operation 23
    conceptual operation 23
  address wraparound 14
  addressing 14
  assigned locations 90
    absolute 91
    real 90
  controlled sharing of by TEST and SET 144
  controlled sharing of by COMPARE AND SWAP 123
  general description 14
  integral boundaries 15
  power-on reset effect 53
  reference and change recording 67
  volatile 14

malfunction-alert external interruption 86, 175
malfunction (selective) reset in I/O system 195
manual indicator 246
manual operation of system (*see* system console)
mask (in BC mode PSW)
  channel 33
  external 33
  I/O 33
  machine-check 34
  program 34
masks (in EC mode PSW)
  external 34
  input/output 34
  machine check 35
  program 35
  program-event recording 34
mask bit in control of interruptions 71
mask position values used in BRANCH ON CONDITION 121
masks, monitor 39
MCEL (*see* machine-check extended logout)
microprogram controls, initial 245
mode
  BC (basic control) 32
  control bit in PSW 34
  EC (extended control) 32
  of channel operation (data transfer) 188
modifier bits in CCW command code 212
MONITOR CALL (MC) instruction 132
monitoring 39
  class number 39, 91
  code 39, 91
  interruption identification (*see* monitor class number; monitor code) 91
  masks (in control register 8) 39
MOVE (MVC, MVI) instruction 133
  example 299
MOVE LONG (MVCL) instruction 133
MOVE NUMERICS (MVN) instruction 135
  example 300
MOVE WITH OFFSET (MVO) instruction 135
  example 300
MOVE ZONES (MVZ) instruction 136
  example 300
multiple PCI interruptions (*see* programming notes) 221
multiple simultaneous interruption requests (*see* priority of interruptions) 89
multiplexer channel (*see* byte-multiplexer channel and block-multiplexer channel)
MULTIPLY (MDR, MD, MER, ME) instruction 167
MULTIPLY (MR, M) instruction 136
  example 301
MULTIPLY (MXD, MXDR) instruction 167
MULTIPLY (MXR) instruction 167
MULTIPLY DECIMAL (MP) instruction 153
  example 307
MULTIPLY HALFWORD (MH) instruction 136
  example 301
multiprocessing 95, 9
  examples 310

near valid CBC 171
no-operation control command 219
nonshared subchannel 189
nonvolatile main storage 14
normal sequential instruction execution 22
normalization in floating-point arithmetic 159
not operational (state of I/O system) 192
not ready status bit 100
nullification (method of ending instruction execution) 74
number representation
  decimal 148
  fixed-point 116
    with two's complement 289
  floating-point 159
numbering
  bits of a byte 14
  byte locations in main storage 14
numerics 147

one's complement, use of in fixed-point operations 117
op code (operation code) 20
operand field length 14
operands
  address specification of 19
  immediate 20
  in main storage 20
  in registers 20
  storage 25
    fetch reference 25
    relation between references 27
    store reference 25
    update reference 26
operating state 30
operation code (of an instruction) 20
operation exception 76
operation, I/O 186, 17
operation, unit of 74
operational (I/O state) 192
operator-intervening status bit 99
operator section (of system control panel) 243
OR (OR, O, OI, OC) instruction 137
  example 301
order
  conditions determining response to 98
  I/O (definition) 189
  signaling and response 97
order-code interpretation, conditions precluding 98
organization, system 13
overflow
  decimal 147
  exponent 158
  fixed-point (*see* ADD (AR)) 117
overlap, destructive (in MOVE LONG) 133
overlapped fields, decimal 147
overrun (sense data) 220

PACK (PACK) instruction 137
  example 302
packed decimal number 147
page 58
  address 60
  index field 58
  invalid bit 60
  size bits (in control register 0) 58
  table 60
    address 60
    entries 60
    entry format 60
    length 60
    lookup 61
page-translation exception 79
parity bit 171

pattern character 150
PCI (*see* program-controlled interruption)
PER (*see* program-event recording)
PER events
  with concurrent exceptions 45
  with dynamic address translation 42
  with interruptible instruction 42
  with LOAD PSW 41
  with SUPERVISOR CALL 40
point of interruption (interruptions) 74
point of interruption (machine-check interruption) 176
postnormalization 159
power check indication, thermal/CB 247
power-off key 246
power-on key 246
power-on reset
  CPU 53
  main storage, effect of on 53
  TOD (time-of-day) clock 53
powers of two, table (*see* Appendix F)
prefixing 95, 14
prenormalization 159
priority
  of access exceptions 84
  of clock comparator interruptions 86
  of CPU timer interruptions 86
  of exigent machine-check interruptions 89
  of external interruptions 86
  of interrupt key interruptions 86
  of interval timer interruptions 86
  of I/O interruptions 227, 89
  of program exceptions 85
  table of program exceptions 85
privileged
  instruction 30
  operation exception 76
problem state 30
processor address (stored by external interruption) 91 90
program
  check (channel status condition) 234
  execution 19
  interruption 75
    identification in main storage 90
program-controlled interruption (PCI) 215
  channel status condition 233
  flag (in CCW) 211
program-event recording (PER) 39, 80
  address in main storage 41
  code 41
  control register allocation 40
  event masks 40
  general-register-alteration masks 40
  indication of events concurrently with other interruption conditions 43
  interruption identification in main storage 90
  mask, EC mode 34
  operation 40
  starting/ending address 40
  storage area designation 42
program events 42
  general register alteration 43
  indication of 45
  instruction fetching 42
  storage (main) alteration 42
  successful branching 42
program execution 19
program-interruption conditions
  access exception, recognition of 80
  addressing exception 76
  data exception 78
  decimal-divide exception 78
  decimal-overflow exception 78
  execute exception 76
  exponent-overflow exception 78
  exponent-underflow exception 79
  fixed-point divide exception 78
  fixed-point overflow exception 78
  floating-point divide exception 79
  monitor event 80
  operation 75
  page-translation 79
  priority of 85
  privileged operation 76
  program event 80
  segment-translation 79
  significance 79
  special operation 80
  specification 77
  table of priorities of 85
  translation specification 79
program mask and condition code validity bit (in machine-check interruption code) 180
program mask in PSW
  BC mode 34
  EC mode 35
program-reset
  initial 52
  order 97
program state (*see* CPU state)
program status word (*see* PSW)
protection check (channel status condition) 234
protection exception 76
  summary table 77
protection key
  in CAW 210
  in CSW 228, 236
protection key in PSW
  BC mode 33
  EC mode 34
protection, storage (*see* storage protection)
PSW (program status word) 32, 22
PSW, BC mode format 33
  channel mask 33
  condition code 34
  current 16
  extended-control (EC) mode 34
  external mask 33
  ILC (instruction-length code) 34
  instruction address 34
  interruption code 34
  I/O mask 33
  machine-check mask 34
  problem state bit 34
  program mask 34
  protection key 34
  wait state bit 34
PSW, EC mode format 33
  condition code 35
  EC mode bit 34
  external mask 34
  instruction address 35
  I/O mask 34

PSW, EC mode format *(continued)*
    machine-check mask 35
    problem state bit 35
    program-event recording (PER) mask 34
    program mask 35
    protection key 34
    translation mode bit 34
    wait state bit 35
PSW EMWP validity bit (in machine-check interruption code) 180
PSW (IPL) in absolute main storage 91
PSW mask and key validity bit (in machine-check interruption code) 180
PSW translation control, bit 5 58
PURGE TLB (PTLB) instruction 107

R field of an instruction 20, 19
rate control 246
read backward command, I/O 218
READ DIRECT (RDD) instruction 107
real address (of a main-storage location) 57, 96, 14
    formation of 62
real-time clock, interval timer as a 49
receiver-check status bit 100
recognition of access exceptions 80
    table 82
recovery
    condition (machine-check interruption condition) 175
    mechanisms 172
    report mask (in control register 14) 182
    system (machine-check interruption condition) 178, 175
redundancy correction 172
reference and change recording (in main storage) 67
    change bit 67, 38
    reference bit 67, 38
references to storage
    block concurrent 27
    single access 26
region code
    in machine-check extended interruption information 178
    in machine-check interruption code validity bits 180
register
    control 36
    floating-point 16
    general 16
    operand 20
    save area (machine-check extended interruption information) 178
    validity bits (in machine-check interruption code) 180
remote operator control panel (ROCP) 248
repressible machine-check interruption condition
    definition of 175
    handling of (interruption action) 175
reset, I/O system 194, 51
    effect on working device 195
    upon malfunction 195
RESET REFERENCE BIT (RRB) instruction 107
resets 50
    CPU reset 51
    initial CPU reset 51
    initial program reset 52
    manual initiation of 50
    power-on reset 53
    program reset 51
    store status facility, effect on 54
    system clear reset 53
restart
    interruption 88
    key 246
    new PSW in main storage 91
    old PSW in main storage 91
    order 97
result
    character in editing 151
    condition in editing 151
retry, CPU 172
right of access to main storage 38
ROCP (remote operator control panel) 248
rounding (LRDR, LRER) instructions 166
RR instruction format 20
RS instruction format 20
running state 30
RX instruction format 20

S instruction format 20
save area (machine-check extended interruption information) 178
segment
    index field 59
    invalid bit 60
    size bits (in control register 0) 59
    table 59
    table address (in control register 1) 59
    table entry 59
    table format 59
    table length (in control register 1) 59
    table lookup 61
    translation exception 79
selective reset, I/O 195
selector channel 188
sense
    command, I/O 219
    data (in I/O) 219
    order 97
sequence of main storage accesses (references) 23
sequence code 241
    validity flag 241
sequential execution of instructions
    change in by interruption 70
    normal 22
serialization 28
SET CLOCK (SCK) instruction 108
SET CLOCK COMPARATOR (SCKC) instruction 108
SET CPU TIMER (SPT) instruction 109
SET PREFIX (SPX) instruction 109
SET PROGRAM MASK (SPM) instruction 138
SET PSW KEY FROM ADDRESS (SPKA) instruction 109
SET STORAGE KEY (SSK) instruction 110
SET SYSTEM MASK (SSM) instruction 110
set system mask suppression 32
shared main storage 95
shared subchannel 189
SHIFT AND ROUND DECIMAL (SRP) instruction 153
    example 307
SHIFT LEFT DOUBLE (SLDA) instruction 138
    example 302
SHIFT LEFT DOUBLE LOGICAL (SLDL) instruction 139
SHIFT LEFT SINGLE (SLA) instruction 139
    example 302
SHIFT LEFT SINGLE LOGICAL (SLL) instruction 139
SHIFT RIGHT DOUBLE (SRDA) instruction 140
SHIFT RIGHT DOUBLE LOGICAL (SRDL) instruction 140

SHIFT RIGHT SINGLE (SRA) instruction 140
SHIFT RIGHT SINGLE LOGICAL (SRL) instruction 141
short block (in I/O) 233
short floating-point number 158
SI instruction format 20
sign and zone codes in decimal operands 147, 148
sign change in fixed-point operations 116
SIGNAL PROCESSOR (SIGP) instruction 110
signaling and response, CPU 97
  orders 97
  status bits 99
significance
  exception 79
  indicator in editing 151
simultaneous interruption requests, multiple (*see* priority of interruptions) 89
skip (SKIP) flag (in CCW) 211
skipping (in I/O) 215
SLI (suppress-length-indication) flag in CCW 211
source digit in editing 150
source field (in limited channel logout) 240
source identification of interruptions 70
special operation exception 80
specification exception 77
SS instruction format 20
SSM suppression bit (in control register 0) 32
START I/O (SIO) instruction 204
START I/O FAST RELEASE (SIOF) instruction 204
start key 247
start order 97
states
  check-stop 31
  CPU 30
  load 31
  of I/O system 192
  of time-of-day clock 46
  operating 30
  problem 30
  program (*see* CPU states)
  running 30
  stopped 30
  supervisor 30
  wait 30
status
  bits (signal processor) 99
  conditions, I/O 229, 237
  in CSW 229, 237
  modifier (I/O unit status condition) 229
  of system (from wait, manual, and system indicators) 248
  word, program (PSW) 32
stop-and-store-status order 98
stop key 247
stop order 97
stopped
  state 30
  status bit 99
storage
  address wraparound 14
  addressing, logical 58
  alteration program event 42
  control unit (SCU) identity (in limited channel logout) 240
  key 38
  logical validity (machine-check interruption code) 181
  main (*see* main storage)
  operand 20
storage *(continued)*
  operand reference (access) 25
  validation 173
storage error
  corrected bit (machine-check interruption code) 180
  uncorrected bit (machine-check interruption code) 180
storage protection 38
  accesses protected 39
  action 38
  fetch 38
  key in storage 38
  violation (*see* protection exception)
STORE CPU ADDRESS (STAP) instruction 112
STORE PREFIX (STPX) instruction 113
store-status
  facility 54
  key 247
  save area in absolute main storage 91
STORE (ST) instruction 141
STORE (STD, STE) instruction 168
STORE CHANNEL ID (STIDC) instruction 206
STORE CHARACTER (STC) instruction 141
STORE CHARACTERS UNDER MASK (STCM) instruction 141
STORE CLOCK (STCK) instruction 141
STORE CLOCK COMPARATOR (STCKC) instruction 111
STORE CONTROL (STCTL) instruction 111
STORE CPU ID (STIDP) instruction 112
STORE CPU TIMER (STPT) instruction 113
STORE HALFWORD (STH) instruction 142
STORE MULTIPLE (STM) instruction 142
  example 302
store protection (*see* storage protection) 38
store reference, storage operand 25
store status 54
STORE THEN AND SYSTEM MASK (STNSM) instruction 113
STORE THEN OR SYSTEM MASK (STOSM) instruction 114
subchannel 188
  nonshared 189
  not operational (I/O system state) 194
  shared 189
  working (I/O system state) 194
submask, external interruption 86
SUBTRACT (SR, S) instruction 143
SUBTRACT DECIMAL (SP) instruction 154
SUBTRACT HALFWORD (SH) instruction 143
SUBTRACT LOGICAL (SLR, SL) instruction 143
SUBTRACT NORMALIZED (SER, SE, SDR, SD) instruction 169
SUBTRACT NORMALIZED (SXR) instruction 169
SUBTRACT UNNORMALIZED (SWR, SW, SUR, SU) instruction 169
successful branching program event 42
supervisor-call interruption 84
supervisor-call interruption identification in main storage 90
SUPERVISOR CALL (SVC) instruction 144
supervisor program (*see* system program) 10
supervisor state 30
suppress-length-indication (SLI) flag in CCW 211
suppression, method of ending instruction execution 74
synchronization of TOD clock 101, 47
synchronous machine check logout (*see* machine-check extended logout and machine-check fixed logout)
synchronous MCEL control bit (in control register 14) 181
system
  clear reset 53
  console 243, 18
  control 29

system *(continued)*
control instructions 103
damage (machine-check interruption condition) 178
indicator 247
operation, I/O 189
organization 13
program 10
recovery (machine-check interruption condition) 178, 175
reset, I/O 194, 51
system control panel (of the system console)
customer-engineer-control section 248
operator-control section 243
system indication 247
system-reset key 247
system status (from wait, manual, and system indicators) 248

table
manipulation 64
page 60
segment 59
translation 60
termination
method of ending instruction execution 74
of channel program 222
of I/O operations (*see* conclusion of I/O operations)
TEST AND SET (TS) instruction 144
TEST CHANNEL (TCH) instruction 207
test indicator 247
TEST I/O (TIO) instruction 208
TEST UNDER MASK (TM) instruction 145
example 303
thermal/CB power check indication 247
TIC (transfer in channel) I/O command 221
time-of-day (TOD) clock 46
key 247
power-on reset 53
states of 46
sync-check external interruption 87
sync control bit 47
synchronization of 101, 47
timeout, channel 188
timer, CPU 48
timer damage (machine-check interruption condition) 179, 175
timer, interval 49
external interruption 86
updating 49
timing facilities (*see* CPU timer, clock comparator, interval timer, and time-of-day clock)
timing facility damage (machine-check interruption condition) 179, 175
TLB (translation lookaside buffer) 64
TOD clock (*see* time-of-day clock)
TOD clock key 247
transfer in channel (TIC) I/O command 221
TRANSLATE (TR) instruction 145
example 303
TRANSLATE AND TEST (TRT) instruction 145
example 304
translation 60
exception address in main storage 79
lookaside buffer (TLB) 64
mode bit in PSW 33
process 61

translation *(continued)*
specification exception 79
table entries, states of 65
table modification 66
tables 59
(*see also* dynamic address translation)
two's complement, use of in fixed-point operations 116, 289
type of termination (in limited channel logout) 241
types of channels 188
types of (storage) addresses 62

unit
check (I/O unit status condition) 232
deletion (machine-check handling) 172
exception (I/O unit status condition) 233
of information (the byte), basic 14
of operation 74
unit status, I/O
conditions 229, 237
validity flag 241
unnormalized floating-point operation 159
UNPACK (UNPK) instruction 146
example 305
update (type of storage operand reference) 25
use of translation lookaside buffer 65

valid CBC 172
validation
of keys in storage 173
of registers 173
of storage 173
validity bits (in machine-check interruption code) 180
variable field length of operands 14
virtual
address 57
storage 57
volatile main storage 14

wait indicator 248
wait/running CPU state 30
bit in PSW, BC mode 34
bit in PSW, EC mode 35
warning
machine-check interruption condition 179, 175
mask bit 182
word (definition) 14
word-concurrent fetch 27
working (state of I/O system) 192
working device, effect of reset on 195
wraparound of main storage addressing 14
wraparound of register addresses in LOAD MULTIPLE 132
write command, I/O 218
WRITE DIRECT (WRD) instruction 114

X field of an instruction 20

ZERO AND ADD (ZAP) instruction 155
example 309
zone and sign codes in decimal operands 147, 148

IBM Technical Newsletter

This Newsletter No.	GN22-0498
Date	September 1, 1975
Base Publication No.	GA22-7000-4
File No.	S/370-01
Previous Newsletters	None

IBM System/370 Principles of Operation

This Technical Newsletter provides replacement pages for the subject publication.
Pages to be inserted and/or removed are:

Title Page, Back of Title Page
Preface, Back of Preface
v through viii
15, 16
47, 48
59 (text rearrangement only), 60
63 through 66 (text rearrangement only)
67 (text rearrangement only), 68
69, 70
81, 82 (text rearrangment only, both pages)
83 (text rearrangement only), 84
87, 88 (text rearrangement only)
89, 90 (text rearrangement only)
91 (text rearrangement only), 92
93, blank
95, 96
119 through 122
125, 126 (text rearrangement only)
129, 130
133, 134
145, 146
163, 164
175 (text rearrangement only), 176
183, blank
205, 206
227, 228
241, 242 (text rearrangement only)
249, 250 (text rearrangement only)
271 through 274
311 through 326

All pages to which a change has been made carry a revision notice in the upper margin. A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

This newsletter corrects technical errors appearing in text and in figures, clarifies certain ambiguous presentations, improves figure placement, and eliminates a number of typographical errors.

Note: *Please file this cover letter at the back of the manual to provide a record of changes.*

IBM Corporation, Product Publications, P.O. Box 390, Poughkeepsie, N.Y. 12602

Printed in U.S.A.

Cut or Fold Along Line

IBM System/370
Principles of Operation
Order No. GA22-7000-4

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.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

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:

____

____

____

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.)

Cut or Fold Along Line

Reader's Comment Form

Fold   Fold

FIRST CLASS
PERMIT NO. 419
POUGHKEEPSIE, N.Y.

Business Reply Mail
No postage stamp necessary if mailed in U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department B98
P.O. Box 390
Poughkeepsie, New York 12602

Fold   Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)