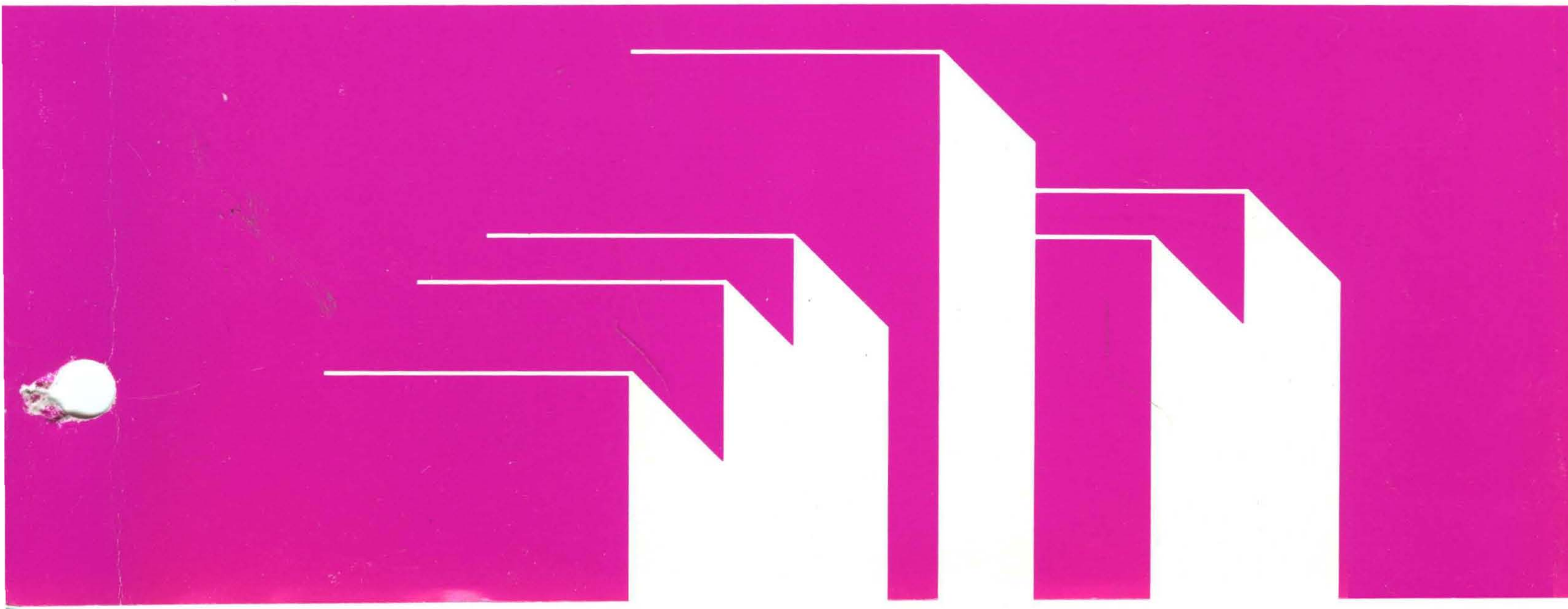
Publication Number
GA22-7000-10

IBM System/370

Principles of Operation



IBM System/370

Principles of Operation

Publication Number
GA22-7000-10

File Number
S370-01

Eleventh Edition (September 1987)

This major revision obsoletes and replaces GA22-7000-8, GA22-7000-9, and Technical Newsletters GN22-0644 and GN22-0683. Significant changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Changes are made occasionally to the information herein; before using this publication in connection with the operation of IBM equipment, refer to the latest IBM System/370, 30xx, and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

IBM may have patents or pending patent applications covering subject matter described herein. Furnishing this publication does not constitute or imply a grant of any license under any patents, patent applications, trademarks, copyrights, or other rights of IBM or of any third party, or any right to refer to IBM in any advertising or other promotional or marketing activities. IBM assumes no responsibility for any infringement of patents or other rights that may result from the use of this publication or from the manufacture, use, lease, or sale of apparatus described herein.

Licenses under IBM's utility patents are available on reasonable and nondiscriminatory terms and conditions. Inquiries relative to licensing should be directed, in writing, to: IBM Corporation, Director of Contracts and Licensing, Armonk, NY, USA 10504.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

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

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to: IBM Corporation, Central Systems Architecture, Department E57, PO Box 390, Poughkeepsie, NY, USA 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

PREFACE

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

The publication applies only to systems operating in the System/370 mode. The IBM 370-XA Principles of Operation, SA22-7085, should be consulted regarding the functions of the architecture which apply to systems operating in the 370-XA mode, and the IBM 4300 Processors Principles of Operation for ECPS:VSE Mode, GA22-7070, should be consulted regarding the functions of the architecture which apply only to systems operating in the VSE mode.

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

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

This publication is written as a reference and should not be considered an introduction or a textbook. It assumes the user has a basic knowledge of data-processing systems. IBM publications relating to System/370 are listed and described in the IBM System/370, 30xx, and 4300 Processors Bibliography, GC20-0001.

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

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

The information presented in this publication is grouped in 13 chapters and several appendixes:

Chapter 1, Introduction, highlights some of the major facilities of System/370.

Chapter 2, Organization, describes the major groupings within the system -- the central processing unit (CPU), storage, and input/output -- with some attention given to the composition and characteristics of those groupings.

Chapter 3, Storage, explains the information formats, the addressing of storage, and the facilities for storage protection. It also deals with dynamic address translation (DAT), which, coupled with special programming support, makes the use of a virtual storage possible in System/370. Dynamic address translation eliminates the need to assign a program to a fixed location in real storage and thus reduces the addressing constraints on system and problem programs.

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

Chapter 5, Program Execution, explains the role of instructions in program execution, looks in detail at instruction formats, and describes briefly the use of the program-status word (PSW), of branching, and of interruptions. It contains the principal description of the dual-address-space (DAS) facility. It also details the aspects of program execution on one CPU as observed by other CPUs and by channels.

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

Chapter 7, General Instructions, contains detailed descriptions of logical and binary-integer data formats and of all unprivileged instructions except the decimal and floating-point instructions.

Chapter 8, Decimal Instructions, describes in detail decimal data formats and the decimal instructions. The decimal instructions are a part of the commercial instruction set.

Chapter 9, Floating-Point Instructions, contains detailed descriptions of floating-point data formats and the instructions provided by the floating-point facility and by the extended-precision floating-point facility.

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

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

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

Chapter 13, Input/Output Operations, explains the programmed control of I/O devices by CPUs and by channels. It includes detailed descriptions of the I/O instructions, channel-command words, and other I/O-control formats.

The Appendixes include:

- Information about number representation
- Instruction-use examples
- Lists of the instructions arranged in several sequences
- A summary of the condition-code settings
- A list of the System/370 facilities
- A table of the powers of 2
- Tabular information helpful in dealing with hexadecimal numbers
- An EBCDIC chart
- A discussion of changes affecting compatibility between System/360 and System/370
- A discussion of changes affecting compatibility within System/370

SIZE NOTATION

In this publication, the letters K and M denote the multipliers 2^{10} and 2^{20}, respectively. Although the letters are borrowed from the decimal system and stand for kilo (10^3) and mega (10^6), they do not have the decimal meaning but instead represent the power of 2 closest to the corresponding power of 10. Their meaning in this publication is as follows:

Symbol	Value
K (kilo)	1,024 = 2^{10}
M (mega)	1,048,576 = 2^{20}

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

2,048 is expressed as 2K.
4,096 is expressed as 4K.
65,536 is expressed as 64K (not 65K).
2^{24} is expressed as 16M.

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

BYTES, CHARACTERS, AND CODES

Although the System/360 architecture was originally designed to support the Extended Binary-Coded-Decimal Interchange Code (EBCDIC), the instructions and data formats of the architecture are for the most part independent of the external code which is to be processed by the machine. For most instructions, all 256 possible combinations of bit patterns for a particular byte can be processed, independent of the character which the bit pattern is intended to represent. For instructions which use the zoned format, and for those few instructions which are dependent on a particular external code, the instruction TRANSLATE may be used to convert data from one code to another code. Thus, a machine operating in the System/370 mode can process EBCDIC, ASCII, or any other code which can be represented in eight or fewer bits per character.

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

OTHER PUBLICATIONS

The channel-to-channel adapter is described in the publication _IBM Channel-to-Channel Adapter_, SA22-7091.

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

Mathematical assists are described in the publication _IBM System/370 Mathematical Assists_, SA22-7094, which describes the instructions ARCTANGENT, COMMON LOGARITHM, COSINE, EXPONENTIAL, MULTIPLY AND ADD, NATURAL LOGARITHM, RAISE TO POWER, SINE, and SQUARE ROOT.

Vector operations are described in the publication _IBM System/370 Vector Operations_, SA22-7125.

This page is intentionally left blank.

CHAPTER 1. INTRODUCTION

This publication describes the IBM System/370 architecture.

The architecture of a system defines its attributes as seen by the programmer, that is, the conceptual structure and functional behavior of the machine, as distinct from the organization of the data flow, the logical design, the physical design, and the performance of any particular implementation. Several dissimilar machine implementations may conform to a single architecture. When the execution of programs on different machine implementations produces the results that are defined by a single architecture, the implementations are considered to be compatible.

System/370 is a product of the experience gained in developing and using several generations of compatible general-purpose systems, starting with System/360 as a base. System/370 incorporates a number of significant facilities, which are described below.

- Dynamic address translation (DAT) is a facility that eliminates the need to assign a program to fixed locations in real storage and thus reduces the addressing constraints on both control programs and problem programs, providing greater freedom in program design. Dynamic address translation permits a more efficient and effective utilization of main storage. When one of the operating systems for virtual storage is used, dynamic address translation allows the use of up to 16,777,216 bytes of virtual storage. Two page sizes (2K and 4K bytes) and two segment sizes (64K and 1M bytes) are provided, although some models offer only the 64K-byte-segment size and some models offer only the 4K-byte-page size. Extensions to this facility include the common-segment bit, the use of which increases the effective size of the translation-lookaside buffer and thus improves CPU performance, and the instruction INVALIDATE PAGE TABLE ENTRY, which improves CPU performance in a demand-paging environment.

- Protection facilities include a storage key which is standard on all models. On some models this is extended by low-address protection, the use of which increases the protection of storage locations at effective addresses 0 through 511, which are vital to the control program. Segment protection, which is available on some models, provides a segment-protection bit in the segment-table entry. When the bit is one, an attempt to store in the segment causes a protection exception to be recognized.

- Extended real addressing, which is an extension to dynamic address translation, provides the CPU with the capability of addressing up to 64M bytes of real storage. This is accomplished by the use of bits 13 and 14 of the page-table entry, which serve as the high-order bits of the page-frame real address when 4K-byte pages are specified. The larger address size applies to the real address provided by dynamic address translation and to the address provided by the LOAD REAL ADDRESS instruction.

- 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 main storage. In the basic form the indirect-data-address word contains a 24-bit address. This becomes a 31-bit address when the 31-bit-IDAW facility is installed.

- 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 4K bytes of real storage for each CPU.

- Channel-set switching permits the collection of channels in a channel set to be connected to any CPU in a multiprocessing configuration.

- Timing facilities include a TOD clock, a clock comparator, and a CPU timer, along with an interval timer that is also available in System/360. The TOD 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 TOD 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 addition, subtraction, and multiplication of floating-point numbers with a fraction of 28 hexadecimal digits. Also included 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.

- The dual-address-space (DAS) facility provides for the support of semiprivileged programs, which are executed in the problem state but which, when allowed by authorization controls, are also permitted to use additional capabilities previously available only through the assistance of supervisor-state programs. The capabilities include (1) a PSW-key mask that controls the PSW keys which can be set by the program, (2) a second address space, called the secondary address space, together with an address-space-control bit in the PSW that permits the program to switch between the primary and secondary address spaces, and (3) a table-based linkage mechanism which permits a program with one authority to call a program with greater authority.

- Start-I/O-fast queuing permits a subchannel to accept an SIOF function even when certain I/O-busy conditions are encountered. If accepted, the SIOF function is held pending until the required facilities are available. An SIOF function is initiated when a START I/O FAST RELEASE instruction is executed and other necessary conditions exist. Start-I/O-fast queuing may be provided for one or more subchannels of a channel.

- The suspend-and-resume facility provides a means for programmed control of the progress of channel-program execution. A flag bit is provided in the channel-command word (CCW) which indicates that channel-program execution is to be suspended by the channel prior to executing the CCW. A new bit is added to the channel-status word which indicates that a channel program has been suspended. A control bit is provided in the channel-address word (CAW) which indicates that the suspend function is permitted for the channel program. An instruction, RESUME I/O, causes a suspended channel program to be resumed. The suspend-and-resume facility may be provided for one or more subchannels of a multiplexer channel.

GENERAL-PURPOSE DESIGN

System/370 is a general-purpose system that can readily be tailored for a variety of applications. A commercial instruction set provides the basic processing capabilities of the system. If the floating-point facility is installed with the commercial instruction set, a universal instruction set is obtained. Adding other facilities, such as the extended-precision floating-point facility or the conditional-swapping facility, extends the processing capabilities of the system still further.

System/370 has the capability of addressing a main storage of up to 64M bytes. The System/370 dynamic-address-translation facility, used with appropriate programming support, can provide each user with an address space of 16M bytes independent of the amount of main storage. The dual-address-space facility extends this by providing each user with multiple address spaces. This facility and this support permit a System/370 model with limited main storage to be used for a much wider set of applications, and they make many applications with requirements for extensive storage practical and convenient.

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 a selector channel and two types of multiplexing channels. System/370 has a byte-multiplexer channel for the attachment of unbuffered devices and of a large number of communications devices. Additionally, it 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

COMPATIBILITY AMONG SYSTEM/370 MODELS

Although systems operating in the System/370 mode may differ in implementation and physical capabilities, logically they are upward and downward compatible. Compatibility provides for simplicity in education, availability of system backup, and ease in system growth. Specifically, any program written for the System/370 mode gives identical results on any system operating in that mode, provided that the program:

1. Is not time-dependent.

2. Does not depend on system facilities (such as storage capacity, I/O equipment, or optional facilities) being present when the facilities are not included in the configuration.

3. Does not depend on system facilities being absent when the facilities are included in the configuration. For example, the program must not depend on interruptions caused by the use of operation codes or command codes that are not installed in some models. Also, it must not use or depend on fields associated with uninstalled facilities. For example, data should not be placed in an area used by another model for logout. Similarly, the program must not use or depend on unassigned fields in machine formats (control registers, instruction formats, etc.) that are not explicitly made available for program use.

4. Does not depend on results or functions that are defined to be unpredictable or model-dependent. This includes the requirement that the program should not depend on the assignment of I/O addresses and CPU addresses.

5. Does not depend on results or functions that are defined in the functional-characteristics publication for a particular model to be deviations from the architecture.

6. Takes into account those changes made to the original System/370 architectural definition that affect compatibility among System/370 models. These changes are described in Appendix I.

COMPATIBILITY BETWEEN SYSTEM/360 AND SYSTEM/370

System/370 is forward-compatible from System/360. A program written for the System/360 operates on the System/370, provided that the program:

1. Complies with the limitations described in the section "Compatibility among System/370 Models."

2. Does not use PSW bit 12 as an ASCII bit (a special case of the second rule in the section "Compatibility among System/370 Models").

3. Does not use or depend on storage locations assigned specifically for System/370, such as the interruption-code areas, the machine-check save areas, and the extended-logout area (a special case of the third rule in the section "Compatibility among System/370 Models").

4. Takes into account other changes made to the System/360 architectural definition that affect compatibility between System/360 and System/370. These changes are described in Appendix H.

Programming Note

This publication assigns meanings to various operation codes, to bit positions in instructions, channel-command words, registers, and table entries, and to fixed locations in the low 512 bytes of storage. Unless specifically noted, the remaining operation codes, bit positions, and low-storage locations are reserved for future assignment to new facilities and other extensions of the architecture. (In addition to fixed locations in the low 512 bytes, logical location 795 is assigned to a specific function.)

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

SYSTEM PROGRAM

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

AVAILABILITY

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

Several design aspects make this possible.

- A program is checked for the correctness of instructions and data as the program is executed, and program errors are indicated separate from equipment errors. Such checking and reporting assists in locating failures and isolating effects.

- The protection facilities, in conjunction with dynamic address translation, permit the protection of the contents of storage from destruction or misuse caused by erroneous or unauthorized storing or fetching by a program. This provides increased security for the user, thus permitting applications with different security requirements to be processed concurrently with other applications.

- Dynamic address translation allows isolation of one application from another, still permitting them to share common resources. Also, it permits the implementation of virtual machines, which may be used in the design and testing of new versions of operating systems along with the concurrent processing of application programs. Additionally, it provides for the concurrent operation of incompatible operating systems.

- Multiprocessing and channel-set switching permit better use of storage and processing capabilities, more direct communication between CPUs, and duplication of resources, thus aiding in the continuation of system operation in the event of machine failures.

- MONITOR CALL, program-event recording, and the timing facilities permit the testing and debugging of programs without manual intervention and with little effect on the concurrent processing of other programs.

- Emulation is performed under control-program supervision, thus making it possible to perform emulation concurrently with other applications.

- On most models, error checking and correction (ECC) in main storage, CPU retry, and command retry provide for circumventing intermittent equipment malfunctions, thus reducing the number of equipment failures.

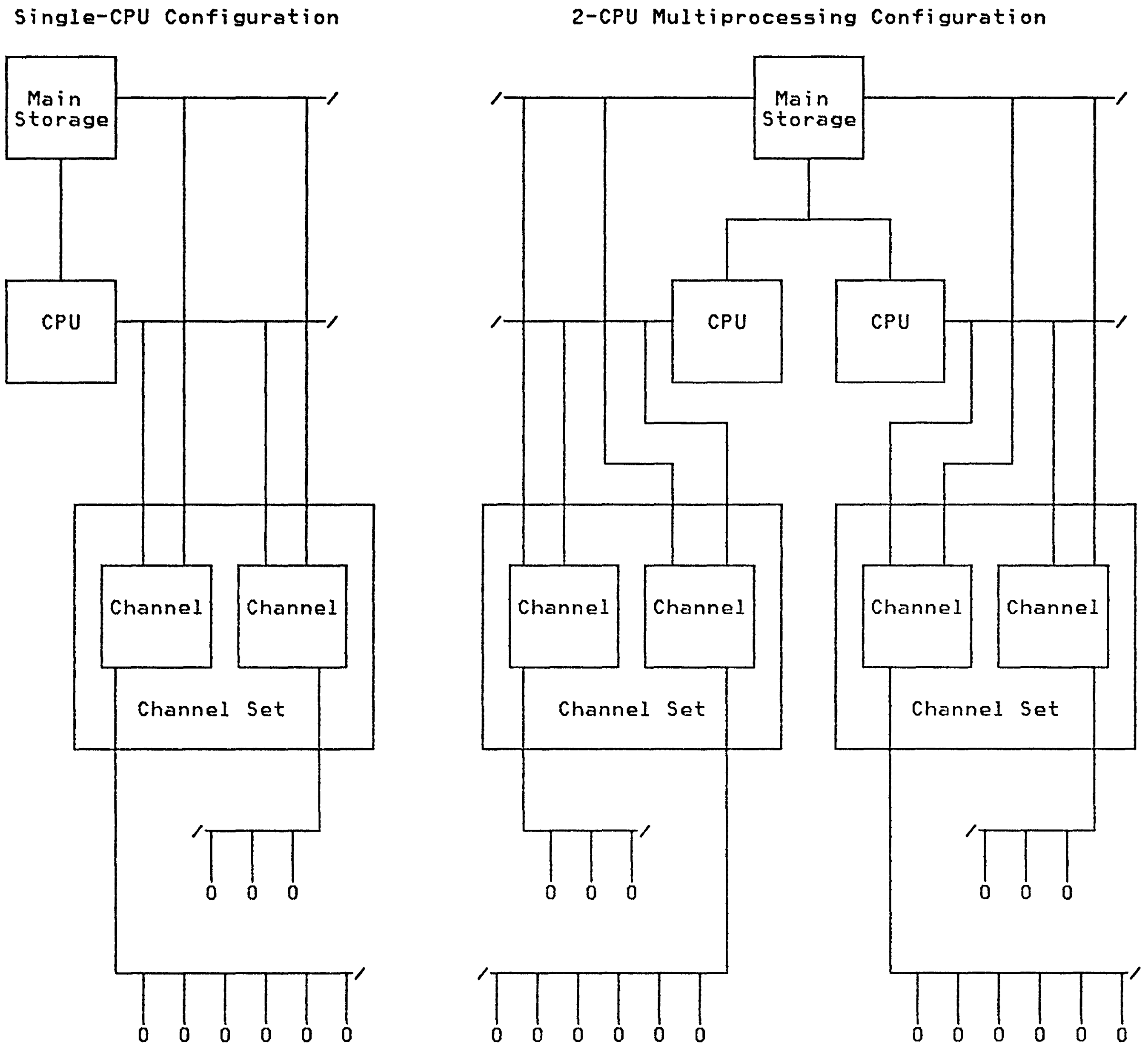
- An enhanced machine-check handling mechanism provides model-independent fault isolation, which reduces the number of programs impacted by uncorrected errors. Additionally, it provides model-independent recording of machine-status information. This leads to greater machine-check handling compatibility between models and improves the capability for loading and operating a program on a different model when a system failure occurs.

- A small number of manual controls are required for basic system operation, permitting most operator-system interaction to take place _via_ a unit operating as an I/O device and thus reducing the possibility of operator errors.

CHAPTER 2. ORGANIZATION

Logically, System/370 consists of main storage, one or more central processing units (CPUs), operator facilities, channel sets, channels, and I/O devices. I/O devices are attached to channels through control units. The physical identity of these functions may vary among implementations, called "models." The figure "Logical Structure" depicts the logical structure for a single-CPU system and for a two-CPU multiprocessing system.

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


Single-CPU Configuration

2-CPU Multiprocessing Configuration

Logical Structure

A system viewed without regard to its I/O devices is referred to as a configuration. All of the physical equipment, whether in the configuration or not, is referred to as the installation. Model-dependent reconfiguration controls may be provided to change the amount of main storage and the number of CPUs, channels, and channel sets in the configuration. In some instances, the reconfiguration controls may be used to partition a single configuration into multiple configurations. Each of the configurations so reconfigured has the same structure, that is, main storage, one or more CPUs, and channels. Each configuration is isolated in that the main storage in one configuration is not directly addressable by the CPUs and channels in another configuration. It is, however, possible for one configuration to communicate with another by means of shared I/O devices, the direct-control facility, or a channel-to-channel adapter. At any one time, the storage, CPUs, channel sets, and channels connected together in a system are referred to as being in the configuration. Channel sets and the associated channels are considered to be in a particular configuration as long as they are connected to main storage independent of whether or not the channel set is connected to a CPU in the configuration. Each CPU, channel set, channel, and main-storage location can be in only one configuration at a time.

MAIN STORAGE

Main storage, which is directly addressable, provides for high-speed processing of data by the CPUs and channels. Both data and programs must be loaded into main storage from input devices before they can be processed. The amount of main storage available on the system depends on the model, and, depending on the model, the amount in the configuration may be under control of model-dependent configuration controls. The storage is available in multiples of 2K-byte blocks. When either TEST BLOCK or the storage-key 4K-byte-block facility is installed, storage is available in multiples of 4K-byte blocks. At any instant in time, all CPUs and all channels in the configuration have access to the same blocks of storage and refer to a particular block of main-storage locations by using the same absolute address.

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

CPU

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

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

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

Instructions which the CPU executes fall into five classes: general, decimal, floating-point, control, and I/O instructions. The general instructions are used in performing binary integer arithmetic operations and logical, branching, and other nonarithmetic operations. The decimal instructions operate on data in the decimal format, and the floating-point instructions on data in the floating-point format. The privileged control instructions and the I/O instructions can be executed only when the CPU is in the supervisor state; the semiprivileged control instructions can be executed in the problem state, subject to the appropriate authorization mechanisms.

To perform its functions, the CPU may use a certain amount of internal storage. Although this internal storage may use the same physical storage medium as main storage, it is not considered part of main storage and is not addressable by programs.

The CPU provides registers which are available to programs but do not have addressable representations in main storage. They include the current program-status word (PSW), the general registers, the floating-point registers, the control registers, the prefix register, and the registers for the clock comparator and the CPU timer. Each CPU in an installation provides access to a time-of-day (TOD) clock, which may be local to that CPU or shared with other CPUs in the installation. The instruction operation code determines which type of register is to be used in an operation. See the figure "General, Floating-Point, and Control Registers" later in this chapter for the format of those registers.

PSW

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

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

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

GENERAL REGISTERS

Instructions may designate information in one or more of 16 general registers. The general registers may be used as base-address registers and index registers in address arithmetic and as accumulators in general arithmetic and logical operations. Each register contains 32 bits. The general registers are identified by the numbers 0-15 and are designated by a four-bit R field in an instruction. Some instructions provide for addressing multiple general registers by having several R fields. For some instructions, the use of a specific general register is implied rather than explicitly designated by an R field of the instruction.

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

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

FLOATING-POINT REGISTERS

Four floating-point registers are available for floating-point operations. They are identified by the numbers 0, 2, 4, and 6 and are designated by a four-bit R field in floating-point instructions. Each floating-point register is 64 bits long and can contain either a short (32-bit) or a long (64-bit) floating-point operand. A short operand occupies the leftmost bit positions of a floating-point register. The rightmost portion of the register is ignored in operations that use short operands and remains unchanged in operations that produce short results. Two pairs of adjacent floating-point registers can be used for extended operands: registers 0 and 2, and registers 4 and 6. Each of these pairs, identified by the numbers 0 and 4, provides for a 128-bit format.

CONTROL REGISTERS

The CPU makes provisions for 16 control registers, each having 32 bit positions. The bit positions in the registers are assigned to particular facilities in the system, such as program-event recording, and are used either to specify that an operation can take place or to furnish special information required by the facility.

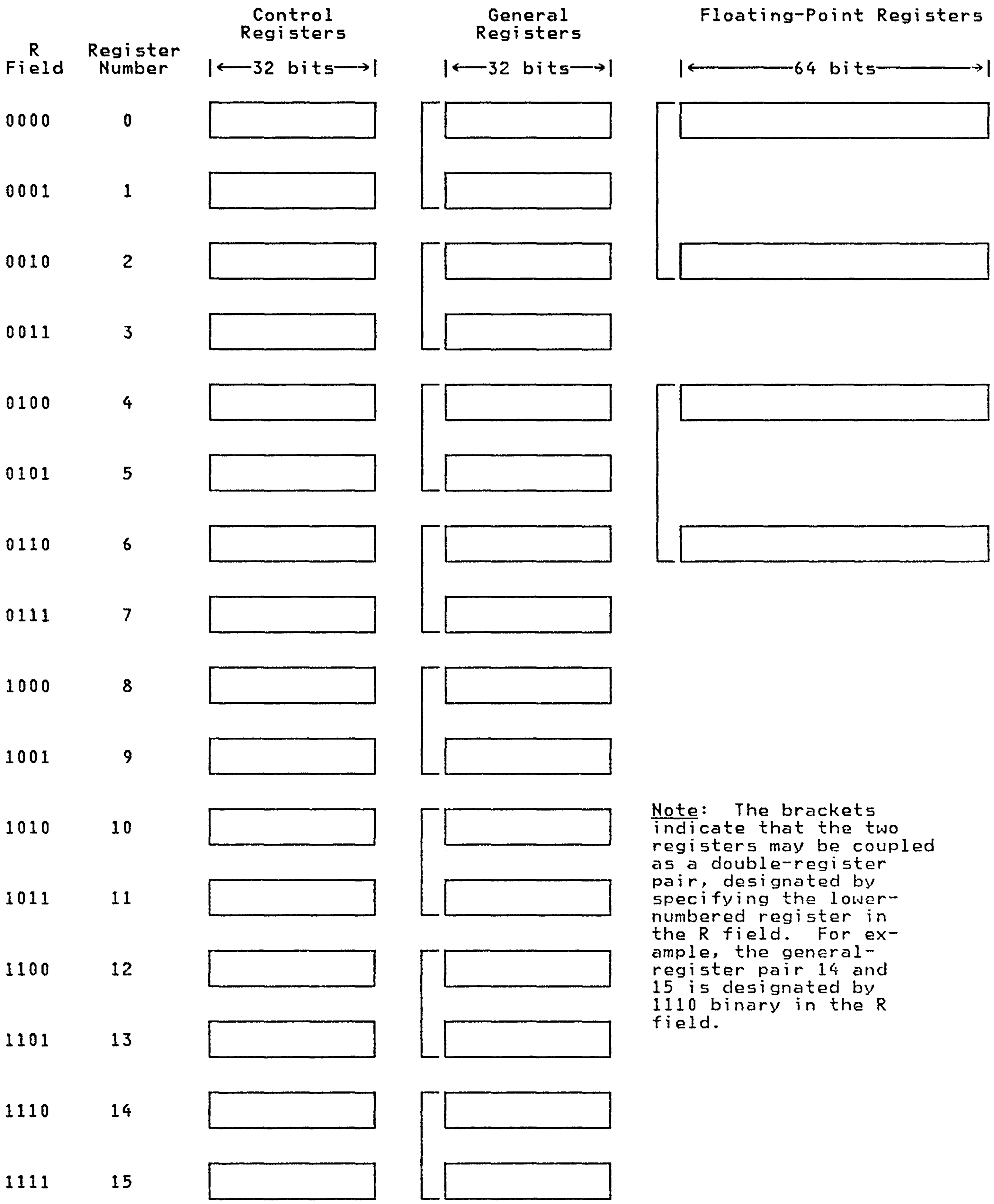
The control registers are identified by the numbers 0-15 and are designated by four-bit R fields in the instructions LOAD CONTROL and STORE CONTROL. Multiple control registers can be addressed by these instructions.

VECTOR FACILITY

Depending on the model, a vector facility may be provided as an extension of the CPU. When the vector facility is provided on a CPU, it functions as an integral part of that CPU. The functions of the vector facility and its registers are described in the publication _IBM System/370 Vector Operations_, SA22-7125.

_I/O_

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



Note: The brackets indicate that the two registers may be coupled as a double-register pair, designated by specifying the lower-numbered register in the R field. For example, the general-register pair 14 and 15 is designated by 1110 binary in the R field.

General, Floating-Point, and Control Registers

CHANNEL SETS

The group of channels which connects to a particular CPU is called a channel set. When channel-set switching is installed in a multiprocessing configuration, the program can control which CPU is connected to a particular channel set. A CPU can be connected to no more than one channel set at a time, and a channel set can be connected to no more than one CPU at a time. When channel-set switching is not installed, the channel sets, in the absence of model-dependent reconfiguration controls, are permanently connected to a single CPU.

CHANNELS

A channel relieves the CPU of the burden of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O operations. A channel is connected with main storage, with control units, and, unless it is a member of a disconnected channel set, with a CPU.

A channel may be an independent unit, complete with the necessary logical and internal-storage capabilities, or it may time-share CPU facilities and be physically integrated with the CPU. In either case, the functions performed by a channel are identical. The maximum data-transfer rate may differ, however, depending on the implementation.

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

I/O DEVICES AND CONTROL UNITS

I/O devices include such equipment as card readers and punches, magnetic-tape units, direct-access storage, displays, keyboards, printers, teleprocessing devices, communications controllers, and sensor-based equipment. Many I/O devices function with an external medium, such as punched cards or magnetic tape. Some I/O devices handle only electrical signals, such as those found in sensor-based networks. In either case, I/O-device operation is regulated by a control unit. 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. The control-unit function may be housed with the I/O device or in the CPU, or a separate control unit may be used.

OPERATOR FACILITIES

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

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

This chapter discusses the representation of information in main storage, as well as addressing, protection, and reference and change recording. The aspects of addressing which are covered include the format of addresses, the concept of address spaces, the various types of addresses, and the manner in which one type of address is translated to another type of address. A list of permanently assigned storage locations appears at the end of the chapter.

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

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

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

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

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

STORAGE ADDRESSING

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

Each byte location in storage is identified by a unique nonnegative integer, which is the address of that byte location or, simply, the byte address. Adjacent byte locations have consecutive addresses, starting with 0 on the left and proceeding in a left-to-right sequence. With the exception of those facilities described in "Storage Addressing with Extended Address Fields" below, addresses are 24-bit unsigned binary integers, which provide 16,777,216 (2^{24} or 16M) byte addresses.

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

When, during address generation, an address is obtained that exceeds $2^{24} - 1$, the carry out of the leftmost bit position of the address is ignored. This handling of an address of excessive size is called wraparound.

The effect of wraparound is to make the sequence of addresses appear circular; that is, address 0 appears to follow the maximum byte address, 16,777,215. Address arithmetic and wraparound occur before transformation, if any, of the address by dynamic address translation or prefixing. With a 16M-byte storage, information may be located partially in the last and partially in the first locations of storage and is processed without any special indication of crossing the maximum-address boundary.

Some channels do not perform address wraparound. Depending on the model, a program check may be generated if an address generated by the channel to fetch a CCW, to fetch an IDAW, or to transfer data is incremented past 16,777,215 or decremented past 0.

Storage Addressing with Extended Address Fields

Extended real addressing, 31-bit IDAWs, the instructions associated with storage-key-instruction extensions, and TEST BLOCK all provide for addresses which are more than 24 bits. This section describes the handling of these addresses.

Extended real addressing provides a 26-bit page-frame real address in the page-table entry for 4K-byte pages. This address is not subject to wraparound because the page-frame real address designates a 4K-byte block. Also provided is a 31-bit failing-storage address for certain machine-check interruptions, and a 26-bit address (extended to 32 bits with zeros on the left) as a result of LOAD REAL ADDRESS.

The 31-bit IDAWs provide a 31-bit absolute address of the I/O data area. This address is not subject to wraparound because all bytes designated by an IDAW must lie within a 2K-byte block.

The instructions INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY EXTENDED, and TEST BLOCK specify 31-bit real addresses. These addresses are not subject to wraparound since they designate a 4K-byte block.

INFORMATION FORMATS

Information is transmitted between storage and a CPU or a channel one byte, or a group of bytes, at a time. Unless otherwise specified, 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 specified by the operation to be performed. When used in a CPU operation, a group of bytes is called a field.

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

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

The bits in an address are numbered 8 through 31 for 24-bit addresses and 1 through 31 for 31-bit addresses. Within any other fixed-length format of multiple bytes, the bits making up the format are consecutively numbered starting from 0.

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

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

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

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

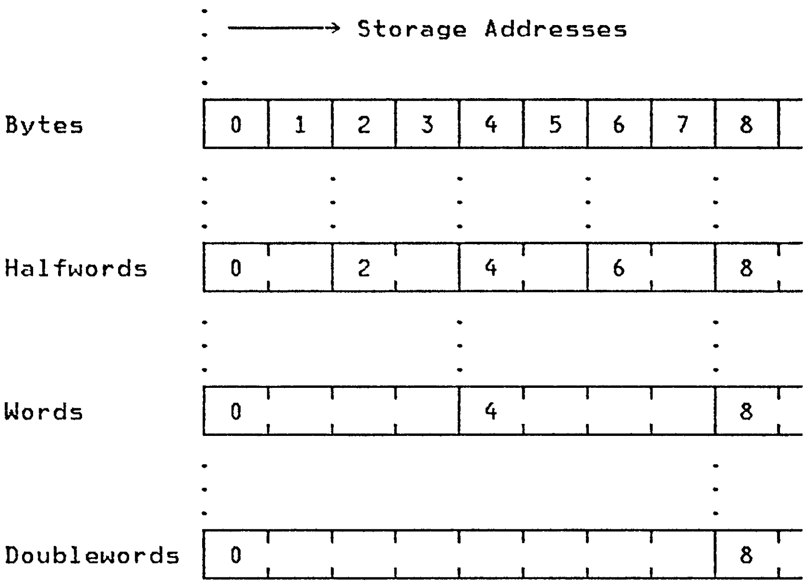
INTEGRAL BOUNDARIES

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

"Integral Boundaries with Storage Addresses.")

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

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



Integral Boundaries with Storage Addresses

BYTE-ORIENTED-OPERAND FACILITY

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

The facility does not pertain to instruction addresses or to the operands for COMPARE AND SWAP and COMPARE DOUBLE AND SWAP. Instructions must appear on two-byte integral boundaries. The rightmost bit of a branch address must be zero, and the instruction EXECUTE must designate the target instruction at an even byte address. COMPARE AND SWAP must designate a four-byte integral boundary, and COMPARE DOUBLE AND SWAP must designate an eight-byte integral boundary.

Programming Note

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

ADDRESS TYPES

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

Absolute Address

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

All CPUs and channels in the configuration refer to a shared main-storage location by using the same absolute address. Available main storage is usually assigned contiguous absolute addresses starting at 0, and the addresses are always assigned in complete 2K-byte blocks on integral boundaries. When either TEST BLOCK or the storage-key 4K-byte-block facility is installed, storage is assigned in complete 4K-byte blocks on integral boundaries. An exception is recognized when an attempt is made to use an absolute address in a block which has not been assigned to physical locations. On some models, storage-reconfiguration controls may be provided which permit the operator to change the correspondence between absolute addresses and physical locations. However, at any one time, a physical location is not associated with more than one absolute address.

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

Real Address

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

At any instant there is one real-address to absolute-address mapping for each CPU in the configuration. When a real address is used by a CPU to access main storage, it is converted to an absolute address by prefixing. The particular transformation is defined by the value in the prefix register for the CPU.

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

Virtual Address

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

Primary Virtual Address

A primary virtual address is a virtual address which is to be translated by means of the primary segment-table designation. When DAS is not installed, all logical addresses are treated as primary virtual addresses when DAT is on. When DAS is installed, logical addresses and instruction addresses are treated as primary virtual addresses when in the primary-space mode. The first-operand address of MOVE TO PRIMARY and the second-operand address of MOVE TO SECONDARY are always treated as primary virtual addresses.

Secondary Virtual Address

A secondary virtual address is a virtual address which is to be translated by means of the secondary segment-table designation. When DAS is not installed, secondary virtual addresses do not occur. When DAS is installed, logical addresses are treated as secondary virtual addresses when in the secondary-space mode. The second-operand address of MOVE TO PRIMARY and the first-operand address of MOVE TO SECONDARY are always treated as secondary virtual addresses.

Logical Address

Except where otherwise specified, the storage-operand addresses for most instructions are logical addresses. When DAS is not installed, logical addresses are treated as real addresses when DAT is off and as virtual addresses when DAT is on. When DAS is installed, logical addresses are treated as real addresses in the real mode, treated as primary virtual addresses in the primary-space mode, and treated as secondary virtual addresses in the secondary-space mode. Some instructions have storage-operand addresses or storage accesses associated with the instruction which do not follow the rules for logical addresses. In all such cases, the instruction definition contains a definition of the type of address.

Instruction Address

Addresses used to fetch instructions from storage are called instruction addresses. When DAS is not installed, instruction addresses are the same as logical addresses. When DAS is installed, instruction addresses are treated as real addresses in the real mode, treated as primary virtual addresses in the primary-space mode, and treated as either primary virtual addresses or secondary virtual addresses in the secondary-space mode. The instruction address in the current PSW and the target address of EXECUTE are instruction addresses.

Note: When the CPU is in the secondary-space mode, it is unpredictable whether instructions, including the target of EXECUTE, are fetched from the primary address space or the secondary address space. For details, see the section "Translation Modes" and the associated programming notes under the section "Dynamic Address Translation" in this chapter.

Effective Address

In some situations, it is convenient to use the term "effective address." An effective address is the address which results from address arithmetic, before address translation, if any, is performed. Address arithmetic is the addition of the base and displacement or of the base, index, and displacement.

STORAGE KEY

A storage key is associated with each 2K-byte block of storage that is available in the configuration. When the storage-key 4K-byte-block facility is installed, all of the storage keys are associated with a 4K-byte block. The storage key has the following format:

ACC	F	R	C
0	4		6

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

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

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

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

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

Storage keys are not part of addressable storage. Provided that the storage-key 4K-byte-block facility is not installed, a storage key is associated with each 2K-byte block of storage. The entire storage key is set by SET STORAGE KEY and 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. Bits 0-4 of the storage key are inspected by the INSERT VIRTUAL STORAGE KEY instruction. The contents of the storage key are unpredictable during and after the execution of the usability test of the TEST BLOCK instruction.

STORAGE-KEY 4K-BYTE-BLOCK FACILITY

Depending on whether the storage-key 4K-byte-block facility is installed, one or two storage keys are associated with each 4K-byte block of storage that is in the configuration. The storage-key-exception control is also provided as part of this facility.

Storage Keys with Storage-Key 4K-Byte-Block Facility Not Installed

When the storage-key 4K-byte-block facility is not installed, two keys are associated with the block, and the block is called a double-key 4K-byte block.

In a double-key 4K-byte block, one key is associated with the first 2K-byte block and one with the second 2K-byte block. The keys are referred to as the low-order and high-order keys, just as the two 2K-byte blocks are referred to as the low-order and high-order 2K-byte blocks. The instructions INSERT STORAGE KEY, RESET REFERENCE BIT, and SET STORAGE KEY designate a 2K-byte block and operate on the high-order or low-order key, whichever is addressed. The instructions INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED designate a 4K-byte block and operate on both the high-order and low-order keys.

Storage Keys with Storage-Key 4K-Byte-Block Facility Installed

When the storage-key 4K-byte-block facility is installed, only one key is associated with a 4K-byte block, and it is called a single-key 4K-byte block. In a single-key 4K-byte block, the single key is associated with both 2K-byte blocks. The instructions INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED operate on the single key of the block. The action taken when the instructions INSERT STORAGE KEY, RESET REFERENCE BIT, and SET STORAGE KEY are executed depends upon the setting of the storage-key-exception control, bit 7 of control register 0.

Storage-Key-Exception Control

When the storage-key 4K-byte-block facility is installed, bit 7 of control register 0, the storage-key-exception-control bit, controls the execution of the instructions INSERT STORAGE KEY, RESET REFERENCE BIT, and SET STORAGE KEY. If the bit is zero, a special-operation exception is recognized. If the bit is one, the operation is performed on the single key associated with the 4K-byte block.

When the storage-key 4K-byte-block facility is not installed, a storage key is associated with each 2K-byte block, and bit 7 of control register 0 is ignored.

STORAGE-KEY-INSTRUCTION EXTENSIONS

The storage-key-instruction-extension facility includes the three instructions INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED. These instructions provide operations similar to those of INSERT STORAGE KEY, RESET REFERENCE BIT, and SET STORAGE KEY, except that they operate on both single-key and double-key 4K-byte blocks without reference to the state of the storage-key-exception-control bit and provide a 31-bit real address.

PROTECTION

Three protection facilities are provided to protect the contents of main storage from destruction or misuse by programs that contain errors or are unauthorized: key-controlled protection, segment protection, and low-address protection.

The protection facilities are applied independently; access to main storage is only permitted when none of the facilities prohibit the access.

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

KEY-CONTROLLED PROTECTION

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

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

The protection action is summarized in the figure "Summary of Protection Action."

When the access to storage is initiated by the CPU and key-controlled protection applies, the PSW key is the access key, except that, for the second operand of MOVE WITH KEY and MOVE TO PRIMARY and the first operand of MOVE TO SECONDARY, the access key is specified in a general register. The PSW key occupies bit positions 8-11 of the current PSW.

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

Explanation:

Match The four access-control bits of the storage key are equal to the access key, or the access key is zero.

Yes Access is permitted.

No Access is not permitted. On fetching, the information is not made available to the program; on storing, the contents of the storage location are not changed.

Summary of Protection Action

When the access to storage is for the purpose of channel-program execution, the subchannel key associated with the I/O operation is the access key. The subchannel key is specified for an I/O operation in bit positions 0-3 of the channel-address word (CAW); the subchannel key is later placed in bit positions 0-3 of the channel-status word (CSW) stored as a result of the I/O operation.

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

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

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

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

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

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

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

- An interruption
- Updating the interval timer
- CPU logout
- Fetching of table entries for dynamic-address translation, PC-number translation, ASN translation, or ASN authorization
- DAS tracing
- A store-status function
- Fetching the CAW during the execution of an I/O instruction
- Storing of the CSW by an I/O instruction or interruption
- Storing channel identification during the execution of STORE CHANNEL ID
- A limited channel logout
- A full channel logout
- Initial program loading

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

SEGMENT PROTECTION

The segment-protection facility controls access to virtual storage by using the segment-protection bit in each segment-table entry. It provides protection against improper storing.

The segment-protection bit, bit 29 of the segment-table entry, controls whether storing is allowed into the corresponding segment. When the bit is zero, both fetching and storing are permitted; when the bit is one, only fetching is permitted. When an attempt is made to store into a protected segment, a program interruption for protection takes place. The contents of the protected location remain unchanged.

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

LOW-ADDRESS PROTECTION

The low-address-protection facility provides protection against the destruction of main-storage information used by the CPU during interruption processing. This is accomplished by prohibiting instructions from storing with effective addresses in the range 0 through 511. The range criterion is applied before address transformation, if any, of the address by dynamic address translation or prefixing.

Low-address protection is under control of bit 3 of control register 0, the low-address-protection-control bit. When the bit is zero, low-address protection is off; when the bit is one, low-address protection is on.

If an access is prohibited because of low-address protection, the contents of the protected location remain unchanged, a program interruption for a protection exception takes place, and the unit of operation is suppressed or the instruction terminated.

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

Low-address protection is not applied to accesses made by the CPU or channel for such sequences as interruptions, updating the interval timer, CPU logout, and the initial-program-loading and store-status functions, nor is it applied to data stores during I/O data transfer. However, explicit stores by a program at any of these locations are subject to low-address protection.

Programming Note

Low-address protection and key-controlled protection apply to the same store accesses, except that:

- Low-address protection does not apply to storing performed by a channel, whereas key-controlled protection does.

- Key-controlled protection does not apply to DAS tracing or the second operand of TEST BLOCK, whereas low-address protection does.

REFERENCE RECORDING

Reference recording provides information for use in selecting pages for replacement. Reference recording uses the reference bit, bit 5 of the storage key. A reference bit is provided in each storage key when the dynamic-address-translation facility is installed. The reference bit is set to one each time a location in the corresponding storage block is referred to either for fetching or storing information, regardless of whether the CPU performing the access is in the EC mode or BC mode or whether DAT is on or off in that CPU.

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

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

INSERT STORAGE KEY
INSERT STORAGE KEY EXTENDED
RESET REFERENCE BIT (reference bit is set to zero)
RESET REFERENCE BIT EXTENDED (reference bit is set to zero)
SET STORAGE KEY (reference bit is set to a specified value)
SET STORAGE KEY EXTENDED (reference bit is set to a specified value)

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

CHANGE RECORDING

Change recording provides information as to which pages have to be saved in auxiliary storage when they are replaced in main storage. Change recording uses the change bit, bit 6 of the storage key. A change bit is provided in each storage key when the dynamic-address-translation facility is installed.

The change bit is set to one each time a store access causes the contents in the corresponding storage block to be changed. A store access that does not change the contents of storage may or may not set the change bit to one.

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

1. For the CPU, a store access is prohibited whenever an access exception exists for that access, or whenever an exception exists which is of higher priority than the priority of an access exception for that access.

2. For a channel, a store access is prohibited whenever a key-controlled-protection violation exists for that access.

Change recording is always active and takes place for all store accesses to storage, including those made by any CPU, channel, or operator facility. It takes place for implicit references made by the machine, such as those which are part of interruptions.

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

RESET REFERENCE BIT
RESET REFERENCE BIT EXTENDED
SET STORAGE KEY (change bit is set to a specified value)
SET STORAGE KEY EXTENDED (change bit is set to a specified value)

Change bits which have been changed from zeros to ones are not necessarily restored to zeros on CPU retry (see the section "CPU Retry" in Chapter 11, "Machine-Check Handling"). See the section "Exceptions to Nullification and Suppression" in Chapter 5, "Program Execution," for a description of the handling of the change bit in certain unusual situations.

PREFIXING

Prefixing provides the ability to assign the range of real addresses 0-4095 (the prefix area) to a different block in absolute storage for each CPU, thus permitting more than one CPU sharing main storage to operate concurrently with a minimum of interference, especially in the processing of interruptions. Prefixing is provided as part of the multiprocessing facility.

Prefixing causes real addresses in the range 0-4095 to correspond to the block of 4K absolute addresses identified by the value in the prefix register for the CPU, and the block of real addresses identified by the value in the prefix register to correspond to absolute addresses 0-4095. The remaining real addresses are the same as the corresponding absolute addresses. This transformation allows each CPU to access all of main storage, including the first 4K bytes and the locations designated by the prefix registers of other CPUs.

The relationship between real and absolute addresses is graphically depicted in the figure "Relationship between Real and Absolute Addresses."

The prefix is a 19-bit quantity contained in bit positions 1-19 of the prefix register. Bits 1-7 of the prefix register are always zeros. The register has the following format:

/	0000000	Prefix	/////////////
0	1	8	20 31

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. When the contents of the prefix register are changed, the change is effective for the next sequential instruction.

With the introduction of the storage-key-instruction-extension facility, the test-block facility, and the extended-real-addressing facility, prefixing is described in terms of 31-bit real addresses, whether or not these facilities are installed. All real addresses are considered to be 31 bits, with any shorter address fields extended to 31 bits by appending zeros on the left. Thus, 24-bit real addresses are extended to 31 bits by appending zeros on the left.

When prefixing is applied, the real address is transformed into an absolute address by using one of the following rules, depending on bits 1-19 of the real address:

1. Bits 1-19 of the address, if all zeros, are replaced with bits 1-19 of the prefix.

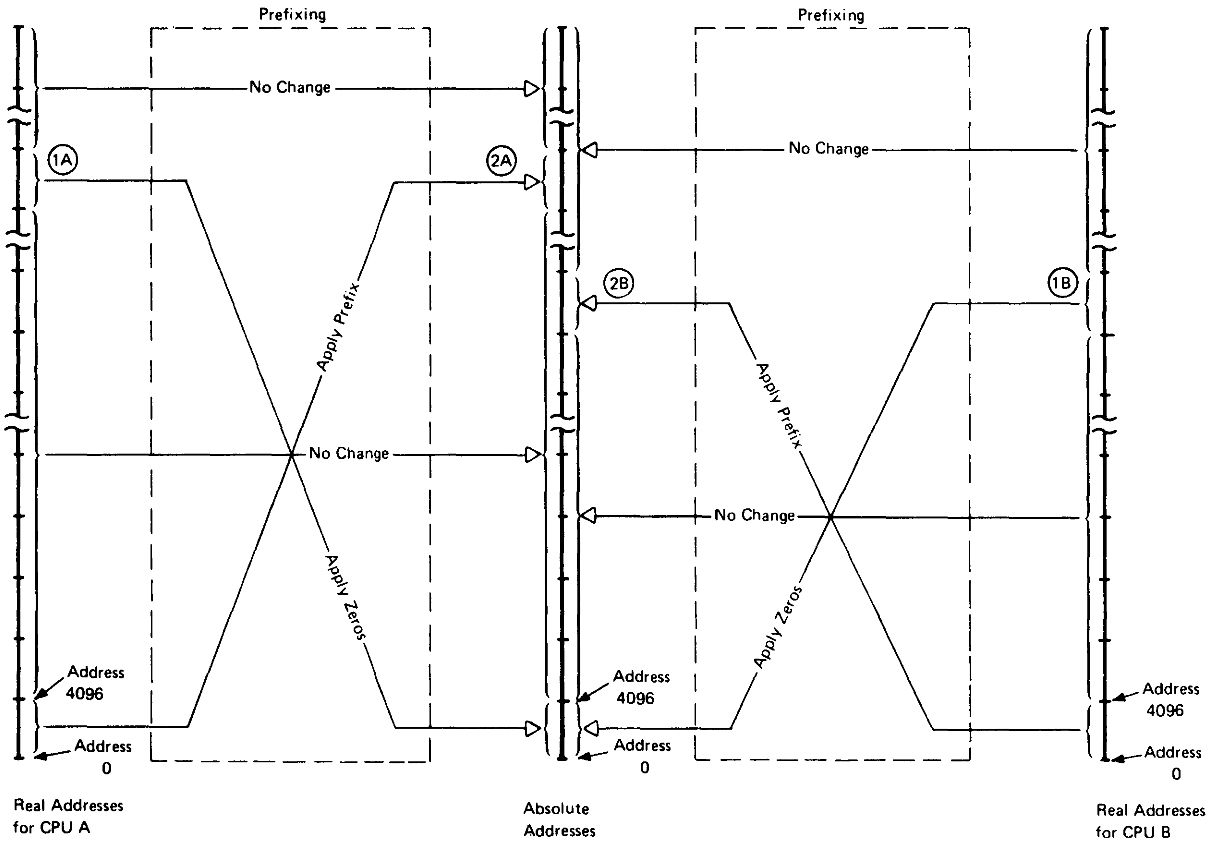
2. Bits 1-19 of the address, if equal to bits 1-19 of the prefix, are replaced with zeros.

3. Bits 1-19 of the address, if not all zeros and not equal to bits 1-19 of the prefix, remain unchanged.

In all cases, bits 20-31 of the 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 the prefix register contains all zeros, in which case a real address and its corresponding absolute address are identical.



(1) Real addresses in which bits 1-19 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 real locations 0-4095.

Relationship between Real and Absolute Addresses

ADDRESS SPACES

An address space is a consecutive sequence of integer numbers (virtual addresses), together with the specific transformation parameters which allow each number to be associated with a byte location in storage. The sequence starts at zero and proceeds left to right.

When a virtual address is used by a CPU to access main storage, it is first converted, by means of dynamic address translation (DAT), to a real address, and then, by means of prefixing, to an absolute address. DAT uses two levels of tables (segment tables and page tables) as transformation parameters. The designation (origin and length) of a segment table is found for use by DAT in a control register.

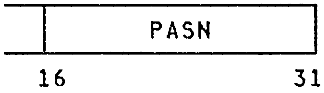
When DAS is not installed, the CPU can translate virtual addresses belonging to one address space -- the primary address space, which consists of primary virtual addresses. When DAS is installed, at any instant the CPU can translate virtual addresses of two address spaces -- the primary address space, consisting of primary virtual addresses, and the secondary address space, consisting of secondary virtual addresses. The segment table defining the primary address space is specified by control register 1 and that defining the secondary address space by control register 7.

With DAS, each address space is assigned an address-space number (ASN). An ASN-translation mechanism is provided with DAS, which, given an ASN, can locate (by using a two-level table lookup) the designation of the segment table which defines the address space. Certain instructions use ASN translation and load the resulting segment-table designation into the appropriate control register.

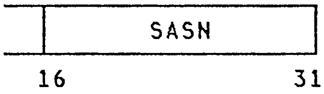
By using the ASN-translation mechanism, any one of up to 64K address spaces can be selected to become the primary or secondary address space.

The ASNs for the primary and secondary address spaces are assigned positions in control registe . The ASN for the primary address s ce, called the primary ASN, is assigned bits 16-31 of control register 4, and that for the secondary address space, called the secondary ASN, is assigned bits 16-31 of control register 3. The registers have the following formats:

Control Register 4



Control Register 3



An instruction that uses ASN translation and loads the primary or secondary segment-table designation into the appropriate control register also loads the corresponding ASN into the appropriate control register.

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

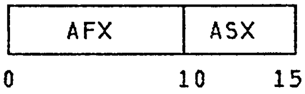
ASN TRANSLATION

ASN translation is the process of translating the 16-bit ASN to locate the address-space-control parameters. ASN translation is performed as part of PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), and SET SECONDARY ASN with space switching (SSAR-ss). ASN translation is also performed as part of LOAD ADDRESS SPACE PARAMETERS. For PC-ss and PT-ss, the ASN which is translated replaces the primary ASN in control register 4. For SSAR-ss, the ASN which is translated replaces the secondary ASN in control register 3. These two translation processes are called primary ASN translation and secondary ASN translation, respectively, and both can occur for LOAD ADDRESS SPACE PARAMETERS. The ASN-translation process is the same for both primary and secondary ASN translation; only the uses of the results of the process are different.

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

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

ASN

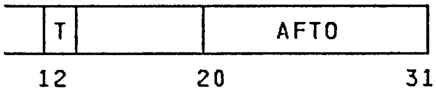


The AFX is used to select an entry from the ASN first table. The origin of the ASN first table is designated by the ASN-first-table origin in control register 14. The ASN-first-table entry contains the origin of the ASN second table. The ASX is used to select an entry from the ASN second table. This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

ASN-TRANSLATION CONTROLS

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

Control Register 14



ASN-Translation Control (T): Bit 12 of control register 14 is the ASN-translation-control bit. This bit provides a mechanism whereby the control program can indicate whether ASN translation can occur while a particular program is being executed. Bit 12 must be one to allow completion of these instructions:

LOAD ADDRESS SPACE PARAMETERS
SET SECONDARY ASN
PROGRAM CALL with space switching
PROGRAM TRANSFER with space switching

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

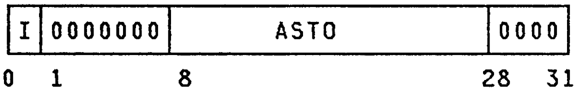
ASN-First-Table Origin (AFTO): Bits 20-31 of control register 14, with 12 zeros appended on the right, form a 24-bit real address that designates the beginning of the ASN first table. With extended real addressing, the ASN-first-table origin is still a 24-bit real address and is extended on the left with zeros.

ASN-TRANSLATION TABLES

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

ASN-First-Table Entries

The entry fetched from the ASN first table has the following format:



The fields in the entry are allocated as follows:

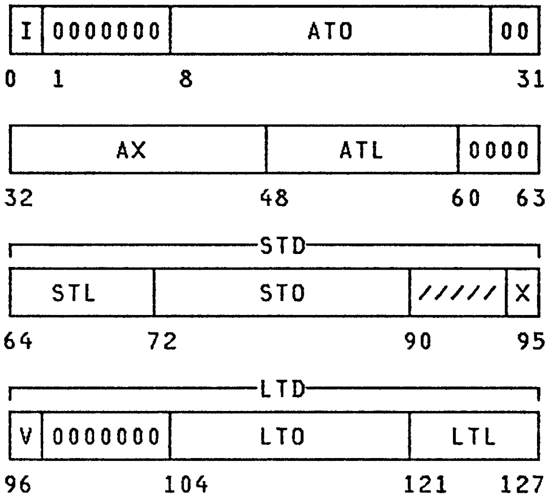
AFX-Invalid Bit (I): Bit 0 controls whether the ASN second table associated with the ASN-first-table entry is available. When bit 0 is zero, ASN translation proceeds by using the designated ASN second table. When the bit is one, the ASN translation cannot continue.

ASN-Second-Table Origin (ASTO): Bits 8-27, with four zeros appended on the right, are used to form a 24-bit real address that designates the beginning of the ASN second table. With extended real addressing, the ASN-second-table origin is still a 24-bit real address and is extended on the left with zeros.

Bits 1-7 and 28-31 of the AFT entry must be zeros; otherwise, an ASN-translation-specification exception is recognized as part of the execution of the instruction using that entry for ASN translation.

ASN-Second-Table Entries

The entry fetched from the ASN second table has the following format:



The fields in the entry are allocated as follows:

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

Authority-Table Origin (ATO): Bits 8-29, with two zeros appended on the right, are used to form a 24-bit real address that designates the beginning of the authority table. With extended real addressing, the authority-table origin is still a 24-bit real address and is extended on the left with zeros.

Authorization Index (AX): Bits 32-47 are used as a result of primary ASN translation by PROGRAM CALL and PROGRAM TRANSFER and may be used by LOAD ADDRESS SPACE PARAMETERS. The AX field is ignored for secondary ASN translation.

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

Segment-Table Designation (STD): Bits 64-95 are used as a result of ASN translation to replace the primary-segment-table designation (PSTD) or the secondary-segment-table designation (SSTD). For SET SECONDARY ASN, the STD field is placed in the SSTD, bits 0-31 of control register 7. For PROGRAM CALL, the STD field is placed in the PSTD, bits 0-31 of control register 1. Each of these actions may occur independently for LOAD ADDRESS SPACE PARAMETERS. For PROGRAM TRANSFER, the

STD field is placed in both the PSTD and SSTD, bits 0-31 of control registers 1 and 7, respectively. The contents of the entire STD field are placed in the appropriate control registers without being inspected for validity.

Space-Switch-Event Control (X): Bit 31 of the segment-table designation is the space-switch-event-control bit. When, in PC-ss or PT-ss, this bit is one in control register 1 either before or after the execution of the PC-ss or PT-ss, a program interruption for a space-switch event occurs after the execution of the instruction is completed. When, in LOAD ADDRESS SPACE PARAMETERS, this bit is one during primary ASN translation, this fact is indicated by the condition code.

Linkage-Table Designation (LTD): Bits 96 and 104-127 are used as a result of primary ASN translation. The linkage-table-designation field contains the subsystem-linkage-control bit (V) (bit 96), the linkage-table origin (LTO) (bits 104-120), and the linkage-table length (LTL) (bits 121-127). The contents of the LTD field are placed in control register 5 as a result of primary ASN translation.

Bits 1-7, 30, 31, 60-63, and 97-103 of the AST entry must be zeros; otherwise, an ASN-translation-specification exception is recognized as part of the execution of the instruction using that entry for ASN translation.

Programming Note

The unused portion of the STD field, bits 90-94 of the AST entry, which corresponds to bits 26-30 of the PSTD and SSTD, should be set to zeros. These bits are reserved for future expansion, and programs which place nonzero values in these bit positions may not operate compatibly on future machines.

ASN-TRANSLATION PROCESS

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

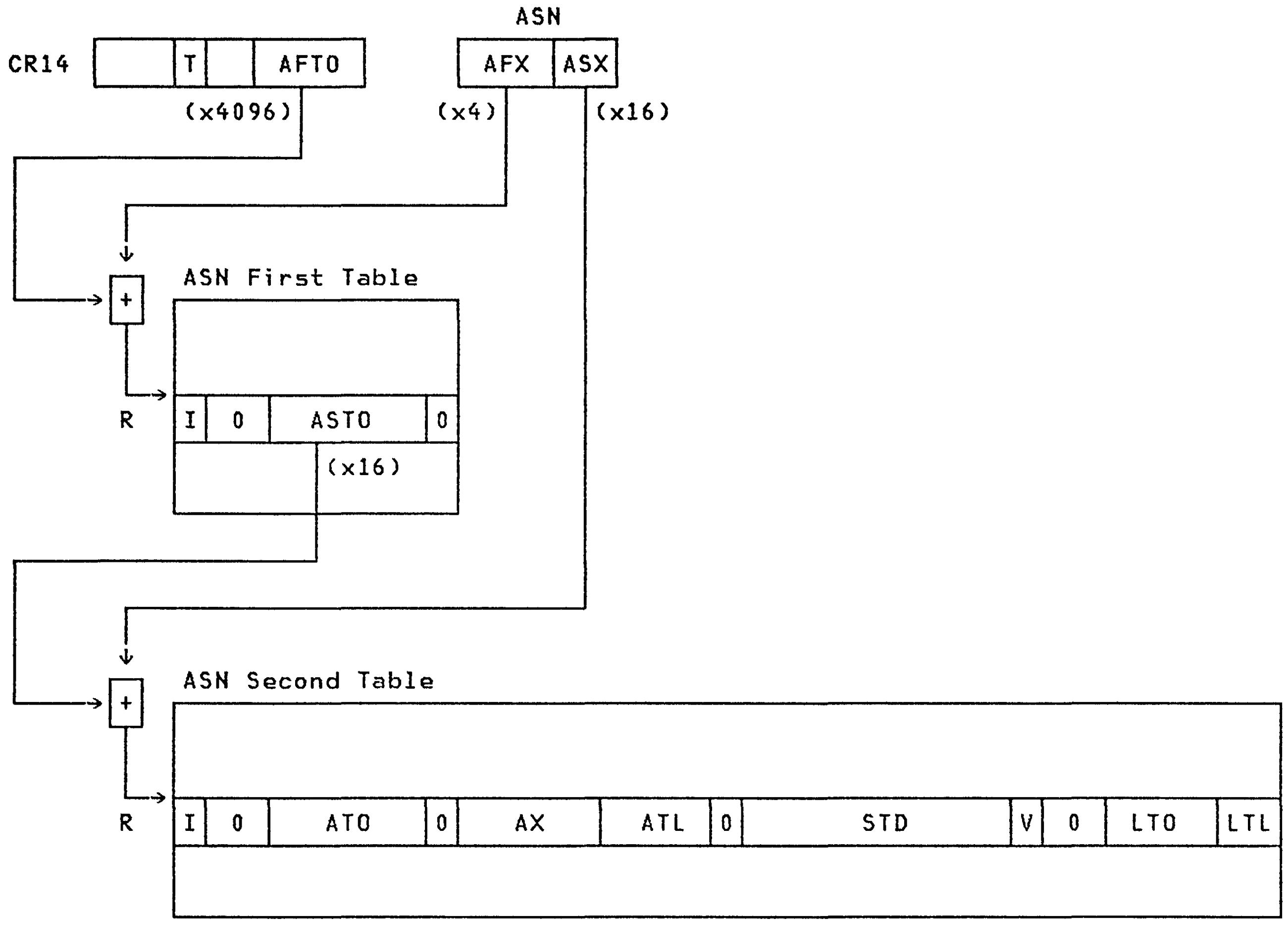
The ASN first index is used to select an entry from the ASN first table. This entry designates the ASN second table to be used.

The ASN second index is used to select an entry from the ASN second table. This entry contains the address-space-control parameters.

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

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

The ASN translation process is shown in the figure "ASN Translation."



R: Address is real

ASN Translation

ASN-First-Table Lookup

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

The 24-bit real address of the ASN-first-table entry is obtained by appending 12 zeros on the right to the AFT origin contained in bit positions 20-31 of control register 14 and adding the AFX with two rightmost and 12 leftmost zeros appended. This addition cannot cause a carry into bit position 7. With extended real addressing, this 24-bit real address is extended on the left with zeros.

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

Bit 0 of the four-byte AFT entry specifies whether the corresponding AST is available. If this bit is one, an AFX-translation exception is recognized. If bit positions 1-7 and 28-31 of the AFT entry do not contain zeros, an ASN-translation-specification exception is recognized. When no exceptions are recognized, the entry fetched from the AFT is used to access the AST.

ASN-Second-Table Lookup

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

The 24-bit real address of the ASN-second-table entry is obtained by appending four zeros on the right to bits 8-27 of the ASN-first-table entry and adding the ASX portion with four rightmost and 14 leftmost zeros appended. A carry, if any, into bit position 7 is ignored. With extended real addressing, this 24-bit real address is extended on the left with zeros; thus, the ASN-second table can wrap from 2^{24} - 1 to zero.

The 16 bytes of the ASN-second-table entry appear to be fetched word-concurrent as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining three words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address which is generated for fetching the ASN-second-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

Bit 0 of the 16-byte ASN-second-table entry specifies whether the address space is accessible. If this bit is one, an ASX-translation exception is recognized. If bit positions 1-7, 30, 31, 60-63, and 97-103 of the ASN-second-table entry do not contain zeros, an ASN-translation-specification exception is recognized.

Recognition of Exceptions during ASN Translation

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

ASN AUTHORIZATION

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

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

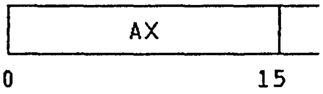
The ASN authorization is performed by means of an authority table in real storage which is designated by the authority-table-origin and authority-table-length fields in the ASN-second-table entry.

ASN-AUTHORIZATION CONTROLS

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

Control Register 4

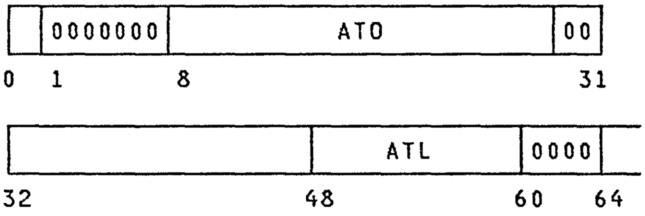
For PT-ss and SSAR-ss, the current contents of control register 4 include the authorization index. For LOAD ADDRESS SPACE PARAMETERS, the value which will become the new contents of control register 4 is used. The register has the following format:



Authorization Index (AX): Bits 0-15 of control register 4 are used as an index to locate the authority bits in the authority table.

ASN-Second-Table Entry

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



Authority-Table Origin (ATO): Bits 8-29, with two zeros appended on the right, are used to form a 24-bit real address that designates the beginning of the authority table. With extended real addressing, the authority-table origin is still a 24-bit real address and is extended on the left with zeros.

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

Authority-Table Entries

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

```
|PS|PS|PS|PS|
0          7
```

The fields are allocated as follows:

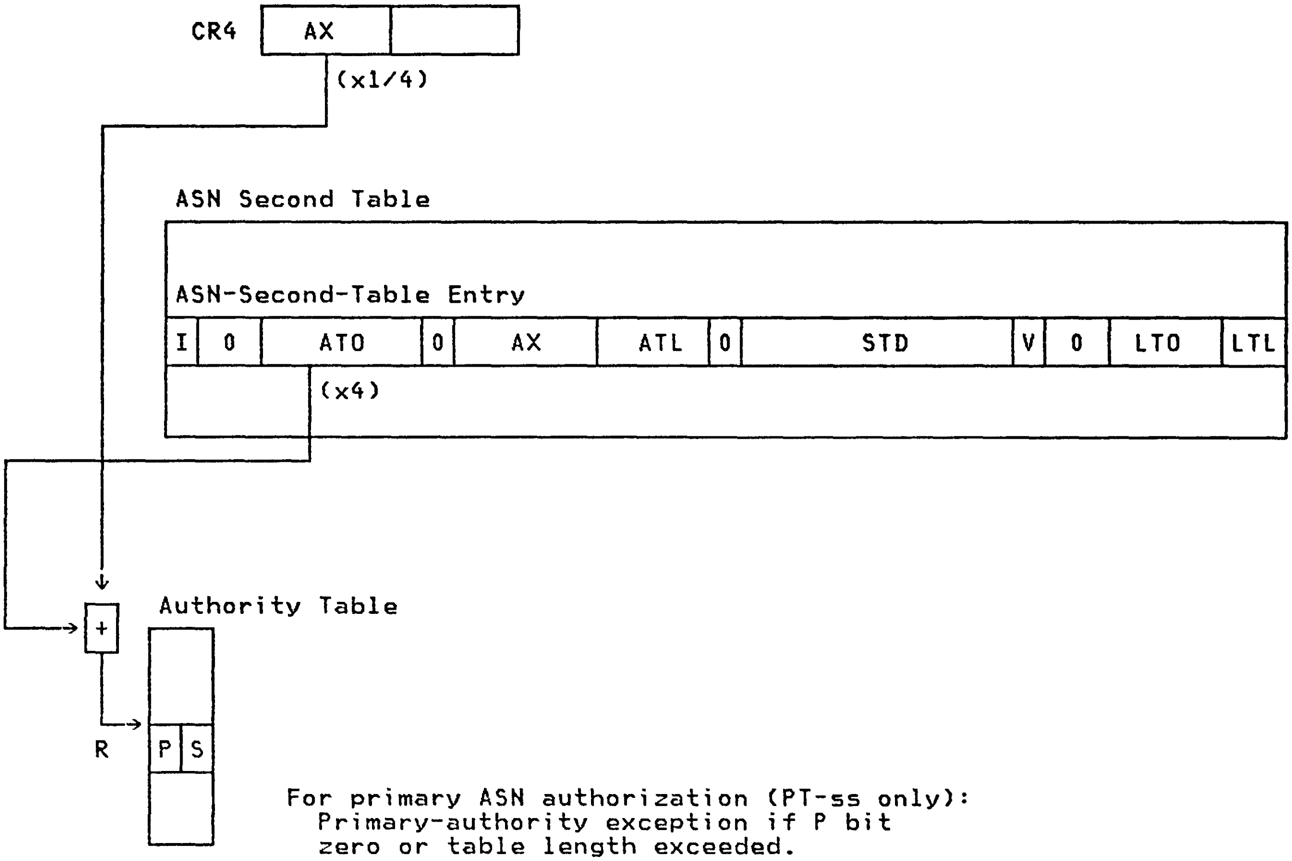
Primary Authority (P): The left bit of an authority-table entry controls whether the program with the authorization index corresponding to the entry is permitted to establish the address space as a primary address space. If the P bit is one, the access is permitted. If the P bit is zero, the access is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

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

ASN-AUTHORIZATION PROCESS

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

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



For primary ASN authorization (PT-ss only):
Primary-authority exception if P bit
zero or table length exceeded.

For secondary ASN authorization (SSAR-ss only):
Secondary-authority exception if S bit
zero or table length exceeded.

For secondary ASN authorization (LASP only):
Set condition code 2 if S bit zero or
table length exceeded.

R: Address is real

ASN Authorization

Authority-Table Lookup

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

The authorization index is contained in bit positions 0-15 of control register 4.

Bit positions 8-31 of the AST entry contain the 24-bit real address of the authority table (ATO), and bit positions 48-59 contain the length of the authority table (ATL).

The 24-bit real address of a byte in the authority table is obtained by appending two zeros on the right to the authority-table origin and adding the 14 leftmost bits of the authorization index with 10 zeros appended on the left. A carry, if any, into bit position 7 is ignored. With extended real addressing, this 24-bit real address is extended on the left with zeros; thus, the authority table can wrap from $2^{24} - 1$ to zero.

As part of the authority-table-entry-lookup process, bits 0-11 of the authorization index are compared against the authority-table length. If the compared portion is greater than the authority-table length, a primary-authority exception or secondary-authority exception is recognized for PT-ss or SSAR-ss, respectively. For LOAD ADDRESS SPACE PARAMETERS, when the authority-table length is exceeded, condition code 2 is set.

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

The byte contains four authority-table entries of two bits each. The rightmost two bits of the authorization index, bits 14 and 15 of control register 4, are used to select one of the four entries. The left or right bit of the entry is then tested, depending on whether the authorization test is for a primary ASN or a secondary ASN. The following table shows the bit which is selected from the byte as a function of bits 14 and 15 of the authorization index and the instruction PT-ss, SSAR-ss, or LOAD ADDRESS SPACE PARAMETERS.

Authorization-Index Bits		Bit Selected from Authority-Table Byte for Test	
14	15	P Bit (PT-ss)	S Bit (SSAR-ss or LASP)
0	0	0	1
0	1	2	3
1	0	4	5
1	1	6	7

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

Recognition of Exceptions during ASN Authorization

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

Programming Note

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

DYNAMIC ADDRESS TRANSLATION

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

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

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

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

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

The dynamic-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. When DAS is installed, the dynamic-address-translation facility also makes use of control register 7.

The dynamic-address-translation facility includes the handling of 2K-byte and 4K-byte pages and 64K-byte and 1M-byte segments. On some models, the 2K-byte-page size and 1M-byte-segment size may not be offered.

Dynamic address translation is enhanced by that part of the extended facility that includes the instruction INVALIDATE PAGE TABLE ENTRY and the common-segment facility. On some models, the common-segment facility permits improvement of TLB utilization by means of a common-segment bit in the segment-table entry. On other models, this bit is ignored, with no performance improvement.

Dynamic address translation is the process of translating a virtual address during a storage reference into the corresponding real address. When DAT is off, the logical address is treated as a real address. When DAS is not installed and DAT is on, a logical address is treated as a virtual address and is translated during a storage reference into the corresponding real address. When DAS is installed and DAT is on, the virtual address may be either a primary virtual address or a secondary virtual address. Primary virtual addresses are translated by means of the primary segment-table designation and secondary virtual addresses by means of the secondary segment-table designation. After selection of the appropriate segment-table designation, the translation process is the same for both types of virtual address.

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

The virtual address, accordingly, is divided into a segment-index (SX) field, a page-index (PX) field, and a byte-index (BX) field. The size of these fields depends on the segment and page size.

The segment index starts with bit 8 of the virtual 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 consists of the remaining 11 or 12 rightmost bits of the virtual address. The virtual address has the following format:

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

```
┌────────┬─────────┬─────┬─────────────┐
│////////│   SX    │ PX  │     BX      │
└────────┴─────────┴─────┴─────────────┘
0         8         16    20           31
```

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

```
┌────────┬─────────┬──────┬────────────┐
│////////│   SX    │ PX   │     BX     │
└────────┴─────────┴──────┴────────────┘
0         8         16     21          31
```

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

```
┌────────┬────┬─────────┬─────────────┐
│////////│ SX │   PX    │     BX      │
└────────┴────┴─────────┴─────────────┘
0         8    12        20           31
```

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

```
┌────────┬────┬───────────┬───────────┐
│////////│ SX │    PX     │    BX     │
└────────┴────┴───────────┴───────────┘
0        8   12          21         31
```

Virtual addresses are translated into real addresses by means of two translation tables: a segment table and a page table. These 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 virtual addresses.

TRANSLATION CONTROL

Address translation is controlled by the DAT-mode bit in the EC-mode PSW and by a set of bits, referred to as the translation parameters, in control registers 0 and 1. When DAS is installed, an additional bit in the EC-mode PSW is included, and control register 7 is included as part of the translation parameters. Additional controls are located in the translation tables.

Translation Modes

When the dynamic-address-translation facility is installed without DAS, the CPU can operate with DAT either on or off. The mode of operation is controlled by bit 5 of the EC-mode PSW, the DAT-mode bit. When this bit is one, DAT is on, and logical addresses are treated as virtual addresses; when this bit is zero or the BC mode is specified, DAT is off, and logical addresses are treated as real addresses.

When DAS is installed, two bits in the EC-mode PSW control dynamic address translation: bit 5, the DAT-mode bit, and bit 16, the address-space-control bit. When a BC-mode PSW is specified, or, when in an EC-mode PSW the DAT-mode bit is zero, DAT is off, the CPU is said to be in the real mode, and instruction and logical addresses are treated as real addresses. When, in an EC-mode PSW, the DAT-mode bit is one (DAT is on) and the address-space-control bit is zero, the CPU is said to be in the primary-space mode, and instruction and logical addresses are treated as primary virtual addresses. When, in an EC-mode PSW, DAT is on and the address-space-control bit is one, the CPU is said to be in the secondary-space mode, and logical addresses are treated as secondary virtual addresses. The various modes are shown in the figures "Translation Modes without DAS" and "Translation Modes with DAS."

PSW Bit				Handling of Addresses	
5	12	DAT	Mode	Logical Addresses	Instruction Addresses
-	0	Off	Real mode (BC mode)	Real	Real
0	1	Off	Real mode	Real	Real
1	1	On	Primary-space mode	Primary virtual	Primary virtual

Translation Modes without DAS

PSW Bit					Handling of Addresses	
5	12	16	DAT	Mode	Logical Addresses	Instruction Addresses
-	0	-	Off	Real mode (BC mode)	Real	Real
0	1	-	Off	Real mode	Real	Real
1	1	0	On	Primary-space mode	Primary virtual	Primary virtual
1	1	1	On	Secondary-space mode	Secondary virtual	See note

Translation Modes with DAS

Note: When the CPU is in the secondary-space mode, it is unpredictable whether instruction addresses are treated as primary virtual or secondary virtual addresses. However, all copies of an instruction used in a single execution are fetched from a single space, and the machine can change the interpretation of instruction addresses as primary virtual or secondary virtual only between instructions and only by performing a checkpoint-synchronizing function.

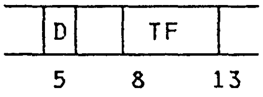
Programming Notes

1. Predictable program operation is ensured in the secondary-space mode only when the instructions are fetched from virtual-address locations which translate to the same real address by means of both the primary and secondary segment tables. Thus, a program should not enter the secondary-space mode unless the aforementioned conditions exist.

2. The requirement limiting when the CPU can change the address space used for fetching instructions eliminates problems with CPU retry, DAT pretesting, and trial execution of instructions for the purposes of determining PER events.

Control Register 0

When DAS is not installed, five bits are provided in control register 0 which are used in controlling dynamic address translation. When DAS is installed, a sixth bit is provided. The bits are assigned as follows:



Secondary-Space Control (D): Bit 5 of control register 0 is the secondary-space-control bit. This bit is provided as part of DAS. When this bit is zero and execution of MOVE TO PRIMARY, MOVE TO SECONDARY, or SET ADDRESS SPACE CONTROL is attempted, a special-operation exception is recognized. When this bit is one, it indicates that the secondary segment table is attached when the CPU is in the primary-space mode.

Translation Format (TF): Bits 8-12 of control register 0 are called the translation format, which controls the page size and segment size. Some models do not implement all four of the combinations, as shown in the following table.

Bits of Control Register 0							
8	9	10	11	12	Provided	Page Size (Bytes)	Segment Size (Bytes)
0	1	0	0	0	Opt	2K	64K
0	1	0	1	0	Opt	2K	1M
1	0	0	0	0	Std	4K	64K
1	0	0	1	0	Opt	4K	1M
All others					Inv		

Explanation:

Opt Optional. The code is invalid on some models, even though the translation facility is installed.

Std Standard. The code is valid on all models with the translation facility installed.

Inv Invalid. The code is not valid on any model.

Translation Format

When an invalid bit combination is detected in bit positions 8-12, a translation-specification exception is recognized as part of the execution of an instruction using address translation.

Control Register 1

Control register 1 contains the primary segment-table designation (PSTD). The register has the following format:

PSTL	Primary Segment-Table Origin		X
0	8	26	31

Primary Segment-Table Length (PSTL): Bits 0-7 of control register 1 specify the length of the primary 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 primary segment table, in units of 64 bytes, is one more than the PSTL value. The contents of the length field are used to establish whether the entry designated by the segment-index portion of a primary virtual address falls within the primary segment table. Without DAS, this field is sometimes referred to as the segment-table length.

Primary Segment-Table Origin (PSTO): Bits 8-25 of control register 1, with six zeros appended on the right, form a 24-bit real address that designates the beginning of the primary segment table. Without DAS, this field is sometimes referred to as the segment-table origin. With extended real addressing, the primary segment-table origin is still a 24-bit real address and extended on the left with zeros.

Space-Switch-Event-Control Bit (X): When bit 31 of control register 1 is one and execution of PROGRAM CALL with space switching (PC-ss) or PROGRAM TRANSFER with space switching (PT-ss) is completed, a space-switch-event program interruption occurs. The space-switch-event-control bit is also examined by LOAD ADDRESS SPACE PARAMETERS, and, if it is one, condition code 3 is set. When DAS is not installed, this bit is ignored.

Bits 26-30 of control register 1 are not assigned and are ignored.

Control Register 7

When DAS is installed, control register 7 contains the secondary segment-table designation (SSTD). The register has the following format:

SSTL	Secondary Segment-Table Origin	
0	8	26 31

Secondary Segment-Table Length (SSTL): Bits 0-7 of control register 7 specify the length of the secondary 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 secondary segment table, in units of 64 bytes, is one more than the SSTL value. The contents of the length field are used to establish whether the entry designated by the segment-index portion of a secondary virtual address falls within the secondary segment table.

Secondary Segment-Table Origin (SSTO): Bits 8-25 of control register 7, with six zeros appended on the right, form a 24-bit real address that designates the beginning of the secondary segment table. With extended real addressing, the secondary segment-table origin is still a 24-bit real address and is extended on the left with zeros.

Bits 26-31 of control register 7 are not assigned and are ignored.

Programming Notes

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

2. The information pertaining to dynamic address translation is considered to be used when an instruction is executed with DAT on or when INVALIDATE PAGE TABLE ENTRY or 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, or when the PSW specifies the wait state.

TRANSLATION TABLES

The translation process consists in a two-level lookup using two tables: a segment table and a page table. These tables reside in real storage.

Segment-Table Entries

The entry fetched from the segment table has the following format:

PTL	0000	Page-Table Origin	P	C	I
0	4	8	29		31

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

Page-Table Length (PTL): Bits 0-3 specify the length of the page table in increments that are equal to 1/16 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 1/16 of the maximum size, is one more than the PTL value. The length field is compared against the leftmost four bits of the page-index portion of the virtual address to determine whether the page index designates an entry within the page table.

Page-Table Origin: Bits 8-28, with three zeros appended on the right, form a 24-bit real address that designates the beginning of a page table. With extended real addressing, the page-table origin is still a 24-bit real address and is extended on the left with zeros.

Segment-Protection Bit (P): Bit 29, with the segment-protection facility installed, controls whether store accesses can be made in the segment. This protection mechanism is in addition to the key-controlled-protection and low-address-protection mechanisms. The bit has no effect on fetch accesses. If the bit is zero, stores are permitted to the segment, subject to the other protection mechanisms. If the bit is one, stores are disallowed. An attempt to store when the segment-protection bit is one causes a protection exception to be recognized.

Common-Segment Bit (C): Bit 30, with the common-segment facility installed, controls the use of translation-lookaside-buffer (TLB) copies of the segment-table entry and of the page table which it designates. A zero identifies a private segment; in this case, the segment-table entry and the page table it designates may be used only in association with the segment-table origin that designates the segment table in which the segment-table entry resides. A one identifies a common segment; in this case, the segment-table entry and the page table it designates may continue to be used for translating addresses corresponding to the segment index, even though a different segment table is specified. In some models, bit 30 in the segment-table entry is ignored, and all segments are treated as private.

The common-segment bit is used only for controlling the loading and use of TLB copies. When the common-segment facility is installed, the common-segment bit is ignored for explicit translation and for implicit translation not using the TLB.

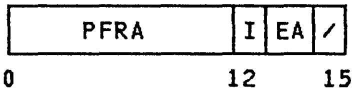
Segment-Invalid Bit (I): Bit 31 controls whether the segment associated with the segment-table entry is available. When the bit is zero, address translation proceeds by using the designated page table. When the bit is a one, the segment-table entry cannot be used for translation.

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 when these bits are not zeros; however, on some models the contents of these bit positions may be ignored. On machines with the segment-protection facility installed, bit 29 is interpreted as the segment-protection bit. On machines with the common-segment facility installed, bit 30 is interpreted as defined or is ignored.

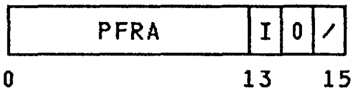
Page-Table Entries

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-Frame Real Address (PFRA): 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 these bits are concatenated with the contents of the byte-index field of the virtual address on the right, a 24-bit real storage address is obtained.

Page-Invalid Bit (I): 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 by using the page-table entry. When the bit is one, the page-table entry cannot be used for translation.

Extended-Storage-Address Bits (EA): When the extended-real-addressing facility is installed, bits 13 and 14 of the page-table entry with 4K-byte pages are the extended-storage-address bits. These bits become bits 6 and 7 of a 26-bit real address.

Except for bit position 15, 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. In models that provide the extended-real-addressing facility, bit positions 13 and 14 of the page-table entry for 4K-byte pages are used as the extended-storage-address bits and do not cause a translation-specification exception. Bit position 15 is unassigned and not checked for zero.

SUMMARY OF DYNAMIC-ADDRESS-TRANSLATION FORMATS

The first table summarizes the possible combinations of the page-frame real address (PFRA) field, byte-index field, and extended-storage-address bits in the formation of a real storage address.

The eight-bit length field in the segment-table designation 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 virtual address. With 1M-byte segments, only 16 segments can be selected, 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 selected, 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					
	PFRA without Extended Real Addressing		PFRA with Extended Real Addressing		Byte Index	
Size of Page (Bytes)	Bit Positions in Page-Table Entry	No. of Bits	Bit Positions in Page-Table Entry	No. of Bits	Bit Positions in Virtual Address	No. of Bits
2K	0-12	13	0-12	13	21-31	11
4K	0-11	12	13, 14, 0-11	14	20-31	12

Size of Segment (Bytes)	Segment Index Field Size (Bits)	Number of Address-able Segments	Max Segm Tbl		Segment-Table Increment (Bytes)
			Size (Bytes)	Usable Length Code	
64K	8	256	1,024	15	64
1M	4	16	64	0	64

Size of		Page Index Field Size (Bits)	Number of Pages in Segment	Max Page Tbl		Page-Table Increment (Bytes)
Segment (Bytes)	Page (Bytes)			Size (Bytes)	Usable Length Code	
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

TRANSLATION PROCESS

This section describes the translation process as it is performed implicitly before a virtual address is used to access main storage. The process of translating the operand address of LOAD REAL ADDRESS and TEST PROTECTION is the same, except that segment-translation and page-translation exceptions do not occur; such situations are instead indicated in the condition code. Translation of the operand address of LOAD REAL ADDRESS also differs in that the CPU may be in the real mode and the translation-lookaside buffer is not used.

Translation of a virtual address is performed by means of a segment table and a page table both of which reside in real storage. It is controlled by the DAT-mode bit in the PSW and by the translation parameters in control registers 0 and 1. When DAS is installed, translation is also controlled by the address-space-control bit in the PSW, and the translation parameters also include control register 7.

Effective Segment-Table Designation

The segment-table designation used for a particular address translation is called the effective segment-table designation. Accordingly, when a primary virtual address is translated, control register 1 is used as the effective segment-table designation, and when a secondary virtual address is translated, control register 7 is used as the effective segment-table designation. Without DAS, the term "effective segment-table designation" is synonymous with "primary segment-table designation."

The segment-index portion of the virtual address is used to select an entry from the segment table, the starting address and length of which are specified by the effective segment-table designation. This entry designates the page table to be used and, if the segment-protection facility is installed, provides the segment-protection bit.

The page-index portion of the virtual address is used to select an entry from the page table. This entry contains the leftmost bits of the real address that represents the translation of the virtual address.

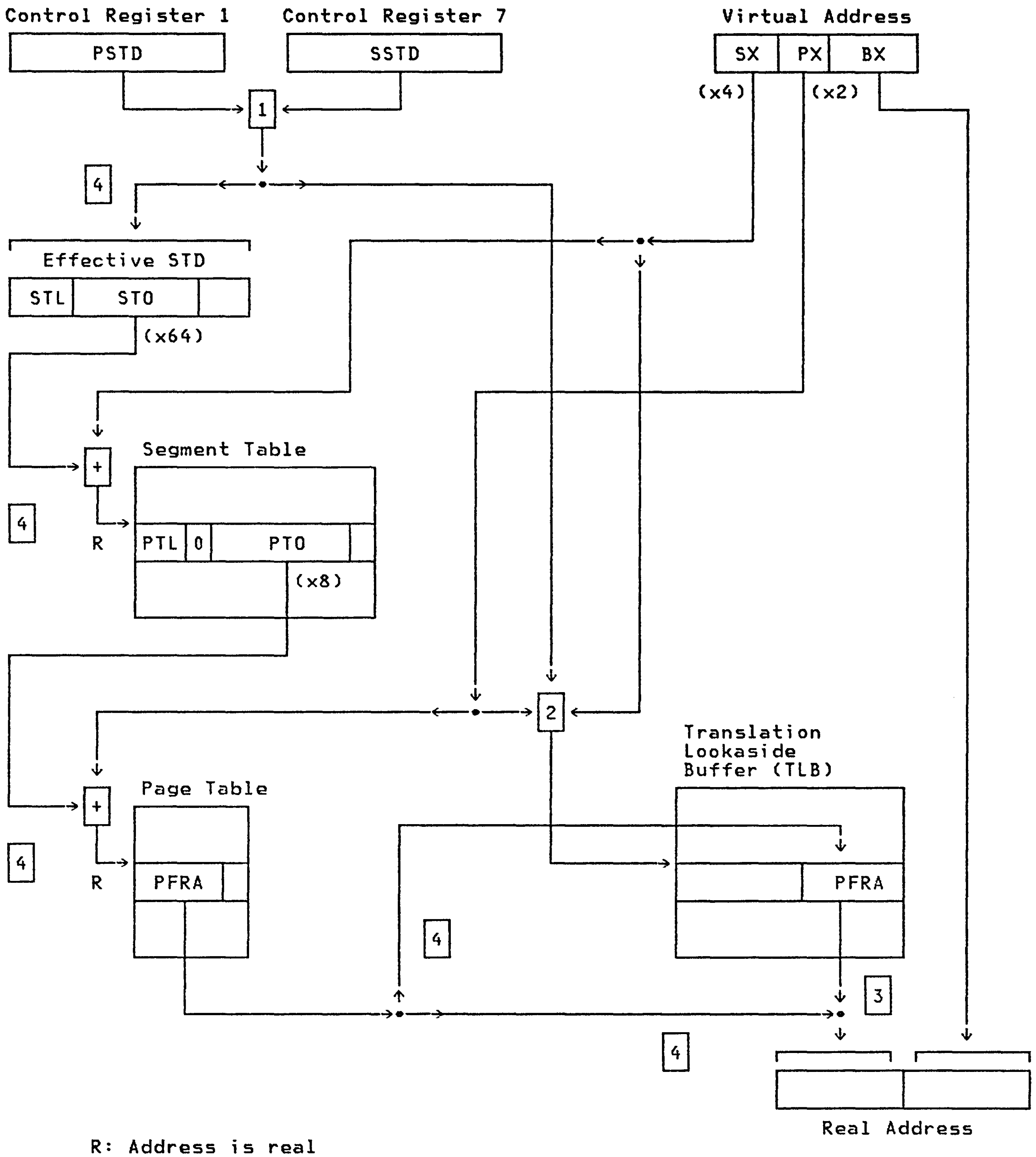
The byte-index field of the virtual address is used unchanged as the rightmost bit positions of the real address.

If the I bit is one in either the segment-table entry or the page-table entry, the entry is invalid, and the translation process cannot be completed for this virtual address. A segment-translation or a page-translation exception is recognized.

In order to eliminate the delay associated with references to translation tables in real storage, the information fetched from the tables normally is also placed in a special buffer, the translation-lookaside buffer (TLB), and subsequent translations involving the same table entries may be performed by using the information recorded in the TLB. The operation of the TLB is described in the section "Translation-Lookaside Buffer" in this chapter.

Whenever access to real storage is made during the address-translation process for the purpose of fetching an entry from a segment table or page table, key-controlled protection does not apply.

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



Translation Process (Part 1 of 2)

1 Control register 1 provides the primary segment-table designation for translation of a primary virtual address, and, when DAS is installed, control register 7 provides the secondary segment-table designation for translation of a secondary virtual address.

2 Information, which may include portions of the virtual address and the translation parameters, is used to search the TLB.

3 If a match exists, the page-frame real address from the TLB is used in forming the real address.

4 If no match exists, table entries in real storage are fetched. The resulting fetched entries, in conjunction with the search information, are used to translate the address and may be used to form an entry in the TLB.

Translation Process (Part 2 of 2)

Inspection of Control Register 0

The interpretation of the virtual address for translation purposes is controlled by the translation format, bits 8-12 of control register 0. If bits 8-12 contain an invalid code, a translation-specification exception is recognized.

Segment-Table Lookup

The segment-index portion of the virtual address, in conjunction with the segment-table origin contained in the effective segment-table designation, is used to select an entry from the segment table.

The 24-bit real address of the segment-table entry is obtained by appending six zeros to the right of bits 8-25 of the effective segment-table designation and adding the segment index to this value, with the rightmost bit position of the segment index aligned with bit position 29 of the address. A carry, if any, into bit position 7 is ignored. With extended real addressing, this 24-bit real address is extended on the left with zeros; thus, the segment table can wrap from $2^{24} - 1$ to zero.

As part of the segment-table-lookup process, the segment index is compared against the segment-table length, bits 0-7 of the effective segment-table designation, to establish whether the addressed entry is within the segment 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 zeros are appended to the left of bits 8-11 of the virtual 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 virtual address, a segment-translation exception is recognized.

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

Bit 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. 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 when they do not contain zeros; however, on some models they may be ignored.

On machines with the segment-protection facility, bit 29 is the segment-protection bit and does not cause a translation-specification exception; bit 29 is retained with the entry in the TLB.

On machines with the common-segment facility, bit 30 is the common-segment bit and does not cause a translation-specification exception. Bit 30 may be retained with the entry in the TLB, or it may be ignored.

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

Page-Table Lookup

The page-index portion of the virtual address, in conjunction with the page-table origin contained in the segment-table entry, is used to select an entry from the page table.

The 24-bit real address of the page-table entry is obtained by appending three zeros to the right of bits 8-28 of the segment-table entry and adding the page index, with the rightmost bit position of the page index aligned with bit 30 of the address. A carry, if any, into bit position 7 is ignored. With extended real addressing, this 24-bit real address is extended on the left with zeros; thus, the page table can wrap from $2^{24} - 1$ to zero.

As part of the page-table-lookup process, the four leftmost 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 leftmost bit positions of the page-index field, a page-translation exception is recognized.

The two bytes of the page-table entry appear to be fetched concurrently as observed by other CPUs. The fetch access is not subject to protection. When the storage address generated for fetching the page-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the unit of operation is suppressed.

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

When the extended-real-addressing facility is installed, bit positions 13 and 14 of the page-table entry for 4K-byte pages are used as bits 6 and 7 of the page-frame real address and do not cause a translation-specification exception when either bit is one.

Formation of the Real Address

When no exceptions in the translation process are encountered, the page-frame real address obtained from the page-table entry and the byte-index portion of the virtual address are concatenated, with the page-frame real address forming the leftmost part. The result is the real storage address which corresponds to the virtual address.

Recognition of Exceptions during Translation

Invalid addresses and invalid formats can cause exceptions to be recognized during the translation process. Exceptions are recognized when information contained in control registers or table entries is used for translation and is found to be incorrect.

The information pertaining to DAT is considered to be used when an instruction is executed with DAT on or when INVALIDATE PAGE TABLE ENTRY or LOAD REAL ADDRESS is executed. The information is not considered to be used when the PSW specifies DAT on but an I/O, external, restart, or machine-check interruption occurs before an instruction is executed, or when the PSW specifies the wait state. Only that information required in order to translate a virtual address is considered to be in use during the translation of that address, and, in particular, addressing exceptions that would be caused by the use of the PSTD or the SSTD are not recognized when the translation of an address uses only the SSTD or only the PSTD, respectively.

A list of translation exceptions, with the action taken for each exception and the priority in which the exceptions are recognized when more than one is applicable, is provided in the section "Recognition of Access Exceptions" in Chapter 6, "Interruptions."

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 DAT-table entry in real storage only for the initial access to that entry. This information may be placed in the TLB, and subsequent translations may be performed by using the information in the TLB. The presence of the TLB affects the translation process to the extent that a modification of the contents of a table entry in real storage does not necessarily have an immediate effect, if any, on the translation.

The size and the structure of the TLB depend on the model. For instance, the TLB may be implemented in such a way as to contain only a few entries pertaining to the currently designated segment table, each entry consisting of the leftmost portion of a virtual address and its corresponding page-frame real address and segment-protection bit; or it may contain arrays of values where the page-frame real address and segment-protection bit are selected on the basis of the effective segment-table origin, the translation format, and the leftmost bits of the virtual address. Entries within the TLB are not explicitly addressable by the program. In a multiple-CPU configuration, each CPU has its own TLB.

The description of the logical structure of the TLB covers all implementations by System/370 models. The TLB entries are considered as being of two types: TLB segment-table entries and TLB page-table entries. A TLB entry is considered as containing within it both the information obtained from the table entry in real storage and the attributes used to fetch the entry from storage. Thus, a TLB segment-table entry would contain the following fields:

TF	STO	SX	PTO	PTL	C	P

TF The translation format in effect when the entry was formed

STO The segment-table origin in effect when the entry was formed

SX The segment index used to select the entry

PTO The page-table origin fetched from the segment-table entry in real storage

PTL The page-table length fetched from the segment-table entry in real storage

C The common-segment bit fetched from the segment-table entry in real storage; when the common-segment facility is not installed, this field is not included in the TLB

P The segment-protection bit fetched from the segment-table entry in real storage; when the segment-protection facility is not installed, this field is not included in the TLB.

A TLB page-table entry would contain the following fields:

TF	PTO	PX	PFRA

TF The translation format in effect when the entry was formed

PTO The page-table origin in effect when the entry was formed

PX The page index used to select the entry

PFRA The page-frame real address fetched from the page-table entry in real storage. When the extended-real-addressing facility is installed, this field for 4K-byte pages includes the extended-storage-address bits.

Depending on the implementation, not all of the above items are required in the TLB. For example, if the implementation combines into a single TLB entry (1) the information obtained from a page-table entry and (2) the attributes of both the page-table entry and the segment-table entry, then the page-table-origin and page-table-length fields are not required.

Note: 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 they 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 be cleared under conditions additional to those for which clearing is mandatory.

Use of the Translation-Lookaside Buffer

The formation of TLB entries and the effect of any manipulation of the contents of a table entry in real storage by the program depend on whether the entry is valid, on whether the entry is attached to a particular CPU, on whether a copy of the entry can be placed in the TLB of a particular CPU, and on whether a copy in the TLB of the entry is usable.

The valid state of a table entry 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.

The attached state of a table entry denotes that the CPU to which it is attached can attempt to use the table entry for implicit address translation. The table entry may be attached to more than one CPU at a time. When a table entry is described as attached, the term "to a CPU" is implied.

The usable state of a TLB entry denotes that the CPU can attempt to use the TLB entry for implicit address translation. Also, the usable state of a TLB segment-table entry is a factor in determining whether a page-table entry is attached.

A segment-table entry or a page-table entry may be placed in the TLB only when the entry is attached and valid and would not cause a translation-specification exception if used for translation. Except for these restrictions, the entry may be placed in the TLB at any time.

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

1. The current PSW specifies DAT on.

2. The current PSW contains no errors which would cause an early exception to be recognized. Those machines without DAS installed do not necessarily comply with this condition.

3. The current translation format, bits 8-12 in control register 0, is valid.

4. The entry meets the requirements in a or b below.

   a. The entry is within the segment table designated by the primary segment-table designation in control register 1.

   b. The entry is within the segment table designated by the secondary segment-table designation in control register 7 and either of the following requirements is met:

      - The CPU is in the secondary-space mode.

      - The secondary-space control, bit 5 of control register 0, is one.

5. The entry can be selected by the segment-index portion of a virtual address.

A page-table entry is attached when it is within the page table designated by either a usable TLB segment-table entry or by an attached and valid segment-table entry which would not cause a translation-specification exception if used for translation.

A TLB segment-table entry is in the usable state when all of the following conditions are met:

1. The current PSW specifies DAT on.

2. The current PSW contains no errors which would cause an early exception to be recognized. Those machines without DAS installed do not necessarily comply with this condition.

3. The translation-format field in the TLB segment-table entry is the same as the current translation format.

4. The TLB segment-table entry meets at least one of the following requirements:

   - The common-segment bit is one in the TLB entry.

   - The segment-table-origin field in the TLB entry is the same as the current PSTO.

   - The segment-table-origin field in the TLB entry is the same as the current SSTO, and either PSW bit 16 is one or bit 5 of control register 0 is one.

A TLB segment-table entry may be used for implicit address translation only when the entry is in the usable state, the segment index of the entry matches the segment index of the virtual address to be translated, and either the common-segment bit is one in the TLB entry or the segment-table-origin field in the TLB entry matches the segment-table origin used to select it.

A TLB page-table entry is in the usable state when all of the following conditions are met:

1. The TLB page-table entry is selected by a usable TLB segment-table entry or by an attached and valid segment-table entry which would not cause a translation-specification exception if used for translation.

2. The page-table-origin field in the TLB page-table entry matches the page-table-origin field in the segment-table entry which selects it.

3. The page-index field in the TLB page-table entry is within the range permitted by the page-table-length field in the segment-table entry which selects it.

4. The translation-format field in the TLB page-table entry is the same as the current translation format.

A TLB page-table entry may be used for implicit address translation only when the TLB entry is in the usable state as selected by the segment-table entry being used and only when the page index of the TLB page-table entry matches the page index of the virtual address being translated.

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 real storage.

Selected page-table entries are cleared from the TLB by means of the INVALIDATE PAGE TABLE ENTRY instruction. All information in the TLB is necessarily cleared only by execution of PURGE TLB, SET PREFIX, or CPU reset.

Programming Notes

1. Although a table entry may be copied into the TLB only when the table entry is both valid and attached, the copy may remain in the TLB even when the table entry itself is no longer valid or attached.

2. No entries can be copied into the TLB when DAT is off because the table entries at this time are not attached. In particular, translation of the operand address of LOAD REAL ADDRESS, with DAT off, does not cause entries to be placed in the TLB.

   Conversely, when DAT is on, information may be copied into the TLB from all translation-table entries that could be used for address translation, given the current translation parameters, the setting of the address-space-control bit, and the setting of the secondary-space-control bit. 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.

3. 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 origin by which the entry becomes attached.

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

5. The states and use of the DAT entries in both real storage and in the TLB are summarized in the figure "Summary of DAT Entries."

State or Function	Conditions to Be Met
STE is attached by means of PSTD (applies only to STE in storage)	• DAT on • No early PSW exception* • TF valid • STE in segment table defined by PSTD in CR1 • STE selectable by a 24-bit address
STE is attached by means of SSTD (applies only to STE in storage)	• DAT on • No early PSW exception • TF valid • STE in segment table defined by SSTD in CR7 • STE selectable by a 24-bit address • PSW bit 16 one or bit 5 of CR0 one
STE in storage is usable for a particular instance of implicit translation	• STE in segment table defined and attached by STD being used for the translation • STE selected by SX
STE can be placed in TLB	• STE attached • STE I bit zero • No TS
STE in TLB is usable	• DAT on • No early PSW exception* • TF matches • STE selectable by an STD: - C bit one, or - STO matches PSTO, or - STO matches SSTO, and either PSW bit 16 one or bit 5 of CR0 one
STE in TLB is usable for a particular instance of implicit translation	• DAT on • No early PSW exception* • TF matches • STE selected by STD being used for the translation: - STO matches, or - C bit one • SX matches
PTE is attached (applies only to PTE in storage)	• PTE in page table defined by usable STE in the TLB, or defined by an STE that can be placed in the TLB
PTE in storage is usable for a particular instance of implicit translation	• PTE attached by means of STE being used for the translation • PTE selected by PX

Summary of DAT Entries (Part 1 of 2)

State or Function	Conditions to Be Met
PTE can be placed in TLB	• PTE attached • PTE I bit zero • No TS
PTE in TLB is usable	• TF matches • PTE selectable by a usable STE in the TLB or by an STE that can be placed in the TLB: - PTO matches and - PX within PTL
PTE in TLB is usable for a particular instance of implicit translation	• TF matches • PTE selected by STE being used for the translation: - PTO matches and - PX within PTL • PX matches

Explanation:

*	Models which do not have DAS installed do not necessarily comply with the condition.
C bit	Common-segment bit in STE.
I bit	Invalid bit in table entry.
PSTD	Primary segment-table designation.
PSTO	Primary segment-table origin.
PTE	Page-table entry.
PTL	Page-table length.
PTO	Page-table origin.
PX	Page index.
SSTD	Secondary segment-table designation.
SSTO	Secondary segment-table origin.
STD	Segment-table designation.
STE	Segment-table entry.
STO	Segment-table origin.
SX	Segment index.
TF	Translation format (control register 0, bits 8-12).
TS	Translation-specification exception. The condition "No TS" means that attempted use of the associated DAT-table entry would not cause a translation-specification exception.

Summary of DAT Entries (Part 2 of 2)

Modification of Translation Tables

When an attached and invalid table entry is made valid and no usable entry for the associated virtual address is in the TLB, the change takes effect no later than the end of the current unit of operation. Similarly, when an unattached and valid table entry is made attached and no usable entry for the associated virtual address is in the TLB, the change takes effect no later than the end of the current unit of operation.

When a valid and attached table entry is changed, and when, before the TLB is cleared of entries which qualify for substitution for that entry, an attempt is made to refer to storage by using a virtual address requiring that entry for translation, unpredictable results may occur, to the following extent. The use of the new value may begin between instructions or during the execution of an instruction, including the instruction that caused the change. Moreover, until the TLB is cleared of entries which qualify for substitution for that entry, the TLB may contain both the old and the new values, and it is unpredictable whether the old or new value is selected for a particular access. If both old and new values of a segment-table entry are present in the TLB, a page-table entry may be fetched by using one value and placed in the TLB associated with the other value. If the new value of the entry is a value which would cause an exception, the exception may or may not cause an interruption to occur. If an interruption does occur, the result fields of the instruction may be changed even though the exception would normally cause suppression or nullification.

Entries are cleared from the TLB in accordance with the following rules:

1. All entries are cleared from the TLB by the execution of PURGE TLB and SET PREFIX and by CPU reset.

2. Selected entries are cleared from all TLBs in the configuration by the execution of INVALIDATE PAGE TABLE ENTRY by any of the CPUs in the configuration.

3. Some or all TLB entries may be cleared at times other than those required by PURGE TLB, SET PREFIX, CPU reset, and INVALIDATE PAGE TABLE ENTRY.

Programming Notes

1. Entries in the TLB may continue to be used for translation after the table entries from which they have been formed have become unattached or invalid. These TLB entries are not necessarily removed unless explicitly cleared from the TLB.

   A change made to an attached and valid entry or a change made to a table entry that causes the entry to become attached and valid is reflected in the translation process for the next instruction, or earlier than the next instruction, unless a TLB entry qualifies for substitution for that table entry. However, a change made to a table entry that causes the entry to become unattached or invalid is not necessarily reflected in the translation process until the TLB is cleared of entries which qualify for substitution for that table entry.

2. Exceptions associated with dynamic address translation may be established by a pretest for operand accessibility that is performed as part of the initiation of instruction execution. Consequently, a segment-translation or page-translation exception may be indicated when a table entry is invalid at the start of execution even if the instruction would have validated the table entry it uses and the table entry would have appeared valid if the instruction was considered to process the operands one byte at a time.

3. A change made to an attached table entry, except to set the I bit to zero or to alter the rightmost bit of a page-table entry, may produce unpredictable results if that entry is used for translation before the TLB is cleared of all copies of that entry. The use of the new value may begin between instructions or during the execution of an instruction, including the instruction that caused the change. When an instruction, such as MOVE (MVC), makes a change to an attached table entry, including a change that makes the entry invalid, and subsequently uses the entry for translation, a changed entry is being used without a prior clearing of the entry from the TLB, and the associated unpredictability of result values and of exception recognition applies.

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

   As another example, if a segment-table entry is changed to designate a different page table and used without clearing the TLB, then the new page-table entries may be fetched and associated with the old page-table origin. In such a case, execution of INVALIDATE PAGE TABLE ENTRY designating the new page-table origin will not necessarily clear the page-table entries fetched from the new page table.

4. To facilitate the manipulation of translation tables, INVALIDATE PAGE TABLE ENTRY is provided, which sets the I bit in a page-table entry to one and clears all TLBs in the configuration of entries formed from that table entry.

   INVALIDATE PAGE TABLE ENTRY is useful for setting the I bit to one in a page-table entry and causing TLB copies of the entry to be cleared from the TLB of each CPU in the configuration. The following aspects of the TLB operation should be considered when using INVALIDATE PAGE TABLE ENTRY. (See also the programming notes following INVALIDATE PAGE TABLE ENTRY.)

   a. INVALIDATE PAGE TABLE ENTRY should be executed before making any change to a page-table entry other than changing the rightmost bit; otherwise, the selective clearing portion of INVALIDATE PAGE TABLE ENTRY may not clear the TLB copies of the entry.

   b. Invalidation of all the page-table entries within a page table by means of INVALIDATE PAGE TABLE ENTRY does not necessarily clear the TLB of the copies, if any, of the segment-table entry designating the page table. When it is desired to invalidate and clear the TLB of a segment-table entry, the rules in note 5 below must be followed.

   c. When a large number of page-table entries are to be invalidated at a single time, the overhead involved in using PURGE TLB and in following the rules in note 5 below may be less than in issuing INVALIDATE PAGE TABLE ENTRY for each page-table entry.

5. Manipulation of table entries should be in accordance with the following rules. If these rules are complied with, translation is performed as if the table entries from real storage were always used in the translation process.

   a. A valid table entry must not be changed while it is attached to any CPU except either to invalidate the entry, by using INVALIDATE PAGE TABLE ENTRY or to alter bit 15 of a page-table entry.

   b. When any change is made to a table entry other than a change to bit 15 of a page-table entry, each CPU which may have a TLB entry formed from that entry must execute PURGE TLB or SET PREFIX or perform CPU reset, after the change occurs and prior to the use of that entry for implicit translation by that CPU, except that the purge is unnecessary if the change was made by using INVALIDATE PAGE TABLE ENTRY.

   c. When any change is made to an invalid table entry in such a way as to allow intermediate valid values to appear in the entry, each CPU to which the entry is attached must execute PURGE TLB or SET PREFIX or perform CPU reset, after the change occurs and prior to the use of the entry for implicit address translation by that CPU.

   d. When any change is made to a segment-table or page-table length, each CPU to which that table has been attached must execute PTLB after the length has been changed but before that table becomes attached again to the CPU.

   Note that when an invalid page-table entry is made valid without introducing intermediate valid values, the TLB need not be cleared in a CPU which does not have any usable TLB copies for that entry. Similarly, when an invalid segment-table entry is made valid without introducing intermediate valid values, the TLB need not be cleared in a CPU which does not have any usable TLB copies for that segment-table entry and which does not have any usable TLB copies for the page-table entries attached by it.

   The execution of PURGE TLB and SET PREFIX may have an adverse effect on the performance of some models. Use of these instructions should, therefore, be minimized in conformity with the above rules.

<u>ADDRESS SUMMARY</u>

ADDRESSES TRANSLATED

Most addresses that are explicitly specified by the program and are used by the CPU to refer to storage for an instruction or an operand are logical addresses and are subject to implicit translation when DAT is on. Analogously, the corresponding addresses indicated to the program on an interruption or as the result of executing an instruction are logical. The operand address of LOAD REAL ADDRESS is explicitly translated, regardless of whether the PSW specifies the EC mode or BC mode, and regardless of whether the EC-mode PSW specifies DAT on or off.

Translation is not applied to quantities that are formed from the values specified in the B and D fields of an instruction but that are not used to address storage. This includes operand addresses in LOAD ADDRESS, MONITOR CALL, and the shifting and I/O instructions. This also includes the addresses in control registers 10 and 11 designating the starting and ending locations for PER.

With the exception of INSERT VIRTUAL STORAGE KEY and TEST PROTECTION, the addresses explicitly designating storage keys (operand addresses in SET STORAGE

KEY, INSERT STORAGE KEY, RESET REFERENCE BIT, SET STORAGE KEY EXTENDED, INSERT STORAGE KEY EXTENDED, and RESET REFERENCE BIT EXTENDED) are real addresses. Similarly, the addresses implicitly used by the CPU or channels for such sequences as interruptions, updating the interval timer at locations 80-83, DAT-table references, and logout, including the machine-check-extended-logout address in control register 15, are real addresses.

The addresses used by channels to transfer data and to refer to CCWs or IDAWs are absolute addresses. Similarly, the I/O-extended-logout address at locations 173-175 is an absolute address.

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

The processing of addresses, including dynamic address translation and prefixing, is discussed in the section "Address Types" in this chapter. 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 the section "Prefixing" in this chapter.

HANDLING OF ADDRESSES

The handling of addresses is summarized in the figure "Handling of Addresses." This figure lists all addresses that are encountered by the program and specifies the address type.

Virtual Addresses

- Address of storage operand for INSERT VIRTUAL STORAGE KEY
- Operand address in LOAD REAL ADDRESS
- Addresses of storage operands for MOVE TO PRIMARY and MOVE TO SECONDARY
- Address stored in the word at real location 144 on a program interruption for page-translation or segment-translation exception

Instruction Addresses

- Instruction address in PSW
- Branch address
- Target of EXECUTE
- Address stored in the word at real location 152 on a program interruption for PER
- Address placed in general register by BRANCH AND LINK, BRANCH AND SAVE, and PROGRAM CALL

Logical Addresses

- Addresses of storage operands for instructions not otherwise specified
- Address placed in general register 1 by EDIT AND MARK and TRANSLATE AND TEST
- Addresses in general registers updated by MOVE LONG and COMPARE LOGICAL LONG

Real Addresses

- Address of storage key for INSERT STORAGE KEY, INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY, and SET STORAGE KEY EXTENDED
- Address of storage operand for TEST BLOCK
- Address of storage operand for READ DIRECT and WRITE DIRECT when INVALIDATE PAGE TABLE ENTRY is installed
- Page-table origin in INVALIDATE PAGE TABLE ENTRY
- Segment-table origin in control registers 1 and 7
- Page-table origin in segment-table entry
- Page-frame real address in page-table entry
- MCEL address in control register 15
- The translated address generated by LOAD REAL ADDRESS
- Address of segment-table entry or page-table entry provided by LOAD REAL ADDRESS
- ASN-first-table origin in control register 14
- ASN-second-table origin in ASN-first-table entry
- Authority-table origin in ASN-second-table entry
- Linkage-table origin in control register 5
- Entry-table origin in linkage-table entry

Handling of Addresses (Part 1 of 2)

Permanently Assigned Real Addresses

- Addresses of PSWs, interruption codes, and associated information used during interruption
- Address used by CPU to update interval timer in the word at real location 80
- Addresses 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

- Prefix value
- CCW address in CAW
- Data address in CCW
- IDAW address in a CCW specifying indirect-data addressing
- CCW address in a CCW specifying transfer in channel
- Data address in IDAW
- IOEL address at real locations 173-175
- Failing-storage address stored in the word at real location 248
- CCW address in CSW

Permanently Assigned Absolute Addresses

- Addresses of PSW and first two CCWs used for initial program loading
- Addresses used for the store-status function

Addresses Not Used to Reference Storage

- PER starting address in control register 10
- PER ending address in control register 11
- Address stored in the word at real location 156 for a monitor event
- Address in shift instructions and other instructions specified not to use the address to reference storage

Handling of Addresses (Part 2 of 2)

ASSIGNED STORAGE LOCATIONS

The figure "Assigned Storage Locations" shows the format and extent of the assigned locations in storage. The locations are used as follows. Unless specifically noted, the usage applies to both the BC and EC modes.

0-7 (Absolute Address)

Initial-Program-Loading PSW: The first eight bytes read during the initial-program-loading (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, and bytes 2 and 3 hold the I/O address at the conclusion of an IPL in the BC mode.

0-7 (Real Address)

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

8-15 (Absolute Address)

Initial-Program-Loading CCW1: Bytes 8-15 read during the initial-program-loading (IPL) initial-read operation are stored at locations 8-15. The contents of these locations are ordinarily used as the next CCW in an IPL CCW chain after completion of the IPL initial-read operation.

8-15 (Real Address)

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

16-23 (Absolute Address)

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

24-31 (Real Address)

External Old PSW: The current PSW is stored as the old PSW at locations 24-31 during an external interruption.

32-39 (Real Address)

Supervisor-Call Old PSW: The current PSW is stored as the old PSW at locations 32-39 during a supervisor-call interruption.

40-47 (Real Address)

Program Old PSW: The current PSW is stored as the old PSW at locations 40-47 during a program interruption.

48-55 (Real Address)

Machine-Check Old PSW: The current PSW is stored as the old PSW at locations 48-55 during a machine-check interruption.

56-63 (Real Address)

Input/Output Old PSW: The current PSW is stored as the old PSW at locations 56-63 during an I/O interruption.

64-71 (Real Address)

CSW: The channel-status word (CSW) is stored at locations 64-71 during an I/O interruption. Part or all of it may be stored during the execution of CLEAR I/O, HALT DEVICE, HALT I/O, START I/O, START I/O FAST RELEASE, STORE CHANNEL ID, or TEST I/O, in which case condition code 1 is set.

72-75 (Real Address)

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 (Real Address)

Interval Timer: Locations 80-83 contain the interval timer. The interval timer is updated whenever the CPU is in the operating state and the manual interval-timer control is set to enable.

84-87 (Logical Address)

Trace-Table-Designation Word: The DAS-trace-control bit and the trace-table-entry-header origin are fetched from locations 84-87.

88-95 (Real Address)

External New PSW: The new PSW is fetched from locations 88-95 during an external interruption.

96-103 (Real Address)

Supervisor-Call New PSW: The new PSW is fetched from locations 96-103 during a supervisor-call interruption.

104-111 (Real Address)

Program New PSW: The new PSW is fetched from locations 104-111 during a program interruption.

112-119 (Real Address)

Machine-Check New PSW: The new PSW is fetched from locations 112-119 during a machine-check interruption.

120-127 (Real Address)

Input/Output New PSW: The new PSW is fetched from locations 120-127 during an I/O interruption.

128-131 (Real Address)

External-Interruption Parameter: During an external interruption due to service signal, the parameter associated with the interruption is stored at locations 128-131.

132-133 (Real Address)

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

134-135 (Real Address)

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

136-139 (Real Address)

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 location 137.

140-143 (Real Address)

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 location 141.

144-147 (Real Address)

Translation-Exception Identification: During a program interruption due to a segment-translation exception or a page-translation exception, the segment-index and page-index portion of the virtual address causing the exception is stored at locations 144-147. This address is sometimes referred to as the translation-exception address. When 2K-byte pages are used, the rightmost 11 bits of the address are unpredictable. When 4K-byte pages are used, the rightmost 12 bits of the address are unpredictable. Bits 1-7 of location 144 are set to zeros. When DAS is installed, bit 0 of location 144 is set to zero if the translation was relative to the primary segment table designated by control register 1, or it is set to one if the translation was relative to the secondary segment table designated by control register 7. When DAS is not installed, bit 0 of location 144 is set to zero.

During a program interruption due to an AFX-translation, ASX-translation, primary-authority, or secondary-authority exception, the ASN being translated is stored at locations 146-147. Zeros are stored at locations 144-145.

During a program interruption for a space-switch event, the old PASN, which is in bits 16-31 of control register 4 before the execution of a space-switching PROGRAM CALL or PROGRAM TRANSFER instruction, is stored at locations 146-147. The old space-switch-event-control bit is stored in bit position 0, and zeros are stored in bit positions 1-15 of locations 144-145.

During a program interruption due to an LX-translation or EX-translation exception, the PC number is stored in bit positions 12-31 of the word at locations 144-147. Bits 0-11 are set to zeros.

In all cases, storing at locations 144-147 only occurs when the old PSW specifies the EC mode.

148-149 (Real Address)

Monitor-Class Number: During a program interruption due to a monitor event, the monitor-class number is stored at location 149, and zeros are stored at location 148.

150-151 (Real Address)

PER Code: During a program interruption due to a PER event, the PER code is stored in bit positions 0-3 of location 150. Zeros are stored in bit positions 4-7 of location 150 and bit positions 0-7 of 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 (Real Address)

PER Address: During a program interruption due to a program event, the 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 (Real Address)

Monitor Code: During a program interruption due to a monitor event, the monitor code is stored at locations 157-159, and zeros are stored at location 156.

168-171 (Real Address)

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

172-175 (Real Address)

IOEL Address: The I/O-extended-logout address is fetched from locations 173-175 during the I/O-extended-logout operation. The contents of location 172 are ignored.

176-179 (Real Address)

Limited Channel Logout: The limited-channel-logout information is stored at locations 176-179. This field may be stored only when the CSW or a portion of the CSW is stored.

185 (Real Address)

Measurement Byte: During an I/O interruption in the EC mode, the measurement byte is stored at location 185. A nonzero value for the measurement byte is part of the start-I/O-fast-queuing facility. When this facility is not installed, zeros are stored.

186-187 (Real Address)

I/O Address: During an I/O interruption in the EC mode and at the conclusion of an IPL in the EC mode, the I/O address is stored at locations 186-187.

216-223 (Absolute Address)

Store-Status CPU-Timer Save Area: During the execution of the store-status operation, the contents of the CPU timer, if the CPU-timer and clock-comparator facility is installed, are stored at locations 216-223.

216-223 (Real Address)

Machine-Check CPU-Timer Save Area: During a machine-check interruption, the contents of the CPU timer, if the CPU-timer and clock-comparator facility is installed, are stored at locations 216-223.

224-231 (Absolute Address)

Store-Status Clock-Comparator Save Area: During the execution of the store-status operation, the contents of the clock comparator, if the CPU-timer and clock-comparator facility is installed, are stored at locations 224-231.

224-231 (Real Address)

Machine-Check Clock-Comparator Save Area: During a machine-check interruption, the contents of the clock comparator, if the CPU-timer and clock-comparator facility is installed, are stored at locations 224-231.

232-239 (Real Address)

Machine-Check-Interruption Code: During a machine-check interruption, the machine-check-interruption code is stored at locations 232-239.

244-247 (Real Address)

External-Damage Code: During a machine-check interruption due to certain external-damage conditions, depending on the model, an external-damage code may be stored at locations 244-247.

248-251 (Real Address)

Failing-Storage Address: During a machine-check interruption, a failing-storage address may be stored at locations 249-251, with zeros stored at location 248. When the extended-real-address facility is installed, the failing-storage address is 31 bits and bit 0 of location 248 is set to zero.

252-255 (Real Address)

Region Code: During a machine-check interruption, model-dependent information may be stored at locations 252-255.

256-263 (Absolute Address)

Store-Status PSW Save Area: During the execution of the store-status operation, the contents of the current PSW are stored at locations 256-263.

256-351 (Real Address)

Fixed-Logout Area: Depending on the model, logout information may be stored at locations 256-351 during a machine-check interruption or full channel logout. 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.

264-267 (Absolute Address)

Store-Status Prefix Save Area: During the execution of the store-status operation, the contents of the prefix register, if the multiprocessing facility is installed, are stored at locations 264-267.

268-271 (Absolute Address)

Store-Status Model-Dependent Save Area: During the execution of the store-status operation, model-dependent information may be stored at locations 268-271.

352-383 (Absolute Address)

Store-Status Floating-Point-Register Save Area: During the execution of the store-status operation, the contents of the floating-point registers, if the floating-point facility is installed, are stored at locations 352-383.

352-383 (Real Address)

Machine-Check Floating-Point-Register Save Area: During a machine-check interruption, the contents of the floating-point registers, if the floating-point facility is installed, are stored at locations 352-383.

384-447 (Absolute Address)

Store-Status General-Register Save Area: During the execution of the store-status operation, the contents of the general registers are stored at locations 384-447.

384-447 (Real Address)

Machine-Check General-Register Save Area: During a machine-check interruption, the contents of the general registers are stored at locations 384-447.

448-511 (Absolute Address)

Store-Status Control-Register Save Area: During the execution of the store-status operation, the contents of the control registers are stored at locations 448-511.

448-511 (Real Address)

Machine-Check Control-Register Save Area: During a machine-check interruption, the contents of the control registers are stored at locations 448-511.

795 (Logical Address)

CPU Identity for DAS Tracing: During execution of DAS tracing, the contents of location 795 are fetched and placed in the trace entry. This field is called "CPU identity" because the control program is expected to place the rightmost eight bits of the CPU address in this area.

Hex	Dec	
0	0	Initial-Program-Loading PSW; or Restart New PSW
4	4	
8	8	Initial-Program-Loading CCW1; or Restart Old PSW
C	12	
10	16	Initial-Program-Loading CCW2
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	Trace-Table-Designation Word
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	

Assigned Storage Locations (Part 1 of 3)

Hex	Dec				
80	128	External-Interruption Parameter			
84	132	CPU Address	External-Interruption Code		
88	136	0 0 0 0 0 0 0 0 0 0 0 0 0	ILC	0	SVC-Interruption Code
8C	140	0 0 0 0 0 0 0 0 0 0 0 0 0	ILC	0	Program-Interruption Code
90	144	Translation-Exception Identification			
94	148	Monitor-Class Number	PER Cde	0 0 0 0 0 0 0 0 0 0 0 0	
98	152	PER Address			
9C	156	Monitor Code			
A0	160				
A4	164				
A8	168	Channel ID			
AC	172	IOEL Address			
B0	176	Limited Channel Logout			
B4	180				
B8	184		Measurement Byte	I/O Address	
BC	188				
C0	192				
C4	196				
C8	200				
CC	204				
D0	208				
D4	212				
D8	216	Store-Status CPU-Timer Save Area; or Machine-Check CPU Timer Save Area			
DC	220				
E0	224	Store-Status Clock-Comparator Save Area; or Machine-Check Clock-Comparator Save Area			
E4	228				
E8	232	Machine-Check-Interruption Code			
EC	236				
F0	240				
F4	244	External-Damage Code			
F8	248	Failing-Storage Address			
FC	252	Region Code			

Assigned Storage Locations (Part 2 of 3)

Hex	Dec	
100	256	Store-Status PSW Save Area; or Fixed-Logout Area (Part 1)
104	260	
108	264	Store-Status Prefix Save Area; or Fixed-Logout Area (Part 2)
10C	268	Store-Status Mod-Dep Save Area; or Fixed-Logout Area (Part 3)
110	272	Fixed-Logout Area (Part 4)
158	344	
15C	348	
160	352	Store-Status Floating-Point-Register Save Area; or Machine-Check Floating-Point-Register Save Area
164	356	
17C	380	
180	384	Store-Status General-Register Save Area; or Machine-Check General-Register Save Area
184	388	
1BC	444	
1C0	448	Store-Status Control-Register Save Area; or Machine-Check Control-Register Save Area
1C4	452	
1FC	508	
200	512	
314	788	
318	792	CPU Identity
31C	796	

Assigned Storage Locations (Part 3 of 3)

CHAPTER 4. CONTROL

This chapter describes in detail the facilities for controlling, measuring, and recording the operation of one or more CPUs.

STOPPED, OPERATING, LOAD, AND CHECK-STOP STATES

The stopped, operating, load, and check-stop states are four mutually exclusive states of the CPU. When the CPU is in the stopped state, instructions and interruptions, other than the restart interruption, are not executed. In the operating state, the CPU executes instructions and takes interruptions, subject to the control of the program-status word (PSW) and control registers, and in the manner specified by the setting of the operator-facility rate control. The CPU is in the load state during the initial-program-loading operation. The CPU enters the check-stop state only as the result of machine malfunctions.

A change between these four CPU states can be effected by use of the operator facilities or by acceptance of certain SIGNAL PROCESSOR orders addressed to that CPU. The states are not controlled or identified by bits in the PSW. The stopped, load, and check-stop states are indicated to the operator by means of the manual indicator, load indicator, and check-stop indicator, respectively. These three indicators are off when the CPU is in the operating state.

The CPU timer is updated when the CPU is in the operating state or the load state. The TOD clock is not affected by the state of any CPU. The interval timer is updated only when the CPU is in the operating state.

STOPPED STATE

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

- The stop key is activated while the CPU is in the operating state.

- The CPU accepts a stop or stop-and-store-status order specified by a SIGNAL PROCESSOR instruction addressed to this CPU while it is in the operating state.

- The CPU has finished the execution of a unit of operation initiated by performing the start function with the rate control set to the instruction-step position.

When the stop function is performed, the transition from the operating to the stopped state occurs at the end of the current unit of operation. When the wait-state bit of the PSW is one, the transition takes place immediately, provided no interruptions are pending for which the CPU is enabled. In the case of interruptible instructions, the amount of data processed in a unit of operation depends on the particular instruction and may depend on the model.

Before entering the stopped state by means of the stop function, all pending allowed interruptions 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. While the CPU is in the stopped state, interruption conditions remain pending.

The CPU is also placed in the stopped state when:

- The CPU reset is completed. However, when the reset operation is performed as part of initial program loading for this CPU, then the CPU is placed in the load state and does not necessarily enter the stopped state.

- An address comparison indicates equality and stopping on the match is specified.

The execution of resets is described in the section "Resets" in this chapter, and address comparison is described in the section "Address-Compare Controls" in Chapter 12, "Operator Facilities."

If the CPU is in the stopped state when an INVALIDATE PAGE TABLE ENTRY instruction is executed on another CPU in the configuration, the invalidation may be performed immediately or may be delayed until the CPU leaves the stopped state.

OPERATING STATE

The CPU changes from the stopped state to the operating state by means of the start function or when a restart interruption (see Chapter 6) occurs.

The start function is performed if the CPU is in the stopped state and (1) the start key associated with that CPU is activated or (2) that CPU accepts the start order specified by a SIGNAL PROCESSOR instruction addressed to that CPU. The effect of performing the start function is unpredictable when the stopped state has been entered by means of a reset.

When the rate control is set to the process position and the start function is performed, the CPU starts operating at normal speed. When the rate control is set to the instruction-step position and the wait-state bit is zero, one instruction or, for interruptible instructions, one unit of operation is executed, and all pending allowed interruptions are taken before the CPU returns to the stopped state. When the rate control is set to the instruction-step position and the wait-state bit is one, the start function causes no instruction to be executed, but all pending allowed interruptions are taken before the CPU returns to the stopped state.

LOAD STATE

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

CHECK-STOP STATE

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

Programming Notes

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

2. When, because of a machine malfunction, an invalid address in the prefix register, or an incomplete READ DIRECT instruction, the CPU is unable to end the execution of an instruction, the stop function is ineffective, and a reset function has to be invoked instead. A similar situation occurs when an unending string of interruptions results from a PSW with a PSW-format error of the type that is recognized early, or from a persistent interruption condition, such as one due to the CPU timer.

3. Pending I/O operations may be initiated, and active I/O operations continue to suspension or completion, after the CPU enters the stopped state. The interruption conditions due to suspension or completion of I/O operations remain pending when the CPU is in the stopped state.

PROGRAM-STATUS WORD

The current program-status word (PSW) in the CPU contains information required for the execution of the currently active program. The PSW is 64 bits in length and includes the instruction address, condition code, and other control fields. In general, the PSW is used to control instruction sequencing and to hold and indicate much of the status of the CPU in relation to the program currently being executed. Additional control and status information is contained in control registers and permanently assigned storage locations.

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

Control is switched during an interruption of the CPU by storing the current PSW, so as to preserve the status of the CPU, and then loading a new PSW.

Execution of LOAD PSW, or the successful conclusion of the initial-program-loading sequence, introduces a new PSW. The instruction address is updated by sequential instruction execution and replaced by successful branches. Other instructions are provided which operate on a portion of the PSW. The figure "Operations on PSW Fields" summarizes these instructions.

A new or modified PSW becomes active (that is, the information introduced into the current PSW assumes control over the CPU) when the interruption or the execution of an instruction that changes the PSW is completed. The interruption for PER associated with an instruction that changes the PSW occurs under control of the PER mask that is effective at the beginning of the operation.

Bits 0-7 of the PSW are collectively referred to as the system mask.

	System Mask (PSW Bits 0-7)		PSW Key (PSW Bits 8-11)		Condition Code and Program Mask[1]		Problem State (PSW Bit 15)		Address-Space Control[2]	
Instruction	Saved	Set	Saved	Set	Saved	Set	Saved	Set	Saved	Set
BRANCH AND LINK	No	No	No	No	Yes	No	No	No	No	No
INSERT PSW KEY	No	No	Yes	No	No	No	No	No	No	No
INSERT ADDRESS SPACE CONTROL	No	No	No	No	No	No	No	No	Yes	No
PROGRAM CALL	No	No	No	No	No	No	Yes	Yes	No	No
PROGRAM TRANSFER	No	No	No	No	No	No	No	Yes[3]	No	No
SET ADDRESS SPACE CONTROL	No	No	No	No	No	No	No	No	No	Yes
SET PROGRAM MASK	No	No	No	No	No	Yes	No	No	No	No
SET PSW KEY FROM ADDRESS	No	No	No	Yes	No	No	No	No	No	No
SET SYSTEM MASK	No	Yes	No	No	No	No	No	No	No	No
STORE THEN AND SYSTEM MASK	Yes	ANDs	No	No	No	No	No	No	No	No
STORE THEN OR SYSTEM MASK	Yes	ORs	No	No	No	No	No	No	No	No

Explanation:

[1] PSW bits 18-23 in the EC mode; PSW bits 34-39 in the BC mode.

[2] Bit 16 of the EC-mode PSW.

[3] Cannot be changed from one to zero.

ANDs The logical AND of the immediate field in the instruction and the current system mask replaces the current system mask.

ORs The logical OR of the immediate field in the instruction and the current system mask replaces the current system mask.

Operations on PSW Fields

EC AND BC MODES

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

Bit 6 of the PSW, in both the BC and EC modes, is the summary-mask bit for controlling I/O interruptions. In addition, I/O interruptions can be controlled individually for up to 32 channels. In the EC mode, the individual control is provided by the 32 mask bits in control register 2, and the summary-mask bit in the PSW applies to all 32 channels. In the BC mode, channels 6 and up are individually controlled by the corresponding bits of control register 2, as well as the summary-mask bit, bit 6 of the PSW. In the BC mode, channels 0-5 are controlled separately by bits 0-5 of the PSW and are not subject to the summary mask or to mask bits in control register 2.

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

The program-mask and condition-code fields in the PSW are allocated to different bit positions in the two control modes.

The instruction INSERT STORAGE KEY provides the reference and change bits when in the EC mode but produces zeros in the corresponding bit positions when in the BC mode. The instruction INSERT STORAGE KEY EXTENDED provides the reference and change bits in both the EC and BC modes.

The following instructions, all of which are associated with the DAS facility, cause a program interruption for special-operation exception if execution is attempted in the BC mode:

EXTRACT PRIMARY ASN
EXTRACT SECONDARY ASN
INSERT ADDRESS SPACE CONTROL
INSERT VIRTUAL STORAGE KEY
MOVE TO PRIMARY
MOVE TO SECONDARY
PROGRAM CALL
PROGRAM TRANSFER
SET ADDRESS SPACE CONTROL
SET SECONDARY ASN

Programming Notes

1. The BC mode provides a PSW format that is compatible with the PSW of System/360.

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

PROGRAM-STATUS-WORD FORMAT IN EC MODE

0	R	0 0 0	T	IO	EX	Key	E	M	W	P	S	0	C C	Prog Mask	0 0 0 0 0 0 0 0
0			5			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

The following is a summary of the functions of the PSW fields in the EC mode. (See the figure "PSW Format in EC Mode.")

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

DAT Mode (T): Bit 5 controls whether implicit dynamic address translation of logical and instruction addresses used to access storage takes place. When the bit is zero, DAT is off, and logical and instruction addresses are treated as real addresses. When the bit is one, DAT is on, and the dynamic-address-translation mechanism is invoked.

I/O Mask (IO): Bit 6 controls whether the CPU is enabled for I/O interruptions. When the bit is zero, an I/O interruption cannot occur. When the bit is one, I/O interruptions are subject to the channel-mask bits in control register 2. When a channel-mask bit is zero, the associated channel cannot cause an I/O interruption; when the channel-mask bit is one, an interruption condition at the channel can cause an interruption. Bit 6 of the EC-mode PSW is provided even when the CPU is not capable of being connected to a channel set.

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

PSW Key: Bits 8-11 form the access key for storage references by the CPU. If the reference is subject to key-controlled protection, the PSW key is matched with a storage key when information is stored or when information is fetched from a location that is protected against fetching. However, for accesses to the second operand of MOVE TO PRIMARY and MOVE WITH KEY, the third operand is used instead of the PSW key. The third operand is also used instead of the PSW key for accesses to the first operand of MOVE TO SECONDARY.

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

Machine-Check Mask (M): Bit 13 controls whether the CPU is enabled for interruption by machine-check conditions. When the bit is zero, a machine-check interruption cannot occur. When the bit is one, machine-check interruptions due to system damage and instruction-processing damage are permitted, but interruptions due to other machine-check-subclass conditions are subject to the subclass-mask bits in control register 14.

Wait State (W): When bit 14 is one, the CPU is waiting; that is, no instructions are processed by the CPU, but interruptions may take place. When bit 14 is zero, instruction fetching and execution occur in the normal manner. The wait indicator is on when the bit is one.

Problem State (P): When bit 15 is one, the CPU is in the problem state. When bit 15 is zero, the CPU is in the supervisor state. In the supervisor state, all instructions are valid. In the problem state, only those instructions are valid that provide meaningful information to the problem program and that cannot affect system integrity; such instructions are called unprivileged instructions. The instructions that are never valid in the problem state are called privileged instructions. When a

CPU in the problem state attempts to execute a privileged instruction, a privileged-operation exception is recognized. Another group of instructions, called semiprivileged instructions, are executed by a CPU in the problem state only if specific authority tests are met; otherwise, a privileged-operation exception or a special-operation exception is recognized.

Address-Space Control (S): Bit 16, in conjunction with PSW bit 5, controls the translation mode. This bit is provided with DAS. See the section "Translation Modes" under "Translation Control" in Chapter 3, "Storage."

Condition Code (CC): Bits 18 and 19 are the two bits of the condition code. The condition code is set to 0, 1, 2, or 3, depending on the result obtained in executing certain instructions. Most arithmatic and logical operations, as well as some other operations, set the condition code. The instruction BRANCH ON CONDITION can specify any selection of the condition-code values as a criterion for branching. A table in Appendix C summarizes the condition-code values that may be set for all instructions which set the condition code of the PSW.

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

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

When the mask bit is one, the exception results in an interruption. When the mask bit is zero, no interruption occurs. The setting of the exponent-underflow-mask bit or the significance-mask bit also determines the manner in which the operation is completed when the corresponding exception occurs. The exponent-underflow and significance mask bits are provided in the PSW even when the floating-point facility is not installed.

Instruction Address: Bits 40-63 form the instruction address. This address designates the location of the leftmost byte of the next instruction to be executed, unless the CPU is in the wait state (bit 14 of the PSW is one).

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

PROGRAM-STATUS-WORD FORMAT IN BC MODE

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

ILC	CC	Prog Mask	Instruction Address	
32	34	36	40	63

PSW Format in BC Mode

The following is a summary of the functions of the PSW fields in the BC mode. (See the figure "PSW Format in BC Mode.")

Channel Masks 0-5: Bits 0-5 control whether the CPU is enabled for I/O interruptions from channels 0-5, respectively. When a bit is zero, the associated channel cannot cause an I/O interruption. When the bit is one, an interruption condition at the channel can cause an I/O interruption. Bits 0-5 of the BC-mode PSW are provided even when the CPU is not capable of being connected to a channel set.

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

External Mask (EX): The meaning of bit 7 is the same as in the EC mode.

PSW Key: The meaning of bits 8-11 is the same as in the EC mode.

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

Machine-Check Mask (M): The meaning of bit 13 is the same as in the EC mode.

Wait State (W): The meaning of bit 14 is the same as in the EC mode.

Problem State (P): The meaning of bit 15 is the same as in the EC mode.

Interruption Code: Bits 16-31 in the old PSW, when stored during a program, supervisor-call, external, or I/O interruption, identify the cause of the interruption. This field is not used or checked in the current PSW. When a new PSW is introduced, the contents of this field are ignored.

Instruction-Length Code (ILC): Bit positions 32 and 33 of the old PSW indicate the length of the last-interpreted instruction when a program or supervisor-call interruption occurs. See the section "Instruction-Length Code" in Chapter 6, "Interruptions." 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. The meaning of the condition code is the same as in the EC mode.

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

The meaning of each mask bit is the same as in the EC mode.

Instruction Address: The meaning of bits 40-63 is the same as in the EC mode.

CONTROL REGISTERS

The control registers provide for maintaining and manipulating control information outside the PSW. There may be up to sixteen 32-bit control registers.

One or more specific bit positions in control registers are assigned to each facility requiring such register space. When the facility is installed, the bits perform the defined control function.

The LOAD CONTROL instruction causes all control-register positions within those registers designated by the instruction to be loaded from storage. The instructions LOAD ADDRESS SPACE PARAMETERS, SET SECONDARY ASN, PROGRAM CALL, and PROGRAM TRANSFER provide specialized functions to place information into certain control-register positions.

Information loaded into the control registers becomes active (that is, assumes control over the system) at the completion of the instruction causing the information to be loaded.

At the time the registers are loaded, the information is not checked for exceptions, such as invalid translation-format 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.

The STORE CONTROL instruction causes all control-register positions, within those registers designated by the instruction, to be placed in storage. The instructions EXTRACT PRIMARY ASN, EXTRACT SECONDARY ASN, and PROGRAM CALL provide specialized functions to obtain information from certain control-register positions. Values corresponding to unassigned or uninstalled register positions are unpredictable.

Only the general structure of the control registers is described here; the definition of a particular control-register position appears in the description of the facility with which the register position is associated. The figure "Assignment of Control-Register Fields" shows the control-register positions which are assigned and the initial value of the field upon execution of initial CPU reset.

Programming Notes

1. The detailed definition of a particular control-register bit position can be located by referring to the entry "control-register assignment" in the Index.

2. To ensure that existing programs operate correctly if and when new facilities using additional control-register positions are installed, the program should load zeros in unassigned control-register positions. Although STORE CONTROL may provide zeros in the bit positions corresponding to unassigned or uninstalled register positions, the program should not depend on such zeros. It is permissible, however, for the program to load into the control registers any information previously stored by means of STORE CONTROL.

Ctrl Reg	Bits	Name of Field	Associated with	Initial Value
0	0	Block-multiplexing control	Block-multiplexing channels	0
0	1	SSM-suppression control	SET SYSTEM MASK	0
0	2	TOD-clock-sync control	Multiprocessing	0
0	3	Low-address-protection control	Low-address protection	0
0	4	Extraction-authority control	Dual-address-space control	0
0	5	Secondary-space control	Dual-address-space control	0
0	7	Storage-key-exception control	Storage-key 4K-byte block	0
0	8-12	Translation format	Dynamic address translation	0
0	14	Vector control[1]	Vector operations	0
0	16	Malfunction-alert subclass mask	Multiprocessing	0
0	17	Emergency-signal subclass mask	Multiprocessing	0
0	18	External-call subclass mask	Multiprocessing	0
0	19	TOD-clock sync-check subclass mask	Multiprocessing	0
0	20	Clock-comparator subclass mask	Clock comparator	0
0	21	CPU-timer subclass mask	CPU timer	0
0	22	Service-signal subclass mask	Service signal	0
0	24	Interval-timer subclass mask	Interval timer	1
0	25	Interrupt-key subclass mask	Interrupt key	1
0	26	External-signal subclass mask	External signals	1
1	0-7	Primary segment-table length	Dynamic address translation	0
1	8-25	Primary segment-table origin	Dynamic address translation	0
1	31	Space-switch-event control	Dual-address-space control	0
2	0-31	Channel masks	Channels	1
3	0-15	PSW-key mask	Dual-address-space control	0
3	16-31	Secondary ASN	Dual-address-space control	0
4	0-15	Authorization index	Dual-address-space control	0
4	16-31	Primary ASN	Dual-address-space control	0
5	0	Subsystem-linkage control	Dual-address-space control	0
5	8-24	Linkage-table origin	Dual-address-space control	0
5	25-31	Linkage-table length	Dual-address-space control	0
7	0-7	Secondary segment-table length	Dual-address-space control	0
7	8-25	Secondary segment-table origin	Dual-address-space control	0

Assignment of Control-Register Fields (Part 1 of 2)

Ctrl Reg	Bits	Name of Field	Associated with	Initial Value
8	16-31	Monitor masks	MONITOR CALL	0
9	0	Successful-branching-event mask	Program-event recording	0
9	1	Instruction-fetching-event mask	Program-event recording	0
9	2	Storage-alteration-event mask	Program-event recording	0
9	3	GR-alteration-event mask	Program-event recording	0
9	16-31	PER 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 control	Machine-check handling	1
14	2	I/O-extended-logout control	I/O extended logout	0
14	4	Recovery subclass mask	Machine-check handling	0
14	5	Degradation subclass mask	Machine-check handling	0
14	6	External-damage subclass mask	Machine-check handling	1
14	7	Warning subclass mask	Machine-check handling	0
14	8	Asynchronous-MCEL control	Machine-check handling	0
14	9	Asynchronous-fixed-log control	Machine-check handling	0
14	12	ASN-translation control	Dual-address-space control	0
14	20-31	ASN-first-table origin	Dual-address-space control	0
15	8-28	MCEL address	Machine-check handling	512[2]

Explanation:

Bits 13, 30, and 31 of control register 0, and bits 0-30 of control register 6 are assigned to functions not described in this publication. The remaining fields not listed are unassigned. The initial value for all unlisted control-register positions is zero.

[1] Bit 14 of control register 0, the vector-control bit, is described in the publication IBM System/370 Vector Operations, SA22-7125.

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

Assignment of Control-Register Fields (Part 2 of 2)

DAS TRACING

Three DAS instructions optionally store 32 bytes of information about the circumstances under which the instructions are executed. This action is called DAS tracing and is performed by placing information in a 32-byte block, called a trace entry, in an area called a trace table. DAS tracing assists in problem determination for privileged and semiprivileged programs by providing an ongoing record in storage of significant events. The trace table and the location of the last-used entry are described by a control block called the trace-table-entry header. The origin of the header is specified in the trace-table-designation word at logical location 84. These relationships are illustrated in the figure "DAS Tracing."

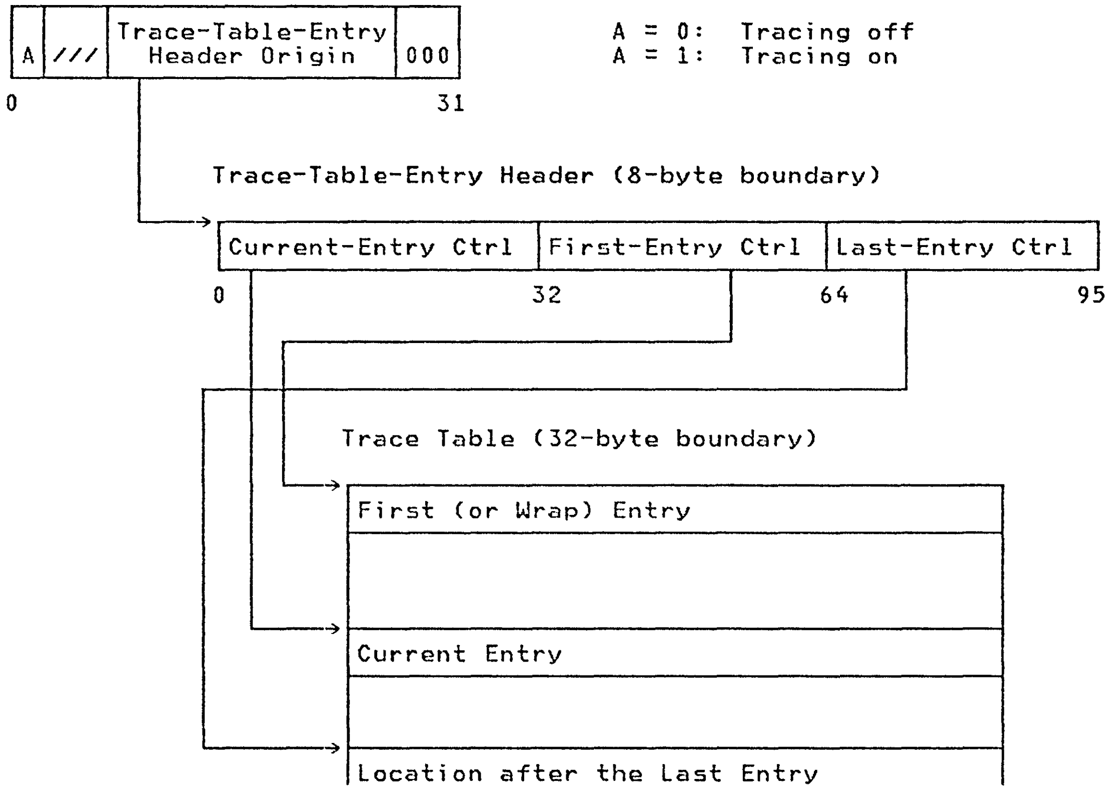
DAS tracing is controlled by bit 0 of the trace-table designation, called the DAS-trace-control bit. When the bit is one, a trace entry is made each time PROGRAM CALL, PROGRAM TRANSFER, or SET SECONDARY ASN is executed.

All locations associated with DAS tracing are treated as logical addresses whose handling depends on the DAT-mode bit and address-space-control bit of the PSW. For PROGRAM CALL and PROGRAM TRANSFER, the addresses are translated by using the old primary segment-table designation. For SET SECONDARY ASN, the addresses are translated by using either the old primary segment-table designation or the old secondary segment-table designation, depending on whether PSW bit 16 specifies the primary-space mode or the secondary-space mode, respectively.

Bits 8-28 of the trace-table designation provide the origin of the three-word trace-table-entry header. Conceptually, the header defines a table of 32-byte elements, called trace entries. The second and third words of the header designate, respectively, the beginning and end of this table. When DAS tracing is on, the first word of the header, called the current-entry control, is updated in conjunction with the execution of the instruction to be traced. The trace entry designated by the updated contents of the current-entry control is used to contain the trace information about the instruction being traced. Updating is interlocked to ensure that distinct entries are produced when a common table is used for tracing by more than one CPU.

Updating the current-entry-control word of the header normally consists in advancing the contents of the current-entry-control word by 32. However, if the advanced value equals or exceeds the value in the last-entry-control word of the header, the contents of the first-entry-control word replace the contents of the current-entry-control word. Thus, the dynamic filling of successive entries wraps from the last entry to the first entry, with no special recognition accorded this event.

Trace-Table Designation
Logical Locations 84-87



DAS Tracing

Protection for DAS Tracing

The references to the trace-table designation, to the trace-table-entry header, and to a trace entry for the purpose of DAS tracing are not subject to key-controlled protection. Low-address protection and segment protection do apply, however, to the store into the current-entry-control word of the header and into a trace entry. Instruction execution is suppressed whenever a protection exception is recognized that is due to DAS tracing.

Other Actions Associated with DAS Tracing

The store accesses made by DAS tracing into the current-entry-control word of the trace-table-entry header and into the trace entry are monitored for PER storage-alteration events. Change recording and reference recording also apply to the storage accesses made by DAS tracing.

Serialization for DAS Tracing

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

TRACE-TABLE DESIGNATION

The trace-table designation is contained in the word at logical location 84 and has the following format.

A	///////	Trace-Table-Entry-Header Origin (logical)	000
0	1	8	29 31

DAS-Trace Control: Bit 0 controls whether implicit tracing is performed for PROGRAM CALL, PROGRAM TRANSFER, and SET SECONDARY ASN. When this bit is zero, no tracing is performed during execution of these instructions. When the bit is one, a trace entry is made each time one of these instructions is executed.

Trace-Table-Entry-Header Origin: Bits 8-28, with three zeros appended on the right, constitute the logical address of the trace-table-entry header.

Bits 1-7 are reserved and should be zeros. They are ignored during implicit tracing.

Bits 29-31 must be zeros if the DAS-trace-control bit is one and execution of PROGRAM CALL, PROGRAM TRANSFER, or SET SECONDARY ASN is attempted; otherwise, a specification exception is recognized.

TRACE-TABLE-ENTRY HEADER

The trace-table-entry header defines a table of 32-byte entries. One entry is filled with information for each traced instruction. After updating, the first word of the header designates the entry in which information is placed for the current instruction. The second and third words of the header designate the beginning and end of the table. The trace-table-entry header has the following format:

Current-Entry Control	First-Entry Control	Last-Entry Control
0	32	64 95

Current-Entry Control: Bits 0-31 are updated to contain the origin of the trace-table entry used for the current instruction.

To update the field, a 32-bit intermediate quantity called the next-entry designator is formed by the logical addition of 32 to the 32-bit contents of the current-entry control, with overflow out of bit 0 ignored. The next-entry designator is then logically compared with the 32-bit contents of the last-entry control. If the next-entry designator is less than the contents of the last-entry control, then the 32-bit next-entry designator replaces the current-entry control. If the next-entry designator is equal to or greater than the contents of the last-entry control, then the 32-bit contents of the first-entry control replace the contents of the current-entry control. A specification exception is recognized if the new value of bits 27-31 would not be zero.

Bits 0-31 are replaced by using a word-concurrent interlocked-update reference. The field is not updated until it is determined that no exceptions would be encountered before the filling of the current trace entry is completed or before the current instruction is completed. This is accomplished by first fetching the contents of the current-entry control, computing the address of the trace entry, and testing the address for access exceptions. If no exceptions would be encountered, the current-entry control is updated by means of a compare-and-swap type of access. If the contents of the location have been changed between the time of the first fetch and the compare-and-swap interlocked update, the new value of the current-entry control is used, and the procedure is repeated.

The new contents of bits 8-26 (called the current-entry origin), with five zero bits appended on the right, constitute the logical address of the trace entry for the current instruction. For the purpose of determining the address of the current entry, the first word of the header has the following format:

////////	Current-Entry Origin (logical)	00000
0	8	27 31

The second and third words of the header are used as follows:

First-Entry Control: Bits 32-63 replace the contents of bit positions 0-31 when the last-entry control disallows tracing in the location following the last-used trace entry.

Last-Entry Control: Bits 64-95 are compared with a derived 32-bit quantity called the next-entry designator. Depending on whether the next-entry designator is (1) less than, or (2) equal to or greater than bits 64-95, bits 0-31 are replaced by using an interlocked-update reference either by (1) the next-entry designator or (2) the contents of bit positions 32-63.

Interlocks

The current-entry-control word is changed by using a word-concurrent interlocked-update reference. The fetches of the first-entry-control and last-entry-control words are word-concurrent and are made without regard to when the interlock on the current-entry-control word is established.

During tracing, the fetches of the first-entry-control word and of the last-entry-control word that are performed in conjunction with updating the current entry-control word are not necessarily interlocked to prevent subsequent storing into these words by other CPUs and by channels.

Programming Notes

1. The last-entry-control word should be thought of as designating the location beyond the last entry in the table. This is because an equal comparison with the last-entry-control value results in wrapping to the first entry.

2. The high-order byte of each word of the header should be set to zero; otherwise, unexpected results can occur. This is because 32 bits participate in the comparison and replacement actions but only 24 bits are used to address the trace entry. Thus, a trace table may wrap from high storage locations to low storage locations, and, depending on high-order bit values, not wrap to the intended beginning of the table.

3. Because current trace information is placed in the location designated by the updated contents of the current-entry-control word, the entry designated before tracing occurs is not used initially, although it may subsequently be used if it is in the range of the table after wrapping.

4. Implicit tracing of SET SECONDARY ASN while in the secondary-space mode requires that the trace-table designation, CPU identity byte, trace-table-entry header, and trace table appear in the secondary space which is current when instruction execution begins.

TRACE ENTRY

A trace entry consists of 32 bytes beginning on a 32-byte boundary. The trace-entry address for the current instruction is formed from bits 8-26 of the updated current-entry-control word of the trace-table-entry header. It is treated as a logical address.

The store-type reference to a trace entry is not necessarily a single-access reference. During the execution of an implicitly traced instruction, another CPU or a channel may observe that an entry, or portions of an entry, are stored more than once. The intermediate results observed may or may not correspond to the final results.

The format of an entry for the instructions PROGRAM CALL, PROGRAM TRANSFER, and SET SECONDARY ASN is shown in the figure "Trace-Entry Formats."

Positions within Trace Entry		Contents of Trace Entry for:		
		PROGRAM CALL	PROGRAM TRANSFER	SET SECONDARY ASN
Bytes 0-1		New PSW, bytes 0-1	New PSW, bytes 0-1	New PSW, bytes 0-1
Byte 2		Hex 90[1]	Hex A0[1]	Hex B0[1]
Bytes 3-7		New PSW, bytes 3-7	New PSW, bytes 3-7	New PSW, bytes 3-7
Bytes 8-9		New PASN	New PASN	PASN
Bytes 10-11		New SASN	0	New SASN
Bytes 12-13		GR14 After	Old PASN	0
Bytes 14-15			0	Old SASN
Bytes 16-19		0	0	0
Byte 20	Bits 0-1 Bits 2-3 Bits 4-7	ILC[2] CC PM	ILC[2] CC PM	ILC[2] CC PM
Byte 21		CPU identity[3]	CPU identity[3]	CPU identity[3]
Bytes 22-23		0	0	0
Bytes 24-27		PC number[4]	0	0
Bytes 28-31		TOD clock, bytes 3-6	TOD clock, bytes 3-6	TOD clock, bytes 3-6

Explanation:

[1] Byte 2 contains the entry-type identifier value. This position is used to uniquely identify the type of event for which the entry is made.

[2] Byte 20 contains the instruction-length code (ILC), condition code (CC), and program mask (PM) of the old PSW. The ILC is always 2.

[3] Byte 21, "CPU identity," is fetched from logical location 795.

[4] Bytes 24-27 for PROGRAM CALL contain eight zero bits appended to the left of the 24-bit effective address specified by the PROGRAM CALL instruction. The rightmost 20 bits constitute the PC number.

Trace-Entry Formats

PROGRAM-EVENT RECORDING

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

- Execution of a successful branch instruction.
- Fetching of an instruction from the designated storage area.
- Alteration of the contents of the designated storage area.
- Alteration of the contents of designated general registers.

The program can selectively specify that one or more of the above types of events be recognized. The information concerning a PER event is provided to the program by means of a program interruption, with the cause of the interruption being identified in the interruption code. PER is only available in the EC mode.

CONTROL-REGISTER ALLOCATION

The information for controlling PER resides in control registers 9, 10, and 11 and has the following format:

Control Register 9

EM		Gen.-Reg. Masks
0	4	16 31

Control Register 10

	Starting Address
0	8 31

Control Register 11

	Ending Address
0	8 31

PER-Event Masks (EM): Bits 0-3 of control register 9 specify which types of events are recognized. 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 types of events be recognized. When a bit is zero, the corresponding type of event is not recognized.

PER General-Register Masks: Bits 16-31 of control register 9 specify which general registers are designated for recognition of the alteration of their contents. The 16 bits, in the sequence of ascending bit numbers, correspond one for one with the 16 registers, in the sequence of ascending register numbers. When a bit is one, the alteration of the associated register is recognized; when it is zero, the alteration of the register is not recognized.

PER Starting Address: Bits 8-31 of control register 10 are the address of the beginning of the designated storage area.

PER Ending Address: Bits 8-31 of control register 11 are the address of the end of the designated storage area.

Programming Notes

1. Models may operate at reduced performance while the CPU is enabled for PER events. In order to ensure that CPU performance is not degraded because of the operation of the PER facility, programs that do not use it should disable the CPU for PER events by setting the PER mask in the EC-mode PSW to zero. No degradation due to PER occurs in the BC mode or when the PER mask in the EC-mode PSW is zero. Disabling of the CPU for PER events in the EC mode by means of the masks in control register 9 does not necessarily prevent performance degradation due to the facility.

2. Some degradation may be experienced on some models every time control registers 9, 10, and 11 are loaded, even when the CPU is disabled for PER events (see the programming note under "Storage-Area Designation").

OPERATION

PER is under control of bit 1 of the EC-mode PSW, the PER mask. When the PER mask, a particular PER-event mask bit, and, for general-register-alteration events, a particular general-register mask bit are all ones, the CPU is enabled for the corresponding type of event; otherwise, it is disabled. In the BC mode, the CPU is disabled for PER events.

An interruption due to a PER event normally occurs after the execution of the instruction responsible for the event. The occurrence of the event does not affect the execution of the instruction, which may be either completed, partially completed, terminated, suppressed, or nullified.

When the CPU is disabled for a particular PER event at the time it occurs, either by the PER mask in the PSW or by the masks in control register 9, the event is not recognized.

A change to the PER mask in the PSW or to the PER control fields in control registers 9, 10, and 11 affects PER starting with the execution of the immediately following instruction. If a PER event occurs during the execution of an instruction which changes the CPU from being enabled to being disabled for that type of event, that PER event is recognized.

PER events may be recognized in a trial execution of an instruction, and subsequently the instruction, DAT-table entries, and operands may be refetched for the actual execution. If any refetched field was modified by another CPU or by a channel between the trial execution and the actual execution, it is unpredictable whether the PER events indicated are for the trial or the actual execution.

For special-purpose instructions that are not described in this publication, the operation of PER may not be exactly as described in this section.

Identification of Cause

A program interruption for PER sets bit 8 of the interruption code to one and places identifying information in real storage locations 150-155. The information stored has the following format:

Locations 150-151:

```
|PERC|000000000000|
0    4           15
```

Locations 152-155:

```
|00000000|       PER Address       |
0        8                        31
```

PER Code (PERC): The occurrence of PER events is indicated by ones in bit positions 0-3 of real location 150, the PER code. The bit position in the PER code for a particular type of event is the same as the bit position for that event in the PER-event-mask field in control register 9. When a program interruption occurs, more than one type of PER event can be concurrently indicated. Additionally, if another program-interruption condition exists, the interruption code for the program interruption may indicate both the PER events and the other condition. Zeros are stored in bit positions 4-7 of location 150 and in bit positions 0-7 of location 151.

PER Address: The PER address at locations 152-155 contains the instruction address used to fetch the instruction in execution when one or more PER events were recognized. When the instruction is the target of EXECUTE, the instruction address used to fetch the EXECUTE instruction is placed in the PER-address field. Zeros are stored in the byte at real location 152.

Instruction Address: The instruction address in the program old PSW is the address of the instruction which would have been executed next, unless another program condition is also indicated, in which case the instruction address is that determined by the instruction ending due to that condition.

ILC: The ILC indicates the length of the instruction designated by the PER address, except when a concurrent specification exception for the PSW introduced by LOAD PSW or a supervisor-call interruption sets an ILC of 0.

When a PER event is recognized during execution of a LOAD PSW or SUPERVISOR CALL instruction which changes CPU operation from the EC mode to the BC mode, the interruption occurs with the old PSW specifying the BC mode and with the interruption code stored in the old PSW. The additional information identifying the PER event is stored in its regular format at real locations 150-155.

Priority of Indication

When a program interruption occurs and more than one PER event has been recognized, all recognized PER events are concurrently indicated in the PER code. Additionally, if another program-interruption condition concurrently exists, the interruption code for the program interruption indicates both the PER condition and the other condition.

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

If a PER event is recognized during the execution of an instruction which also introduces a new PSW with the type of PSW-format error which is recognized early (see the section "Exceptions Associated with the PSW" in Chapter 6, "Interruptions"), both the specification exception and PER are indicated concurrently in the interruption code of the program interruption. However, for a PSW-format error of the type which is recognized late, only PER is indicated in the interruption code. In both cases, the invalid PSW is stored as the program old PSW.

Recognition of a PER event does not normally affect the ending of instruction execution. However, in the following cases, execution of an interruptible instruction is not completed normally:

- When the instruction is due to be interrupted for an asynchronous condition (I/O, external, restart, or repressible machine-check condition), a program interruption for the PER event occurs first, and the other interruptions occur subsequently (subject to the mask bits in the new PSW) in the normal priority order.

- When the stop function is performed, a program interruption indicating the PER event occurs before the CPU enters the stopped state.

- When any program exception is recognized, PER events recognized for that instruction execution are indicated concurrently.

- Depending on the model, in certain situations, recognition of a PER event may appear to cause the instruction to be interrupted prematurely without concurrent indication of a program exception, without an interruption for any asynchronous condition, or without the CPU entering the stopped state.

Programming Notes

1. In the following cases, an instruction can both cause a program interruption for a PER event and change the value of masks controlling an interruption for PER events. The original mask values determine whether a program interruption takes place for the PER event.

   a. The instructions LOAD PSW, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and SUPERVISOR CALL can cause an instruction-fetching event and disable the CPU for PER interruptions. Additionally, STORE THEN AND SYSTEM MASK can cause a storage-alteration event to be indicated. In all these cases, the program old PSW associated with the program interruption for the PER event may indicate that the CPU was disabled for PER events.

   b. An instruction-fetching event may be recognized during execution of a LOAD CONTROL instruction that changes the value of the PER-event masks in control register 9 or the addresses in control registers 10 and 11 controlling indication of instruction-fetching events.

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

3. When a PER interruption occurs during the execution of an interruptible instruction, the ILC indicates the length of that instruction or EXECUTE, as appropriate. When a PER interruption occurs as a result of LOAD PSW or SUPERVISOR CALL, the ILC indicates the length of these instructions or EXECUTE, as appropriate, unless a concurrent specification exception on LOAD PSW calls for an ILC of 0.

4. When a PER interruption is caused by branching, the PER address identifies the branch instruction (or EXECUTE, as appropriate), whereas the old PSW points to the next instruction to be executed. When the interruption occurs during the execution of an interruptible instruction, the PER address and the instruction address in the old PSW are the same.

STORAGE-AREA DESIGNATION

Two types of PER events -- instruction fetching and storage alteration -- involve the designation of an area in storage. The storage area starts at the location designated by the starting address in control register 10 and extends up to and including the location designated by the ending address in control register 11. The area extends to the right of the starting address.

An instruction-fetching event occurs whenever the first byte of an instruction or the first byte of the target of an EXECUTE instruction is fetched from the designated area. A storage-alteration event occurs when a store access is made to the designated area by using an operand address that is defined to be a logical or a virtual address. A storage-alteration event does not occur for a store access made with an operand address defined to be a real address.

The set of addresses designated for instruction-fetching and storage-alteration events wraps around at address 16,777,215; that is, address 0 is considered to follow address 16,777,215. When the starting address is less than the ending address, the area is contiguous. When the starting address is greater than the ending address, the set of locations designated includes the area from the starting address to address 16,777,215 and the area from address 0 to, and including, the ending address. When the starting address is equal to the ending address, only that one location is designated.

Address comparison for instruction-fetching and storage-alteration events is performed by comparing all 24 bits of the virtual, logical or instruction address used for the reference with the starting and ending addresses.

Programming Note

In some models, performance of address-range checking is assisted by means of an extension to each page-table entry in the TLB. In such an implementation, changing the contents of control registers 10 and 11 when the instruction-fetching or storage-alteration-event mask is one, or setting either of these PER-event masks to one, may cause the TLB to be cleared of entries. This degradation may be experienced even when the CPU is disabled for PER events. Thus, when possible, the program should avoid loading control registers 9, 10, or 11.

PER EVENTS

Successful Branching

A successful-branching event occurs whenever one of the following instructions causes branching:

BRANCH AND LINK (BAL, BALR)
BRANCH AND SAVE (BAS, BASR)
BRANCH ON CONDITION (BC, BCR)
BRANCH ON COUNT (BCT, BCTR)
BRANCH ON INDEX HIGH (BXH)
BRANCH ON INDEX LOW OR EQUAL (BXLE)

A successful-branching event also occurs whenever one of the following instructions is completed:

PROGRAM CALL (PC)
PROGRAM TRANSFER (PT)

A successful-branching event causes a PER successful-branching event to be recognized if bit 0 of the PER-event masks is one and the PER mask in the EC-mode PSW is one.

A PER successful-branching event is indicated by setting bit 0 of the PER code to one.

Instruction Fetching

An instruction-fetching event occurs if the first byte of the instruction is fetched from the storage area designated by control registers 10 and 11. An instruction-fetching event also occurs if the first byte of the target of EXECUTE is within the designated storage area.

An instruction-fetching event causes a PER instruction-fetching event to be recognized if bit 1 of the PER-event masks is one and the PER mask in the EC-mode PSW is one.

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

Storage Alteration

A storage-alteration event occurs whenever a CPU, by using a logical or virtual address, makes a store access without an access exception to the storage area designated by control registers 10 and 11.

The contents of storage are considered to have been altered whenever the CPU executes an instruction that causes all or part of an operand or a DAS-trace value to be stored within the designated storage area. Alteration is considered to take place whenever storing is considered to take place for purposes of indicating protection exceptions, except that recognition does not occur for the storing of data by a channel program. (See the section "Recognition of Access Exceptions" in Chapter 6, "Interruptions.") Storing constitutes alteration for PER purposes even if the value stored is the same as the original value.

Implied locations that are referred to by the CPU in the process of (1) interval-timer updating, (2) interruptions, and (3) execution of I/O instructions are not monitored. Such locations include the interval-timer, old-PSW, interruption-code, and CSW locations. These locations, however, are monitored when information is stored there explicitly by an instruction. Similarly, monitoring does not apply to the storing of data by a channel program.

When an interruptible vector instruction which performs storing is interrupted, and PER storage alteration applies to storage locations corresponding to elements due to be changed beyond the point of interruption, PER storage alteration is indicated if any such store actually occurred and may be indicated even if such a store did not occur. PER storage alteration is reported for such locations only if no access exception exists at the time that the instruction is executed.

Storage alteration does not apply to instructions whose operands are specified to be real addresses. Thus, storage alteration does not apply to INVALIDATE PAGE TABLE ENTRY, RESET REFERENCE BIT, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY, SET STORAGE KEY EXTENDED, and TEST BLOCK. When INVALIDATE PAGE TABLE ENTRY is installed, the operand address of READ DIRECT is a real address and storage alteration does not apply. When INVALIDATE PAGE TABLE ENTRY is not installed, the operand address of READ DIRECT is a logical address, and storage alteration does apply.

A storage-alteration event causes a PER storage-alteration event to be recognized if bit 2 of the PER-event masks is one and the PER mask in the EC-mode PSW is one.

A PER storage-alteration event is indicated by setting bit 2 of the PER code to one.

General-Register Alteration

A general-register-alteration event occurs whenever the contents of a general register are replaced.

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

The instructions MOVE LONG and COMPARE LOGICAL LONG are always considered to alter the contents of the four registers specifying the two operands, including the cases where the padding byte is used, when both operands have zero length. However, when condition code 3 is set for MOVE LONG, the general registers containing the operand lengths may or may not be considered as having been altered.

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.

It is unpredictable whether general-register-alteration events are indicated for instructions of the vector facility.

A general-register-alteration event causes a PER general-register-alteration event to be recognized if bit 3 of the PER-event masks is one, the PER mask in the EC-mode PSW is one, and the corresponding bit in the PER general-register mask is one.

The PER general-register-alteration event is indicated by setting bit 3 of the PER code to one.

Programming Note

The following are some examples of general-register alteration:

1. Register-to-register load instructions are considered to alter the register contents even when both operand addresses designate the same register.

2. Addition or subtraction of zero and multiplication or division by one are considered to constitute alteration.

3. Logical and fixed-point shift operations are considered to alter the register contents even for shift amounts of zero.

4. The branching instructions BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL are considered to alter the first operand even when zero is added to its value.

INDICATION OF PER EVENTS CONCURRENTLY WITH OTHER INTERRUPTION CONDITIONS

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

1. The indication of an instruction-fetching 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 halfword 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. Similarly, when an access exception prohibits access to all or a portion of the target of EXECUTE, it is unpredictable whether the instruction-fetching events for EXECUTE and the target are indicated.

2. When the operation is completed or partially completed, the event is indicated, regardless of whether any program exception, space-switch event, or monitor event is also recognized.

3. Successful branching, storage alteration, and general-register alteration are not indicated for an operation or, in case the instruction is interruptible, for a unit of operation that is suppressed or nullified.

4. When the execution of the instruction is terminated, general-register or storage alteration is indicated whenever the event has occurred, and a model may indicate the event if the event would have occurred had the execution of the instruction been completed, even if altering the contents of the result field is contingent on operand values. For purposes of this definition, the occurrence of those exceptions which permit termination (addressing, protection, and data) are considered to be termination, even if no result area is changed.

5. When LOAD PSW, SET SYSTEM MASK, STORE THEN OR SYSTEM MASK, or SUPERVISOR CALL causes a PER condition and at the same time introduces a new PSW with the type of PSW-format error that is recognized immediately after the PSW becomes active, the interruption code identifies both the PER condition and the specification exception. When LOAD PSW or SUPERVISOR CALL introduces a PSW-format error of the type that is recognized as part of the execution of the following instruction, the PSW is stored as the old PSW without the specification exception being recognized.

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

		PER Event			
Concurrent Condition	Type of Ending	Branch	Instr Fetch	Storage Alter.	GR Alter.
Specification					
Odd instruction address in the PSW	S	No	No	No	No
Instruction access					
First halfword	N or S	No	No	No	No
Second, third halfwords	N or S	No	U	No	No
Specification					
EXECUTE target address odd	S	No	U	No	No
EXECUTE target access	N or S	No	U	No	No
Other nullifying	N	No	Yes	No[1]	No[1]
Other suppressing	S	No	Yes	No[1]	No[1]
All terminating	T	No	Yes	Yes[2]	Yes[2]
All completing	C	Yes	Yes	Yes	Yes

Explanation:

[1] Although PER events of this type are not indicated for the current unit of operation of an interruptible instruction, PER events of this type that were recognized on completed units of operation of the interruptible instruction are indicated.

[2] This event may be indicated, depending on the model, if the event has not occurred but would have been indicated if execution had been completed.

C The operation or, in the case of the interruptible instructions, the unit of operation is completed.

N The operation or, in the case of the interruptible instructions, the unit of operation is nullified.

S The operation or, in the case of the interruptible instructions, the unit of operation is suppressed.

T The execution of the instruction is terminated.

Yes The PER event is indicated with the other program-interruption condition if the event has occurred; that is, the contents of the designated storage location or general register were altered, or an attempt was made to execute an instruction whose first byte is located in the designated storage area.

No The PER event is not indicated.

U It is unpredictable whether the PER event is indicated.

Indication of PER Events with Other Concurrent Conditions

Programming Notes

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

   Interruption of such an instruction may cause a PER event to be indicated more than once. It may be necessary, therefore, for a program to remove the redundant event indications from the PER data. The following rules govern the indication of the applicable events during execution of these instructions:

   a. The instruction-fetching event is indicated whenever the instruction is fetched for execution, regardless of whether it is the initial execution or a resumption.

b. The general-register-alteration event is indicated on the initial execution and on each resumption and does not depend on whether or not the register actually is changed.

c. The storage-alteration event is indicated only when data has been stored in the designated storage area by the portion of the operation starting with the last initiation and ending with the last byte transferred before the interruption. No special indication is provided on premature interruptions as to whether the event will occur again upon the resumption of the operation. When the designated storage area is a single byte location, a storage-alteration event can be recognized only once in the execution of MOVE LONG.

2. The following is an outline of the general action a program must take to delete multiple entries in the PER data for an interruptible instruction so that only one entry for each complete execution of the instruction is obtained:

   a. Check to see if the PER address is equal to the instruction address in the old PSW and if the last instruction executed was interruptible.

   b. If both conditions are met, delete instruction-fetching and register-alteration events.

   c. If both conditions are met and the event is storage alteration, delete the event if some part of the remaining destination operand is within the designated storage area.

DIRECT CONTROL

The direct-control facility consists of two facilities: (1) a read-write-direct facility, including the two instructions READ DIRECT and WRITE DIRECT and an associated 27-line interface, and (2) an external-signal facility with six signal-in lines. These facilities operate independent of the facilities that perform I/O operations.

READ-WRITE-DIRECT FACILITY

The READ DIRECT and WRITE DIRECT instructions use the 27-line interface to provide timing signals and to transfer a single byte of information, normally for controlling and synchronizing purposes, between CPUs or between a CPU and an external device. The 27 lines are:

Name	Number of Lines	Direction
Write out	1	Output
Read out	1	Output
Hold	1	Input
Signal out	8	Output
Direct out	8	Output
Direct in	8	Input

EXTERNAL-SIGNAL FACILITY

The external-signal facility consists of six signal-in lines and an external-signal mask, which is bit 26 of control register 0. Each of the six signal-in lines, when pulsed, sets up the condition for one of six distinct interruptions (see the section "External Signal" in Chapter 6, "Interruptions").

Note: Some models provide the external-signal facility without the read-write-direct facility.

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

TIMING

The timing facilities include four facilities for measuring time: the TOD clock, the clock comparator, the CPU timer, and the interval timer.

In a multiprocessing configuration, a single TOD clock may be shared by more than one CPU, or each CPU may have a separate TOD clock. However, each CPU has a separate clock comparator, CPU timer, and interval timer.

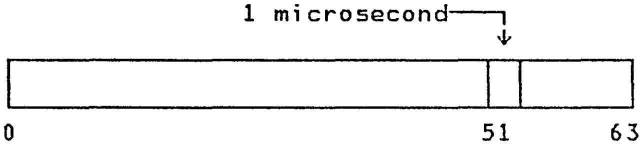
TIME-OF-DAY CLOCK

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

In an installation with more than one CPU, each CPU may have a separate TOD clock, or more than one CPU may share a clock, depending on the model. In all cases, each CPU has access to a single clock.

Format

The TOD clock is a binary counter with the format shown in the following illustration. The bit positions of the clock are numbered 0 to 63, corresponding to the bit positions of a 64-bit unsigned binary integer.



In the basic form, the TOD 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 advancing the clock is the same as if a one were added in bit position 51 every microsecond. The resolution of the TOD clock is such that the incrementing rate is comparable to the instruction-execution rate of the model.

A TOD clock is said to be in a particular multiprocessing configuration if at least one of the CPUs which shares that clock is in the configuration. Thus, it is possible for a single TOD clock to be in more than one configuration. Conversely, if all CPUs having access to a particular TOD clock have been removed from a particular configuration, then the TOD clock is no longer considered to be in that configuration.

When more than one TOD clock exists in the configuration, the stepping rates are synchronized such that all TOD clocks in the configuration are incremented at exactly the 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. The program is not alerted, and no interruption condition is generated as a result of the overflow.

The operation of the clock is not affected by any normal activity or event in the system. Incrementing of the clock does not depend on whether the wait-state bit of the PSW is one or whether the CPU is in the operating, load, stopped, or check-stop state. Its operation is not affected by CPU, initial-CPU, program, initial-program, or clear resets or by initial program loading. Operation of the clock is also not affected by the setting of the rate control or by an initial-microprogram-loading operation. Depending on the model and the configuration, a TOD clock may or may not be powered independent of a CPU that accesses it.

States

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

Not-Set State: When the power for the clock is turned on, the clock is set to zero, and the clock enters the not-set state. The clock is incremented when in the not-set state.

When the clock is in the not-set state, execution of STORE CLOCK causes condition code 1 to be set and the current value of the running clock to be stored.

Stopped State: The clock enters the stopped state when SET CLOCK is executed on a CPU accessing that clock and the clock is set. This occurs when SET CLOCK is executed without encountering any exceptions and any manual TOD-clock control in the configuration is set to 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, execution of STORE CLOCK on a CPU accessing that clock causes condition code 3 to be set and the value of the stopped clock to be stored.

Set State: The clock enters the set state only from the stopped state. The change of state is under control of the TOD-clock-sync-control bit, bit 2 of control register 0, in the CPU which most recently caused that clock to enter the stopped state. If the bit is zero or the TOD-clock-synchronization facility is not installed, the clock enters the set state at the completion of execution of SET CLOCK. If the bit is one, the clock remains in the stopped state until the bit is set to zero on that CPU, until another CPU executes a SET CLOCK instruction affecting the clock, or until any other clock in the configuration is incremented to a value of all zeros in bit positions 32-63. If any 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, the updating of bits 32-63 of the two clocks is in synchronism.

Incrementing of the clock begins with the first stepping pulse after the clock enters the set state.

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

Error State: The clock enters the error state when a malfunction is detected that is likely to have affected the validity of the clock value. A timing-facility-damage machine-check-interruption condition is generated on each CPU which has access to that clock whenever it enters the error state.

When STORE CLOCK is executed and the clock accessed is in the error state, condition code 2 is set, and the value stored is unpredictable.

Not-Operational State: The clock is in the not-operational state when its power is off or when it is disabled for maintenance. It depends on the model if the clock can be placed in this state. Whenever the clock enters the not-operational state, a timing-facility-damage machine-check-interruption condition is generated on each CPU that has access to that clock.

When the clock is in the not-operational state, execution of STORE CLOCK causes condition code 3 to be set, and zero is stored.

Changes in Clock State

When the TOD clock accessed by a CPU changes value because of the execution of SET CLOCK or changes state, interruption conditions pending for the clock comparator, CPU timer, interval timer, and TOD-clock-sync check may or may not be recognized for up to 1.048576 seconds (2^{20} microseconds) after the change.

Setting and Inspecting the Clock

The clock can be set to a specific value by execution of SET CLOCK if the manual TOD-clock control of any CPU in the configuration is in the enable-set position. Setting the clock replaces the values in all bit positions from bit position 0 through the rightmost position that is incremented when the clock is running. However, on some models, the rightmost bits starting at or to the right of bit 52 of the specified value are ignored, and zeros are placed in the corresponding positions of the clock.

The TOD clock can be inspected by executing STORE CLOCK, which causes a 64-bit value to be stored. Two executions of STORE CLOCK, possibly on different CPUs in the same configuration, always store different values if the clock is running or, if separate clocks are accessed, both clocks are running and are synchronized.

The values stored for a running clock always correctly imply the sequence of execution of STORE CLOCK on one or more CPUs for all cases where the sequence can be established by means of the program. Zeros are stored in positions to the right of the bit position that is incremented. In a configuration with more than one CPU, however, when the value of a running clock is stored, nonzero values may be stored in positions to the right of the rightmost position that is incremented. This ensures that a unique value is stored.

In a configuration where more than one CPU accesses the same clock, SET CLOCK is interlocked such that the entire contents appear to be updated concurrently; that is, if SET CLOCK instructions are executed simultaneously by two CPUs, the final result is either one or the other value. If SET CLOCK is executed on one CPU and STORE CLOCK on the other, the result obtained by STORE CLOCK is either the entire old value or the entire new value. When SET CLOCK is executed by one CPU, a STORE CLOCK executed on another CPU may find the clock in the stopped state even when the TOD-clock-sync-control bit is zero in each CPU. The TOD-clock-sync-control bit is bit 2 of control register 0. Since the clock enters the set state before incrementing, the first STORE CLOCK executed after the clock enters the set state may still find the original value introduced by SET CLOCK.

Programming Notes

1. Bit position 31 of the clock is incremented every 1.048576 seconds; for some applications, reference to the leftmost 32 bits of the clock may provide sufficient resolution.

2. Communication between systems is facilitated by establishing a standard time origin, or standard epoch, which is the calendar date and time to which a clock value of zero corresponds. January 1, 1900, 0 a.m. Greenwich Mean Time (GMT) is recommended as the standard epoch for the clock.

3. A program using the clock value as a time-of-day and calendar indication must be consistent with the programming support under which the program is to be executed. If the programming support uses the stand-

ard epoch, bit 0 of the clock remains one through the years 1972-2041. (Bit 0 turned on at 11:56:53.685248 (GMT) May 11, 1971.) Ordinarily, testing bit 0 for a one is sufficient to determine if the clock value is in the standard epoch.

4. Because of the limited accuracy of manually setting the clock value, the rightmost bit positions of the clock, expressing fractions of a second, are normally not valid as indications of the time of day. However, they permit elapsed-time measurements of high resolution.

5. The following chart shows the time interval between instants at which various bit positions of the TOD clock are stepped. This time value may also be considered as the weighted time value that the bit, when one, represents.

TOD-Clock Bit	Stepping Interval			
	Days	Hours	Min.	Seconds
51				0.000 001
47				0.000 016
43				0.000 256
39				0.004 096
35				0.065 536
31				1.048 576
27				16.777 216
23			4	28.435 456
19		1	11	34.967 296
15		19	5	19.476 736
11	12	17	25	11.627 776
7	203	14	43	6.044 416
3	3257	19	29	36.710 656

6. The following chart shows the clock setting at the start of various years. The clock settings, expressed in hexadecimal notation, correspond to 0 a.m. Greenwich Mean Time on January 1 of each year.

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

7. The stepping value of TOD-clock bit position 63, if implemented, is 2^{-12} microseconds, or approximately 244 picoseconds. This value is called a clock unit.

The following chart shows various time intervals in clock units expressed in hexadecimal notation.

Interval	Clock Units (Hex)
1 microsecond	1000
1 millisecond	3E 8000
1 second	F424 0000
1 minute	39 3870 0000
1 hour	D69 3A40 0000
1 day	1 41DD 7600 0000
365 days	1CA E8C1 3E00 0000
366 days	1CC 2A9E B400 0000
1,461 days*	72C E4E2 6E00 0000

* Number of days in four years, including a leap year. Note that the year 1900 was not a leap year. Thus, the four-year span starting in 1900 has only 1460 days.

8. In a multiprocessing configuration, after the TOD clock is set and begins running, the program should delay activity for 2^{20} microseconds (1.048576 seconds) to ensure that the CPU-timer, clock-comparator, interval timer, and TOD-clock-sync-check interruption conditions are recognized by the CPU.

TOD-CLOCK SYNCHRONIZATION

In an installation with more than one CPU, each CPU may have a separate TOD clock, or more than one CPU may share a TOD clock, depending on the model. In all cases, each CPU has access to a single clock.

The TOD-clock-synchronization facility, in conjunction with a clock-synchronization program, makes it possible to provide the effect of all CPUs in a multiprocessing configuration sharing a single TOD clock. The result is such that, to all programs storing the TOD-clock value, it appears that all CPUs in the configuration read the same TOD clock. The TOD-clock-synchronization facility provides these functions in such a way that even though the number of CPUs sharing a TOD clock is model-dependent, a single model-independent clock-synchronization routine can be written. The following functions are provided:

- Synchronizing the stepping rates for all TOD clocks in the configuration. Thus, if all clocks are set to the same value, they stay in synchronism.

- Comparing the rightmost 32 bits of each clock in the configuration. An unequal condition is signaled by an external interruption with the interruption code 1003 hex, indicating the TOD-clock-sync-check condition.

- Setting a TOD clock to the stopped state.

- Causing a stopped clock, with the TOD-clock-sync-control bit set to one, to start incrementing 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.

Programming Notes

1. TOD-clock synchronization provides for checking and synchronizing only the rightmost bits of the TOD clock. The program must check for synchronization of the leftmost bits and must communicate the leftmost-bit values from one CPU to another in order to correctly set the TOD-clock contents.

2. The TOD-clock-sync-check external interruption can be used to determine the number of TOD clocks in the configuration.

CLOCK COMPARATOR

The clock comparator provides a means of causing an interruption when the TOD-clock value exceeds a value specified by the program.

In a configuration with more than one CPU, each CPU has a separate clock comparator.

The clock comparator has the same format as the TOD clock. In the basic form, the clock comparator consists of bits 0-47, which are compared with the corresponding bits of the TOD clock. In some models, higher resolution is obtained by providing more than 48 bits. The bits in positions provided in the clock comparator are compared with the corresponding bits of the clock. When the resolution of the clock is less than that of the clock comparator, the contents of the clock comparator are compared with the clock value as this value would be stored by executing STORE CLOCK.

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:

1. The TOD clock is running and the value of the clock comparator is less than the value in the compared portion of the clock, both values being considered unsigned binary integers. Comparison follows the rules of unsigned binary arithmetic.

2. The TOD clock is in the error state or the not-operational state.

A request for a clock-comparator interruption does not remain pending when the value of the clock comparator is made equal to or greater than that of the TOD clock or when the value of the TOD clock is made less than the clock-comparator value. The latter may occur as a result of the TOD clock either being set or wrapping to zero.

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

The contents of the clock comparator are initialized to zero by initial CPU reset.

Programming Notes

1. An interruption request for the clock comparator persists as long as the clock-comparator value is less than that of the TOD clock or as long as the TOD clock is in the error state or the not-operational state. Therefore, one of the following actions must be taken after an external interruption for the clock comparator has occurred and before the CPU is again enabled for external interruptions: the value of the clock comparator has to be replaced, the TOD clock has to be set, the TOD clock has to wrap to zero, or the clock-comparator-subclass mask has to be set to zero. Otherwise, loops of external interruptions are formed.

2. The instruction STORE CLOCK may store a value which is greater than that in the clock comparator, even though the CPU is enabled for the clock-comparator interruption. This is because the TOD clock may be incremented one or more times between when instruction execution is begun and when the clock value is accessed. In this situation,

the interruption occurs when the execution of STORE CLOCK is completed.

CPU TIMER

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

In a configuration with more than one CPU, each CPU has a separate CPU timer.

The CPU timer is a binary counter with a format which is the same as that of the TOD 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 decrementing the CPU timer is the same as if a one were subtracted in bit position 51 every microsecond. The resolution of the CPU timer is such that the stepping rate is comparable to the instruction-execution rate of the model.

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

When both the CPU timer and the TOD clock are running, the stepping rates are synchronized such that both are stepped at the same rate. Normally, decrementing the CPU timer is not affected by concurrent I/O activity. However, 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 would have been recorded had the CPU timer continued to step.

The CPU timer is decremented when the CPU is in the operating state or the load state. When the manual rate control is set to instruction step, the CPU timer is decremented only during the time in which the CPU is actually performing a unit of operation. However, depending on the model, the CPU timer may or may not be decremented when the TOD clock is in the error, stopped, or not-operational state.

Depending on the model, the CPU timer may or may not be decremented when the CPU is in the check-stop state.

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

The CPU timer is set to zero by initial CPU reset.

Programming Notes

1. The CPU timer in association with a program may be used both to measure CPU-execution time and to signal the end of a time interval on the CPU.

2. The time measured for the execution of a sequence of instructions may depend on the effects of such things as I/O interference, the availability of pages, and instruction retry. Hence, repeated measurements of the same sequence on the same installation may differ.

3. The fact that a CPU-timer interruption does not remain pending when the CPU timer is set to a positive value eliminates the problem of an undesired interruption. This would occur if, between the time when the old value is stored and a new value is set, the CPU is disabled for CPU-timer interruptions and the CPU timer value goes from positive to negative.

4. The fact that CPU-timer interruptions are requested whenever the CPU timer is negative (rather than just when the CPU timer goes from positive to negative) eliminates the requirement for testing a value to ensure that it is positive before setting the CPU timer to that value.

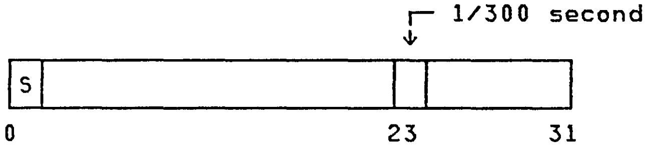
   As an example, assume that a program being timed by the CPU timer is interrupted for a cause other than the CPU timer, external interruptions are disallowed by the new PSW, and the CPU-timer value is then saved by STORE CPU TIMER. This value could be negative if the CPU timer went from positive to negative since the interruption. Subsequently, when the program being timed is to continue, the CPU timer may be set to the saved value by SET CPU TIMER. A CPU-timer interruption occurs immediately after external interruptions are again enabled if the saved value was negative.

   The persistence of the CPU-timer-interruption request means, however, that after an external interruption for the CPU timer has occurred, the value of the CPU timer has to be replaced, the value in the CPU timer has to wrap to a positive value, or the CPU-timer-subclass mask has to be set to zero before the CPU is again enabled for external interruptions. Otherwise, loops of external interruptions are formed.

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

INTERVAL TIMER

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



The interval timer is treated as a 32-bit signed binary integer. In the basic form, the contents of the interval timer are decremented by one in both bit positions 21 and 22 every 1/50 of a second, or the interval-timer contents are decremented by one in both bit positions 21 and 23 every 1/60 of a second. Higher resolution of timing may be obtained in some models by counting with higher frequency in other bit positions. In each case, the frequency is adjusted so that bits to the left of bit position 23 change as if bit position 23 were being decremented by one every 1/300 of a second. The cycle of the interval timer is approximately 15.5 hours.

In a configuration with more than one CPU, each CPU has an interval timer.

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

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

The interval timer is not necessarily synchronized with the TOD clock.

The interval-timer contents are updated at the appropriate frequency whenever other machine activity permits. The updating occurs only between instruction executions, except that the interval timer may be updated between units of operation of an interruptible instruction, such as MOVE LONG. An updated interval-timer value is normally available at the end of each instruction execution. When the execution of an instruction, I/O data transmission, or other machine activity causes updating to be delayed by more than one period, the contents of the interval timer may be decremented by more than one unit in a single updating cycle. Interval-timer updating may be omitted when such delay is extreme. The program is not alerted when omission of updating causes the real-time count to be lost.

When the contents of the interval timer are fetched by a channel or another CPU, or when they are used as the source of an instruction, the result is unpredictable. Similarly, storing by a channel or another CPU into the interval timer causes the contents of the interval timer to be unpredictable. This unpredictability is true even for the case of COMPARE AND SWAP or COMPARE DOUBLE AND SWAP when executed by another CPU.

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

Depending on the model, the interval timer may or may not be decremented when the TOD clock is in the error, stopped, or not-operational state.

When the TOD clock accessed by a CPU is set or changes state, interruption conditions pending for the interval timer may or may not be recognized for up to 1.048576 seconds after the change.

Programming Notes

1. The value of the interval timer is accessible by fetching the word at real location 80 as an operand, provided the location is not protected against fetching. It may be changed at any time by storing a

word at real location 80. When real location 80 is protected, any attempt by the program to change the value of the interval timer causes a program interruption for protection exception.

2. The value of the interval timer may be changed without losing the real-time count by storing the new value at real locations 84-87 and then copying the contents of real locations 80-87 to real locations 76-83 by means of the MOVE (MVC) instruction. Thus, in a single operation, the new interval-timer value is placed at real locations 80-83, and the old value is made available at real locations 76-79.

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

   Logical locations 84-87 are used as the trace-table designation by DAS tracing. If the above means for updating the interval timer by using MOVE are used in a system which also uses DAS tracing, and if logical location 84 maps to real location 84, then the program must restore the contents of the word at real location 84 after updating the interval timer.

3. When the value of the interval timer is to be recorded on an I/O device, the program should first store the interval-timer value in a temporary storage location to which the I/O operation subsequently refers. When a channel program fetches from locations 80-83, the value obtained is unpredictable.

EXTERNALLY INITIATED FUNCTIONS

RESETS

Seven reset functions are provided:

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

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

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

Subsystem reset provides a means for clearing floating interruption conditions and for initializing channel-set connections as well as for invoking I/O-system reset.

Program reset and initial program reset cause CPU reset and initial CPU reset, respectively, to be performed and cause I/O-system reset to be performed (see the section "I/O-System Reset" in Chapter 13, "Input/Output Operations").

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

The power-on-reset 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.

CPU reset, initial CPU reset, subsystem reset, and clear reset may be initiated manually by using the operator facilities (see Chapter 12, "Operator Facilities"). Initial CPU reset is part of the initial-program-loading function. The figure "Manual Initiation of Resets" summarizes how these four resets are manually initiated. Power-on reset is performed as part of turning power on. The reset actions are tabulated in the figure "Summary of Reset Actions." For information concerning what resets can be performed by the SIGNAL PROCESSOR instruction, see the section "Signal-Processor Orders" in this chapter.

Key Activated	Function Performed on[1]		
	CPU on Which Key Was Activated	Other CPUs in Config	Remainder of Configuration
System-reset-normal key			
• without store-status facility	Initial CPU reset	*	Subsystem reset
• with store-status facility	CPU reset	CPU reset	Subsystem reset
System-reset-clear key	Clear reset[2]	Clear reset[2]	Clear reset[3]
Load-normal key	Initial CPU reset, followed by IPL	CPU reset	Subsystem reset
Load-clear key	Clear reset[2], followed by IPL	Clear reset[2]	Clear reset[3]

Explanation:

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

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

[2] Only the CPU elements of this reset apply.

[3] Only the non-CPU elements of this reset apply.

Manual Initiation of Resets

	Reset Function						
Area Affected	Subsystem Reset	CPU Reset	Program Reset	Initial CPU Reset	Initial Program Reset	Clear Reset	Power-On Reset
CPU	U	S	S	S[1]	S	S[1]	S
PSW[2]	U	U/V	U/V	C*[1]	C*	C*[1]	C*
Prefix	U	U/V	U/V	C	C	C	C
CPU timer	U	U/V	U/V	C	C	C	C
Clock comparator	U	U/V	U/V	C	C	C	C
Control registers	U	U/V	U/V	I	I	I	I
General registers	U	U/V	U/V	U/V	U/V	C/V	C/X
Floating-point registers	U	U/V	U/V	U/V	U/V	C/V	C/X
Vector-facility registers	U	U/V	U/V	U/V	U/V	C	C
Storage keys	U	U	U	U	U	C	C/X[3]
Volatile main storage	U	U	U	U	U	C	C/X[3]
Nonvolatile main storage	U	U	U	U	U	C	U
Expanded storage	U[4]	U[4]	U[4]	U[4]	U[4]	U[4]	C[3]
TOD clock	U[5]	U[5]	U[5]	U[5]	U[5]	U[5]	T[3]
Channel-set connection	I	U	U	U	U	I	I[6]
Floating interruption conditions	C	U	U	U	U	C	C[3]
Channels in the configuration	RA	U	RC	U	RC	RA	RA[3]

Explanation:

* Clearing the contents of the PSW to zero places the CPU in the BC mode.

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

[2] For a BC-mode PSW, the ILC and interruption-code fields are unpredictable in the current PSW.

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

[4] Access to change expanded storage at the time a reset function is performed may cause the contents of the 4K-byte block in expanded storage to be unpredictable. Access to examine expanded storage does not affect the contents of the expanded storage.

[5] 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.

[6] When these units are separately powered, the action is model-dependent.

C The condition or contents are cleared. If the area affected is a field, the contents are set to zeros with valid checking-block code.

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

I The state or contents are initialized. If the area affected is a field, the contents are set to the initial value with valid checking-block code.

Summary of Reset Actions (Part 1 of 2)

Explanation (Continued):

RA I/O-system reset is performed in all the channels in the configuration 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 attached to the channels being reset.

RC I/O-system reset is performed in those channels connected to the CPU performing the program reset or initial-program reset. As part of this reset, system reset is signaled to the I/O control units and devices attached to the channels being reset.

S The CPU is reset; current operations, if any, are terminated; the TLB is cleared of entries; interruption conditions in the CPU are cleared; and the CPU is placed in the stopped state. The effect of performing the start function is unpredictable when the stopped state has been entered by means of a reset.

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

U The state, condition, or contents of the field remain unchanged. However, the result is unpredictable if an operation is in progress that changes the state, condition, or contents of the field at the time of reset.

U/V The contents remain unchanged, provided the field is not being changed at the time the reset function is performed. However, on some models, the checking-block code of the contents may be made valid. The result is unpredictable if an operation is in progress that changes the contents of the field at the time of reset.

Summary of Reset Actions (Part 2 of 2)

CPU Reset

CPU reset causes the following actions:

1. The execution of the current instruction or other processing sequence, such as an interruption, is terminated, and all program-interruption and supervisor-call-interruption conditions are cleared.

2. Any pending external-interruption conditions which are local to the CPU are cleared. Floating external-interruption conditions are not cleared.

3. Any pending machine-check-interruption conditions and error indications which are local to the CPU and any check-stop states are cleared. Floating machine-check-interruption conditions are not cleared. Any machine-check condition which is reported to all CPUs in the configuration and which has been made pending to a CPU is said to be local to the CPU.

4. All copies of prefetched instructions or operands are cleared. Additionally, any results to be stored because of the execution of instructions in the current checkpoint interval are cleared.

5. The translation-lookaside buffer is cleared of entries.

6. The CPU is placed in the stopped state after actions 1-5 have been completed. When the IPL sequence follows the reset function on that CPU, the CPU enters the load state at the completion of the reset function and does not necessarily enter the stopped state during the execution of the reset operation.

Registers, storage contents, and the state of conditions external to the CPU remain unchanged by CPU reset. However, the subsequent contents of the register, location, or state are unpredictable if an operation is in progress that changes the contents at the time of the reset.

When the reset function in the CPU is initiated at the time the CPU is executing an I/O instruction or is performing an I/O interruption, the current operation between the CPU and the channel may or may not be completed, and the resultant state of the associated channel may be unpredictable.

Programming Note

Most operations which would change a state, a condition, or the contents of a field cannot occur when the CPU is in the stopped state. However, some signal-processor functions and some operator functions may change these fields. To eliminate the possibility of losing a field when CPU reset is issued, the CPU should be stopped, and no operator functions should be in progress.

Initial CPU Reset

Initial CPU reset combines the CPU reset functions with the following clearing and initializing functions:

1. The contents of the current PSW, prefix, CPU timer, and clock comparator are set to zero. When the IPL sequence follows the reset function on that CPU, the contents of the PSW are not necessarily set to zero.

2. All assigned control-register positions are set to their initial value.

These clearing and initializing functions include validation.

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

Subsystem Reset

Subsystem reset operates only on those elements in the configuration which are not CPUs. It performs the following actions:

1. I/O-system reset is performed in each channel in the configuration.

2. All floating interruption conditions in the configuration are cleared.

3. Channel-set connections are initialized to connect each channel set to its home CPU if one exists, is operational, and is in the configuration, or else to make the channel set disconnected.

As part of I/O-system reset, pending I/O-interruption conditions are cleared, and system reset is signaled to all control units and devices attached to the channel (see the section "I/O-System Reset" in Chapter 13, "Input/Output Operations"). The effect of system reset on I/O control units and devices and the resultant control-unit and device state are described in the appropriate System Library publication for the control unit or device. A system reset, in general, resets only those functions in a shared control unit or device that are associated with the particular channel signaling the reset.

Program Reset

For program reset, CPU reset is performed, and I/O-system reset is performed in each channel connected to this CPU.

Initial Program Reset

Initial program reset combines the program-reset functions with the clearing and initializing functions of initial CPU reset.

Clear Reset

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

1. In most models, the contents of the general and floating-point registers of those CPUs which are in the configuration are set to zero, but in some models the contents may be left unchanged except that the checking-block code is made valid.

2. The registers (vector-status register, vector-mask register, vector-activity count, and all vector registers) of those vector facilities, if any, which are in the configuration are cleared to zero with valid checking-block code.

3. The contents of the main storage in the configuration and the associated storage keys are set to zero with valid checking-block code.

4. A subsystem reset is performed.

Validation is included in setting registers and in clearing storage and storage keys.

Programming Notes

1. For the CPU-reset or program-reset operation not to affect the contents of fields that are to be left unchanged, the CPU must not be executing instructions and must be

disabled for all interruptions at the time of the reset. Except for the operation of the interval timer and CPU timer and for the possibility of a machine-check interruption occurring, all CPU activity can be stopped by placing the CPU in the wait state and by disabling it for I/O and external interruptions. To avoid the possibility of causing a reset at the time that the interval timer or CPU timer is being updated or a machine-check interruption occurs, the CPU must be in the stopped state.

2. CPU reset, initial CPU reset, subsystem reset, program reset, initial program reset, and clear reset do not affect the value and state of the TOD clock.

3. The conditions under which the CPU enters the check-stop state are model-dependent and include malfunctions that preclude the completion of the current operation. Hence, if CPU reset, initial CPU reset, program reset, or initial program reset is executed while the CPU is in the check-stop state, the contents of the PSW, registers, and storage locations, including the storage keys and the storage location accessed at the time of the error, may have unpredictable values, and, in some cases, the contents may still be in error after the check-stop state is cleared by these resets. In this situation, a clear reset is required to clear the error.

4. Clear reset causes all bit positions of the interval timer to be cleared to zeros.

Power-On Reset

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

The power-on sequences for the TOD clock, vector facility, main storage, expanded 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, vector facility, main storage, expanded storage, and channels. See also Chapter 13, "Input/Output Operations," and the appropriate System Library 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 may or may not cause I/O-system reset to be performed in the channels connected to the CPU. The contents of general registers and floating-point registers normally are cleared to zeros, but in some models may be left unpredictable, with valid checking-block code.

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

Vector-Facility Power-On Reset: The power-on reset causes the registers of the vector facility (vector-status register, vector-mask register, vector-activity count, and all vector registers) to be cleared to zeros with valid checking-block code.

Main-Storage Power-On Reset: For volatile main storage (one that does not preserve its contents when power is off) and for storage keys, power-on reset causes 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.

Expanded-Storage Power-On Reset: The contents of the expanded storage are cleared to zeros with valid checking-block code.

Channel Power-On Reset: The channel power-on reset causes I/O-system reset to be performed. (See the section "I/O-System Reset" in Chapter 13, "Input/Output Operations.")

INITIAL PROGRAM LOADING

Initial program loading (IPL) provides a manual means for causing a program to be read from a designated device and for initiating execution of that program.

Some models may provide additional controls and indications relating to IPL; this additional information is specified in the System Library publication for the model.

IPL is initiated manually by setting the load-unit-address controls to designate an input device and by subsequently activating the load-clear or load-normal key for a particular CPU. In the description which follows, the term "this CPU" refers to the CPU in the configuration for which the load-clear or load-normal key was activated.

Activating the load-clear key causes a clear reset to be performed on the configuration.

Activating the load-normal key causes an initial CPU reset to be performed on this CPU, CPU reset to be propagated to all other CPUs in the configuration, and a subsystem reset to be performed on the remainder of the configuration.

In the loading part of the operation, after the resets have been performed, this CPU then enters the load state. This CPU does not necessarily enter the stopped state during the execution of the reset operations. The load indicator is on while the CPU is in the load state.

Subsequently, a channel program read operation is initiated from the channel and I/O device designated by the load-unit-address controls.

The read operation is performed as if a START I/O instruction were executed that specified the channel, subchannel, and I/O device designated by the load-unit-address controls. The operation uses an implied channel-address word (CAW) containing a subchannel key of zero, a suspend-control bit of zero, and a channel-command-word (CCW) address of 0, but the CAW at real location 72 is not accessed. The load-unit-address controls provide the 16-bit I/O address, of which the leftmost eight bits are the channel address and the rightmost eight bits the device address; any leftmost bits of the channel address that are omitted because they are not needed to select a channel are implied to be zeros.

Although the absolute location of the first CCW to be executed is specified by the CCW address as 0, the first CCW actually executed is an implied CCW, containing, in effect, a read command with the modifier bits set to zeros, a data address of 0, a byte count of 24, the chain-command and SLI flags set to ones, and the chain-data, skip, indirect-data-address, suspend, and PCI flags set to zeros. The CCW fetched, as a result of command chaining, from absolute location 8 or 16, as well as any subsequent CCW in the IPL sequence, is interpreted the same as a CCW in any I/O operation, except that any PCI flags that are specified in CCWs used in the IPL channel program are ignored.

When the I/O device provides channel-end status for the last operation of the IPL channel program and no exceptional conditions are detected in the operation, a new PSW is loaded from absolute storage locations 0-7. When this PSW specifies the EC mode, the I/O address that was used for the IPL operation is stored at absolute locations 186-187, and zeros are stored at absolute location 185; when the BC mode is specified, the I/O address is stored at absolute locations 2-3. If the PSW loading is successful and if no machine malfunctions are detected, this CPU leaves the load state and the load indicator is turned off. If the rate control is set to the process position, the CPU enters the operating state and the CPU operation proceeds under control of the new PSW. If the rate control is set to the instruction-step position, the CPU enters the stopped state, with the manual indicator on, after the new PSW is loaded.

When channel-end status for the last CCW of the IPL channel program 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 successfully, the CPU remains in the load state, and the load indicator remains on. This occurs when the device designated by the load-unit-address controls is not operational, when the device or channel signals any condition other than channel end, device end, or status modifier during or at the completion of the last CCW of the IPL channel program, or when the PSW loaded from absolute location 0 has a PSW-format error of the type that is recognized early. The address of the I/O device used in the IPL operation is not stored. The contents of absolute storage locations 0-7 are unpredictable. The contents of other storage locations remain unchanged, except possibly for those locations due to be changed by the read operations.

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

Programming Notes

1. The information read and placed at absolute locations 8-15 and 16-23 may be used as CCWs for reading additional information during the IPL I/O operation: the CCW at absolute location 8 may specify reading additional CCWs elsewhere in storage, and the CCW at absolute location 16 may specify the transfer-in-channel command, causing transfer to these CCWs.

2. The status-modifier bit, in conjunction with the device-end bit, has its normal effect during the IPL I/O operation, causing the channel to fetch and chain to the CCW whose address is 16 higher than that of the current CCW. This applies also to the initial chaining that occurs after completion of the read operation specified by the implicit CCW.

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

4. When the PSW in absolute location 0 has bit 14 set to one, the CPU is placed in the wait state after the IPL operation is completed; at that point, the load and manual indicators are off, and the wait indicator is on.

5. Activating the load-normal key 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 may be helpful in debugging and recovery. When the load-clear key is activated, the IPL program starts with a cleared machine in a known state, except that information on external storage remains unchanged.

STORE STATUS

The store-status facility includes:

1. A change to the operation of the system-reset-normal key. With the store-status facility installed, activating the system-reset-normal key causes a CPU-reset operation and a subsystem-reset operation to be performed; without this facility, an initial-CPU-reset operation and subsystem-reset operation are performed.

2. An operator-initiated store-status function.

The store-status operation places the contents of the CPU registers, except for the TOD clock, in assigned storage locations.

The figure "Assigned Storage Locations for Store Status" lists the fields that are stored, their length, and their location in main storage.

Field	Length in Bytes	Absolute Address
CPU timer*	8	216
Clock comparator*	8	224
Current PSW#	8	256
Prefix*	4	264
Model-dependent feat.*	4	268
Fl-pt registers 0-6*	32	352
General registers 0-15	64	384
Control registers 0-15	64	448

Explanation:

* If the facility is not installed, the contents of the field in storage remain unchanged.

# In the BC mode, the ILC is unpredictable, and the interruption code is stored as zeros.

Assigned Storage Locations for Store Status

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 uninstalled or unassigned control-register positions is unpredictable. If the CPU timer, clock comparator, prefix register or floating-point facility is not installed, the contents of the corresponding locations in storage remain unchanged.

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

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

The store-status operation can be initiated manually by use of the store-status key (see Chapter 12, "Operator Facilities"). The store-status operation can also be initiated at the addressed CPU by executing SIGNAL PROCESSOR, specifying the stop-and-store-status order.

MULTIPROCESSING

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

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

TOD-clock synchronization is described earlier in this chapter. Prefixing is described in Chapter 3, "Storage." Shared main storage, CPU-address identification, and CPU signaling and response are described in the sections which follow.

Associated with these facilities are four extensions to the external interruption (external call, emergency signal, TOD-clock-sync check, and malfunction alert), which are described in Chapter 6, "Interruptions"; control-register positions for the TOD-clock-sync-control bit and for the masks for the external-interruption conditions, which are listed in the section "Control Registers" in this chapter; and the instructions SET PREFIX, SIGNAL PROCESSOR, STORE CPU ADDRESS, and STORE PREFIX, which are described in Chapter 10, "Control Instructions."

Channels in a multiprocessing configuration are connected to a particular CPU. Only that CPU which is connected to a channel can initiate I/O operations at that channel, and all interruption conditions are directed to that CPU. When channel-set switching is installed, the channel-CPU connection can be changed by means of the program.

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 2K-byte block containing that location and to the associated storage key. When the storage-key 4K-byte-block facility is installed, all CPUs having access to a common main-storage location have access to the entire 4K-byte block containing that location and to the associated single key in that block. All CPUs and all channels in the configuration refer to a shared main-storage location using the same absolute address.

CPU-ADDRESS IDENTIFICATION

Each CPU in a multiprocessing configuration has a number assigned, called its CPU address. A CPU address uniquely identifies one CPU within a configuration. The CPU is designated by specifying this address in the CPU-address field of SIGNAL PROCESSOR. The CPU signaling a malfunction alert, emergency signal, or external call is identified by storing this address in the CPU-address field with the interruption. The CPU address is assigned during system installation and is not changed as a result of reconfiguration changes. The program can determine the address of the CPU by using STORE CPU ADDRESS.

CPU SIGNALING AND RESPONSE

The CPU-signaling-and-response facility consists of SIGNAL PROCESSOR and a mechanism to interpret and act on several order codes. The facility provides for communications among CPUs, including transmitting, receiving, and decoding a set of assigned order codes; initiating the specified operation; and responding to the signaling CPU. If a CPU has the CPU-signaling-and-response facility installed, it can address SIGNAL PROCESSOR to itself. SIGNAL PROCESSOR is described in Chapter 10, "Control Instructions."

SIGNAL-PROCESSOR ORDERS

The signal-processor orders are specified in bit positions 24-31 of the second-operand address of SIGNAL PROCESSOR and are encoded as shown in the figure "Encoding of Orders."

Code	Order
00	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	Unassigned

Encoding of Orders

The orders are defined as follows:

Sense: The addressed CPU presents its status to the issuing CPU (see the section "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 of the SIGNAL PROCESSOR instruction, and condition code 1 is set; if all status bits are zeros, condition code 0 is set.

External Call: An external-call external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of SIGNAL PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. Only one external-call condition can be kept pending in a CPU at a time. The order is effective only when the addressed CPU is in the stopped or the operating state.

Emergency Signal: An emergency-signal external-interruption condition is generated at the addressed CPU. The interruption condition becomes pending during the execution of SIGNAL PROCESSOR. The associated interruption occurs when the CPU is enabled for that condition and does not necessarily occur during the execution of SIGNAL PROCESSOR. The address of the CPU sending the signal is provided with the interruption code when the interruption occurs. At any one time the receiving CPU can keep pending one emergency-signal condition for each CPU in the configuration, including the receiving CPU itself. The order is effective only when the addressed CPU is in the stopped or the operating state.

Start: The addressed CPU performs the start function (see the section "Stopped, Operating, Load, and Check-Stop States" in this chapter). The CPU does not necessarily enter the operating state during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped state. The effect of performing the start function is unpredictable when the stopped state has been entered by reset.

Stop: The addressed CPU performs the stop function (see the section "Stopped, Operating, Load, and Check-Stop States" in this chapter). The CPU does not necessarily enter the stopped state during the execution of SIGNAL PROCESSOR. The order is effective only when the CPU is in the operating state.

Restart: The addressed CPU performs the restart operation (see the section "Restart Interruption" in Chapter 6, "Interruptions"). The CPU does not necessarily perform the operation during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped or the operating state.

Initial Program Reset: The addressed CPU performs initial program reset (see the section "Resets" in this chapter). The execution of the reset does not affect other CPUs. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

Program Reset: The addressed CPU performs program reset (see the section "Resets" in this chapter). The execution of the reset does not affect other CPUs. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

Stop and Store Status: The addressed CPU performs the stop function, followed by the store-status function (see the section "Store Status" in this chapter). The CPU does not necessarily complete the operation, or even enter the stopped state, during the execution of SIGNAL PROCESSOR. The order is effective only when the addressed CPU is in the stopped or the operating state.

Initial Microprogram Load (IML): The addressed CPU performs initial program reset and then initiates the IML function. The IML function is the same as that which is performed as part of manual initial microprogram loading. If the IML 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 SIGNAL PROCESSOR.

Initial CPU Reset: The addressed CPU performs initial CPU reset (see the section "Resets" in this chapter). The execution of the reset does not affect other CPUs and does not cause I/O to be reset. If the initial-CPU-reset order is not provided on the addressed CPU, the order is treated as unassigned and invalid. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

CPU Reset: The addressed CPU performs CPU reset (see the section "Resets" in this chapter). The execution of the reset does not affect other CPUs and does not cause I/O to be reset. If the CPU-reset order is not provided on the addressed CPU, the order is treated as unassigned and invalid. The reset operation is not necessarily completed during the execution of SIGNAL PROCESSOR.

Programming Note

For a discussion on the relative performance of the SIGNAL PROCESSOR orders, see the programming note following the instruction SIGNAL PROCESSOR in Chapter 10, "Control Instructions."

CONDITIONS DETERMINING RESPONSE

Conditions Precluding Interpretation of the Order Code

The following situations preclude the initiation of the order. The sequence in which the situations are listed is the order of priority for indicating concurrently existing situations:

1. The access path to the addressed CPU is busy because a concurrently executed SIGNAL PROCESSOR is using the CPU-signaling-and-response facility. The CPU which is concurrently executing the instruction can be any CPU in the configuration other than this CPU, and the CPU address can be any address, including that of this CPU or an invalid address. The order is rejected. Condition code 2 is set.

2. The addressed CPU is not operational; that is, it is not provided in the installation, it is not in the configuration, it is in any of certain customer-engineer test modes, or its power is off. The order is rejected. Condition code 3 is set. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

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 function requested by the order has not yet been completed.

   b. A manual start, stop, restart, or store-status function has been initiated at the addressed CPU, and the function has not yet been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

   If the currently specified order is sense, external call, emergency signal, start, stop, restart, or stop and store status, then the order is rejected, and condition code 2 is set. If the currently specified order is an IML, one of the reset orders, or an unassigned or not-implemented order, the order code is interpreted as described in the section "Status Bits" in this chapter.

4. One of the following conditions exists at the addressed CPU:

   a. A previously issued initial-program-reset, program-reset, IML, initial-CPU-reset, or CPU-reset order has been accepted by the addressed CPU, and execution of the function requested by the order has not yet been completed.

   b. A manual-reset or IML function has been initiated at the addressed CPU, and the function has not yet been completed. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

   If the currently specified order is sense, external call, emergency signal, start, stop, restart, or stop and store status, then the order is rejected, and condition code 2 is set. If the currently specified order is an IML, 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 in the section "Status Bits" in this chapter.

When any of the conditions described in items 3 and 4 exists, the addressed CPU is referred to as "busy." Busy is not indicated if the addressed CPU is in the check-stop state or when the operator-intervening condition exists. A CPU-busy condition is normally of short duration; however, the conditions described in item 3 may last indefinitely because of a string of interruptions, because of an incomplete READ DIRECT operation, 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 an IML.

When the conditions described in items 1 and 2 above do not apply and operator-intervening and receiver-check status conditions do not exist at the addressed CPU, reset orders may be accepted regardless of whether the addressed CPU has completed a previously accepted order. This may cause the previous order to be lost when it is only partially completed, making unpredictable whether the results defined for the lost order are obtained.

Status Bits

Various status conditions are defined whereby the issuing and addressed CPUs can indicate their responses to the specified order. The status conditions and their bit positions in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction are shown in the figure "Status Conditions."

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	Inoperative
30	Invalid order
31	Receiver check

Status Conditions

The status condition assigned to bit position 0 is generated by the CPU executing SIGNAL PROCESSOR. The remaining status conditions are generated by the addressed CPU.

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, unassigned bits of the status register are set to zeros, and the contents of other status bits are unpredictable. In this case, condition code 1 is set independent of whether the access path to the addressed CPU is busy and independent of whether the addressed CPU is not operational, is busy, or has presented zero status.

When the access path to the addressed CPU is not busy and the addressed CPU is operational and does not indicate busy to the currently specified order, the addressed CPU presents its status to the issuing CPU. These status bits are of two types:

1. Status bits 24-29 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 specified 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.

2. 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 order has been rejected, the status is stored at the issuing CPU in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction, zeros are stored in the unassigned bit positions of the register, and condition code 1 is set.

The status conditions are defined as follows:

Equipment Check: This condition exists when the CPU executing the instruction detects equipment malfunctioning that has affected only the execution of this instruction and the associated order. The order code may or may not have been transmitted and may or may not have been accepted, and the status bits provided by the addressed CPU may be in error.

External Call Pending: This condition exists when an external-call interruption condition is pending in the addressed CPU because of a previously issued SIGNAL PROCESSOR order. The condition exists from the time an external-call order is accepted until the resultant external interruption has been completed or a CPU reset occurs. The condition may be due to the issuing CPU or another CPU. The condition, when present, is indicated only in response to sense and to external call.

Stopped: This condition exists when the addressed CPU is in the stopped state. The condition, when present, is indicated only in response to sense. This condition cannot be reported as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

Operator Intervening: This condition exists when the addressed CPU is executing certain operations initiated from local or remote operator facilities. The particular manually initiated operations that cause this condition to be present depend on the model and on the order specified. On machines which do not implement the IML order, the conditions described under "Not Ready" may be indicated as an operator-intervening condition. The operator-intervening condition, when present, can be indicated in response to all orders. Operator intervening is indicated in response to sense if the condition is present and precludes the acceptance of any of the installed orders. The condition may also be indicated in response to unassigned or uninstalled orders. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

Check Stop: This condition exists when the addressed CPU is in the check-stop state. The condition, when present, is indicated only in response to sense, external call, emergency signal, start, stop, restart, and stop and store status. The condition may also be indicated in response to unassigned or uninstalled orders. This condition cannot be reported as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

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 IML. This condition cannot arise as a result of a SIGNAL PROCESSOR by a CPU addressing itself.

Inoperative: This condition indicates that the execution of the operation specified by the order code requires the use of a service processor which is inoperative. The failure of the service processor may have been previously reported by a service-processor-damage machine-check condition. The inoperative condition cannot occur for the sense, external-call, or emergency-signal order code.

Invalid Order: This condition exists during the communications associated with the execution of SIGNAL PROCESSOR when an unassigned or uninstalled order code is decoded.

Receiver Check: This condition exists when the addressed CPU detects malfunctioning of equipment during the communications associated with the execution of SIGNAL PROCESSOR. When this condition is indicated, the order has not been initiated, and, since the malfunction may have affected the generation of the remaining receiver status bits, these bits are not necessarily valid. A machine-check condition may or may not have been generated at the addressed CPU.

The following chart summarizes which status conditions are presented to the issuing CPU in response to each order code.

Status Condition

31 Receiver check≠
30 Invalid order
29 Inoperative
28 Not ready
27 Check stop
26 Operator intervening#
25 Stopped
24 External call pend.

Order	24	25	26	27	28	29	30	31
Sense	X	X	X	X	X	0	0	X
External call	X	0	X	X	X	0	0	X
Emergency signal	0	0	X	X	X	0	0	X
Start	0	0	X	X	X	X	0	X
Stop	0	0	X	X	X	X	0	X
Restart	0	0	X	X	X	X	0	X
Initial program reset	0	0	X	0	X	X	0	X
Program reset	0	0	X	0	X	X	0	X
Stop and store status	0	0	X	X	X	X	0	X
IML*	0	0	X	0	0	X	0	X
Initial CPU reset*	0	0	X	0	X	X	0	X
CPU reset*	0	0	X	0	X	X	0	X
Unassigned order	0	0	X	E	X	X	1	X

Explanation:

# The current state of the operator-intervening condition may depend on the order code that is being interpreted.

≠ If a one is presented in the receiver-check bit position, the values presented in the other bit positions are not necessarily valid.

* 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."

0 A zero is presented in this bit position regardless of the current state of this condition.

1 A one is presented in this bit position.

X A zero or a one is presented in this bit position, reflecting the current state of the corresponding condition.

E Either a zero or the current state of the corresponding condition is indicated.

If the presented status bits are all zeros, the order has been accepted, and the issuing CPU sets condition code 0. If one or more ones are presented, the order has been rejected, and the issuing CPU stores the status in the general register designated by the R_1 field of the SIGNAL PROCESSOR instruction and sets condition code 1.

Programming Notes

1. All SIGNAL PROCESSOR orders can be addressed to this same CPU. The following are examples of functions obtained by a CPU addressing SIGNAL PROCESSOR to itself:

   a. Sense indicates whether an external-call condition is pending.

   b. 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.

   c. Start sets condition code 0 and has no other effect.

   d. Stop causes the CPU to set condition code 0, take pending interruptions for which it is enabled, and enter the stopped state.

   e. Restart provides a means to store the current PSW.

   f. Stop and store status causes the machine to stop and store all current status.

2. Two CPUs can simultaneously execute SIGNAL PROCESSOR, with each CPU addressing the other. When this occurs, one CPU, but not both, can find the access path busy because of the transmission of the order code or status bits associated with SIGNAL PROCESSOR that is being executed by the other CPU. Alternatively, both CPUs can find the access path available and transmit the order codes to each other. In particular, two CPUs can simultaneously stop, restart, or reset each other.

CHANNEL-SET SWITCHING

The channel-set-switching facility permits a collection of channels to be switched from one CPU to another. The collection of channels which are switched as a group is called a channel set. A CPU can be connected to only one channel set at a time, and a channel set can be connected to only one CPU at a time. The switching operation controls only the execution of I/O instructions and I/O interruptions. Other channel activity, such as chaining and data-transfer operations, is not controlled by the switching.

When a channel set is switched to a particular CPU, it is said to be connected to that CPU. Channel-set switching permits any channel set in the configuration to be connected to any CPU in the configuration. However, a channel set can be connected to no more than one CPU at a time, and vice versa. When a channel set is not connected to a CPU, it is said to be disconnected. On a particular CPU, all I/O instructions executed address only the channels within the channel set which is currently connected to that CPU. Initial program reset and program reset issued to a CPU result in the resetting of the CPU and of only those channels which are currently connected to that CPU. Similarly, I/O interruptions caused by a channel which is part of a particular channel set occur on the CPU to which the channel set is currently connected. Chaining and data-transfer operations by the channel continue, independent of whether the channel set is connected to a CPU.

Channel sets can be connected and disconnected by means of two instructions, CONNECT CHANNEL SET and DISCONNECT CHANNEL SET, which are defined in Chapter 10, "Control Instructions." These instructions select a particular channel set by means of a 16-bit channel-set address. When the addressed channel set is not operational, execution of these instructions results in a setting of condition code 3. A channel set is not operational when it is not provided in the installation, its power is off, it is not in the configuration, or it is in any of certain customer-engineer test modes. Depending on the model, a channel set may be not operational when all of the channels in the channel set are not operational.

When a channel set is connected to a CPU and the CPU becomes not operational, the channel set may also become not operational, or it may become disconnected and remain in the configuration. A CPU can become not operational because of certain customer-engineer test modes being set, because model-dependent reconfiguration controls remove it from the configuration, or because its power is off.

The number of CPUs and channel sets in a particular configuration is not necessarily the same.

When system reset normal, system reset clear, load normal, or load clear is activated on any CPU in the configuration, in the absence of any override by model-dependent reconfiguration controls, then:

- All channels within all channel sets in the configuration perform system reset,

- Each channel set which has a home CPU which is operational and in the configuration is connected to its home CPU, and

- Each channel set which does not have a home CPU which is operational and in the configuration is disconnected.

By definition, the CPU to which a channel set is connected after subsystem reset is called the home CPU for that channel set. The address of the channel set may or may not be the same as the address of its home CPU.

When no channel set is connected to a particular CPU, the execution of any I/O instruction results in a setting of condition code 3. When a channel set is connected to a particular CPU, condition code 3 to an I/O instruction normally indicates that the addressed channel, subchannel, or device is not operational. The I/O instructions are described in Chapter 13, "Input/Output Operations." The connection or disconnection of a channel set is not considered to be a change in the channel state for purposes of setting to one the machine-check external-damage-code bit 3, channel not operational. The setting of this bit, even when a channel set is disconnected, indicates only those changes from the operational state to the not-operational state which would be seen if the channel set were connected to a CPU.

CHAPTER 5. PROGRAM EXECUTION

Normally, operation of the CPU is controlled by instructions in storage that are executed sequentially, one at a time, left to right in an ascending sequence of storage addresses. A change in the sequential operation may be caused by branching, LOAD PSW, interruptions, SIGNAL PROCESSOR orders, or manual intervention.

INSTRUCTIONS

Each instruction consists of two major parts:

- An operation code (op code), which specifies the operation to be performed
- The designation of the operands that participate

OPERANDS

Operands can be grouped in three classes: operands located in registers, immediate operands, and operands in storage. Operands may be either explicitly or implicitly designated.

Register operands can be located in general, floating-point, or control registers, with the type of register identified by the op code. The register containing the operand is specified by identifying the register in a four-bit field, called the R field, in the instruction. For some instructions, an operand is located in an implicitly designated register, the register being implied by the op code.

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

Operands in storage may have an implied length; be specified by a bit mask; be specified by a four-bit or eight-bit length specification, called the L field, in the instruction; or have a length specified by the contents of a general register. The addresses of operands in storage are specified by means of a format that uses the contents of a general register as part of the address. This makes it possible to:

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

The address used to refer to storage either is contained in a register designated by the R field in the instruction or is calculated from a base address, index, and displacement, specified by the B, X, and D fields, respectively, in the instruction.

To describe the execution of instructions, operands are designated as first and second operands and, in some cases, third operands.

In general, two operands participate in an instruction execution, and the result replaces the first operand. However, CONVERT TO DECIMAL, TEST BLOCK, and instructions with "store" in the instruction name (other than STORE THEN AND SYSTEM MASK and STORE THEN OR SYSTEM MASK) use the second-operand address to designate a location in which to store. TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP may perform an update on the second operand. Except when otherwise stated, the contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

INSTRUCTION FORMAT

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

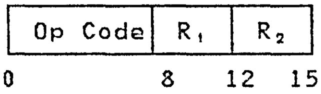
Some instructions contain fields that vary slightly from the basic format, and in some instructions the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

Those instruction formats which are unique to instructions associated with the vector facility are described in the publication _IBM System/370 Vector Operations_, SA22-7125.

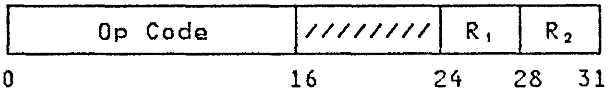
The format names indicate, in general terms, the classes of operands which participate in the operation:

- RR denotes a register-and-register operation.
- RRE denotes a register-and-register operation having an extended op-code field.
- RX denotes a register-and-indexed-storage operation.
- RS denotes a register-and-storage operation.
- SI denotes a storage-and-immediate operation.
- S denotes an operation using an implied operand and storage.
- SS denotes a storage-and-storage operation.
- SSE denotes a storage-and-storage operation having an extended op-code field.

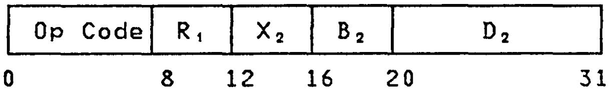
RR Format



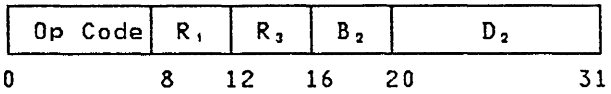
RRE Format



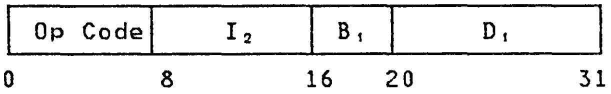
RX Format



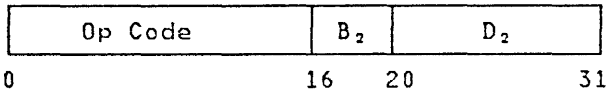
RS Format



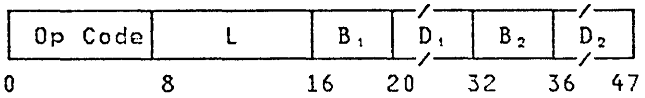
SI Format

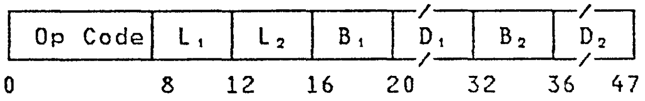


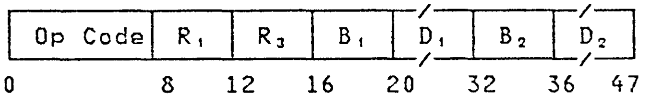
S Format



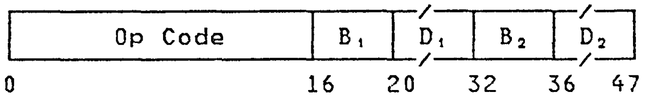
SS Format







SSE Format



Basic Instruction Formats

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

The first two bits of the first or only byte of the op code specify the length and format of the instruction, as follows:

Bit Positions 0-1	Instruction Length (in Halfwords)	Instruction Format
00	One	RR
01	Two	RX
10	Two	RRE/RS/RX/S/SI
11	Three	SS/SSE

In the format illustration for each individual instruction description, the op-code field shows the op code as hexadecimal digits within single quotes. The hexadecimal representation uses 0-9 for the binary codes 0000-1001 and A-F for the binary codes 1010-1111.

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

Register Operands

In the RR, RRE, RX, and RS formats, the contents of the register designated by the R_1 field are called the first operand. The register containing the first operand is sometimes referred to as the "first-operand location," and sometimes as "register R_1." In the RR and RRE formats, the R_2 field designates the register containing the second operand, and the R_2 field may designate the same register as R_1. In the RS format, the use of the R_3 field depends on the instruction.

The R field designates a general register in the general and control 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 specify the first operand, which is one byte in length.

Storage Operands

In the SI, SSE, 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, SSE, 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 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.

In the SS format, with two length fields given, L_1 specifies the number of additional operand bytes to the right of the byte designated by the first-operand address. Therefore, the length in bytes of the first operand is 1-16, corresponding to a length code in L_1 of 0-15. Similarly, L_2 specifies the number of additional operand bytes to the right of the location designated by the second-operand address. Results replace the first operand and are never stored outside the field specified by the address and length. If the first operand is longer than the second, the second operand is extended on the left with zeros up to the length of the first operand. This extension does not modify the second operand in storage.

In the SS format with two R fields, the contents of the general register specified by the R_1 field are a 32-bit unsigned value called the true length. The operands are of the same length, called the effective length. The effective length is equal to the true length or 256, whichever is less. The instructions using this format, which are MOVE TO PRIMARY, MOVE TO SECONDARY, and MOVE WITH KEY, set the condition code to facilitate programming a loop to move the total number of bytes specified by the true length.

ADDRESS GENERATION

Execution of instructions by the CPU involves generation of the addresses of instructions and operands. This section describes address generation as it applies to most instructions. In some instructions, the operation performed does not follow the general rules stated in this section. All of these exceptions are explicitly identified in the individual instruction descriptions.

SEQUENTIAL INSTRUCTION-ADDRESS GENERATION

When an instruction is fetched from the location designated by the current PSW, the instruction address is increased by the number of bytes in the instruction, and the instruction is executed. The same steps are then repeated by using the new value of the instruction address to fetch the next instruction in the sequence.

Instruction addresses wrap around, with the halfword at instruction address $2^{24} - 2$ being followed by the halfword at instruction address 0. Thus, any carry out of PSW bit position 40, as a result of updating the instruction address, is lost.

OPERAND-ADDRESS GENERATION

An operand address that refers to storage either is contained in a register designated by an R field in the instruction or is calculated from the sum of three binary numbers: base address, index, and displacement.

The base address (B) is a 24-bit number contained in a general register specified by the program in a four-bit field, called the B field, in the instruction. Base addresses can be used as a means of independently addressing each program and data area. In array-type calculations, it can designate the location of an array, and, in record-type processing, it can identify the record. The base address provides for addressing the entire storage. The base address may also be used for indexing.

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

The displacement (D) is a 12-bit number contained in a field, called the D field, in the instruction. The displacement provides for relative addressing of up to 4,095 bytes beyond the location designated by the base address. In array-type calculations, the displacement can be used to specify one of many items associated with an element. In the processing of records, the displacement can be used to identify items within a record.

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

A zero in any of the B_1, B_2, or X_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.

When an instruction description specifies that the contents of a general register designated by an R field are used to address an operand in storage, bit positions 8-31 of the register provide the operand address. For the instructions INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY EXTENDED, and TEST BLOCK, bits 1-31 of the register provide the address.

An instruction can designate the same general register both for address computation and as the location of an operand. Address computation is completed before registers, if any, are changed by the operation.

Unless otherwise indicated in an individual instruction definition, the generated operand address designates the leftmost byte of an operand in storage.

Programming Note

Negative values may be used in index and base-address registers. Bits 0-7 of these values are always ignored.

BRANCH-ADDRESS GENERATION

For branch instructions, the address of the next instruction to be executed when the branch is taken is called the branch address. Depending on the branch instruction, the instruction format may be RR, RS, or RX.

In the RS and RX formats, the branch address is specified by a base address, a displacement, and, for RX, an index. In the RS and RX formats, the branch address generation follows the normal rules for operand-address generation.

In the RR format, the contents of bit positions 8-31 of the general register designated by the R_2 field are used as the branch address, and bits 0-7 of the register are ignored. General register 0 cannot be designated as containing a branch address. A value of zero in the R_2 field causes the instruction to be executed without branching.

For several branch instructions, branching depends on satisfying a specified condition. When the condition is not satisfied, the branch is not taken, normal sequential instruction execution continues, and the branch address is not used. When a branch is taken, bits 8-31 of the branch address replace bits 40-63 of the current PSW. The branch address is not used to access storage as part of the branch operation.

A specification exception due to an odd branch address and access exceptions due to fetching of the instruction at the branch location are not recognized as part of the branch operation but instead are recognized as exceptions associated with the execution of the instruction at the branch location.

A branch instruction, such as BRANCH AND LINK, can designate the same general register for branch-address computation and as the location of an operand. Branch-address computation is completed before the remainder of the operation is executed.

INSTRUCTION EXECUTION AND SEQUENCING

The program-status word (PSW), described in Chapter 4, "Control," contains information required for proper program execution. The PSW is used to control instruction sequencing and to hold and indicate the status of the CPU in relation to the program currently being executed. The active or controlling PSW is called the current PSW.

Branch instructions perform the functions of decision making, loop control, and subroutine linkage. A branch instruction affects instruction sequencing by introducing a new instruction address into the current PSW.

DECISION MAKING

Facilities for decision making are provided by BRANCH ON CONDITION. This instruction inspects a condition code that reflects the result of a majority of the arithmetic, logical, and I/O operations. The condition code, which consists of two bits, provides for four possible condition-code settings: 0, 1, 2, and 3.

The specific meaning of any setting depends on the operation that sets the condition code. For example, the condition code reflects such conditions as zero, nonzero, first operand high, equal, overflow, and channel busy. Once set, the condition code remains unchanged until modified by an instruction that causes a different condition code to be set. See Appendix C, "Condition-Code Settings," for a summary of the instructions which set the condition code.

LOOP CONTROL

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

SUBROUTINE LINKAGE

Subroutine linkage is provided by the BRANCH AND LINK and BRANCH AND SAVE instructions, which permit not only the introduction of a new instruction address but also the preservation of the return address and associated information. Linkage between a problem-state program and the supervisor or monitoring program is provided by means of the SUPERVISOR CALL and MONITOR CALL instructions.

The instructions PROGRAM CALL and PROGRAM TRANSFER provide the facility for linkage between programs of different authority and in different address spaces. PROGRAM CALL permits linkage to a number of preassigned programs that may be in either the problem or the supervisor state and may be in either the same address space or an address space different from that of the caller. In general, it is used to transfer control to a program of higher authority. PROGRAM TRANSFER permits a change of the instruction address and address space. PROGRAM TRANSFER also permits a reduction in PSW-key-mask authority and a change from the supervisor to the problem state. In general, it is used to transfer control from one program to another of equal or lower authority. PROGRAM TRANSFER can be used to return from a program called by PROGRAM CALL.

The operation of PROGRAM CALL is controlled by means of an entry-table entry, which is located as part of a table-lookup process during the execution of the instruction. The instruction causes the primary address space to be changed only when the ASN in the entry-table entry is nonzero. When the primary address space is changed, the operation is called PROGRAM CALL with space switching (PC-ss). When the primary address space is not changed, the operation is called PROGRAM CALL to current primary (PC-cp).

PROGRAM TRANSFER specifies the address space which is to become the new primary address space. When the primary address space is changed, the operation is called PROGRAM TRANSFER with space switching (PT-ss). When the primary address space is not changed, the operation is called PROGRAM TRANSFER to current primary (PT-cp).

The linkage instructions provided and the functions performed by each are summarized in the figure "Linkage-Instruction Summary."

		Instruction Address PSW Bits 40-63		Problem State PSW Bit 15		PASN CR4 Bits 16-31		PSW-Key Mask Changed in CR3
Instruction	Format	Save	Set	Save	Set	Save	Set	
BALR*	RR	Yes	R_2[1]	-	-	-	-	-
BAL*	RX	Yes	Yes	-	-	-	-	-
BASR	RR	Yes	R_2[1]	-	-	-	-	-
BAS	RX	Yes	Yes	-	-	-	-	-
MC#[2]	SI	Yes	Yes	Yes	Yes	-	-	-
PC-cp	S	Yes	Yes	Yes	Yes	-	-	"OR" EKM
PC-ss	S	Yes	Yes	Yes	Yes	Yes	Yes	"OR" EKM
PT-cp	RRE	-	R_2	-	R_2**	-	-	"AND" R_1
PT-ss	RRE	-	R_2	-	R_2**	-	Yes	"AND" R_1
SVC[2]	RR	Yes	Yes	Yes	Yes	-	-	-

Explanation:

- No

* The instruction-length code, condition code, and program mask are also saved.

** A change from the supervisor to the problem state is allowed; a privileged operation exception is recognized when a change from the problem to the supervisor state is specified.

# Monitor-mask bits provide a means of disallowing linkage, or enabling linkage, for selected classes of events.

[1] The action takes place only if the R_2 field in the instruction is nonzero.

[2] MC and SVC, as part of the interruption, save the entire current PSW and load a new PSW.

Linkage-Instruction Summary

INTERRUPTIONS

Interruptions permit the CPU to change state as a result of conditions external to the system, in channels or input/output (I/O) devices, in other CPUs, or in the CPU itself. Details are to be found in Chapter 6, "Interruptions."

Six classes of interruption conditions are provided: external, I/O, machine check, program, restart, and supervisor call. Each class has two related PSWs, called old and new, in permanently assigned real storage locations. In all classes, an interruption involves storing information identifying the cause of the interruption, storing the current PSW at the old-PSW location, and fetching the PSW at the new-PSW location, which becomes the current PSW.

The old PSW contains CPU-status information necessary for resumption of the interrupted program. At the conclusion of the program invoked by the interruption, the instruction LOAD PSW may be used to restore the current PSW to the value of the old PSW.

TYPES OF INSTRUCTION ENDING

Instruction execution ends in one of five ways: completion, nullification, suppression, termination, and partial completion.

Partial completion of instruction execution occurs only for interruptible instructions; it is described in the section "Interruptible Instructions" later in this chapter.

Completion

Completion of instruction execution provides results as called for in the definition of the instruction. When an interruption occurs after the completion of the execution of an instruction, the instruction address in the old PSW designates the next sequential instruction.

Suppression

Suppression of instruction execution causes the instruction to be executed as if it specified "no operation." The contents of any result fields, including the condition code, are not changed. The instruction address in the old PSW on an interruption after suppression designates the next sequential instruction.

Nullification

Nullification of instruction execution has the same effect as suppression, except that when an interruption occurs after the execution of an instruction has been nullified, the instruction address in the old PSW designates the instruction whose execution was nullified (or an EXECUTE instruction, as appropriate) instead of the next sequential instruction.

Termination

Termination of instruction execution causes the contents of any fields due to be changed by the instruction to be unpredictable. The operation may replace all, part, or none of the contents of the designated result fields and may change the condition code if such change is called for by the instruction. Unless the interruption is caused by a machine-check condition, the validity of the instruction address in the PSW, the interruption code, and the ILC are not affected, and the state or the operation of the machine is not affected in any other way. The instruction address in the old PSW on an interruption after termination designates the next sequential instruction.

INTERRUPTIBLE INSTRUCTIONS

Point of Interruption

For most instructions, the entire execution of an instruction is one operation. An interruption is permitted between operations; that is, an interruption can occur after the performance of one operation and before the start of a subsequent operation.

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

COMPARE LOGICAL LONG
MOVE LONG
TEST BLOCK
Interruptible instructions of the vector facility (see the publication _IBM System/370 Vector Operations_, SA22-7125)

The execution of an interruptible instruction is considered to consist in the execution of a number of units of operation, and an interruption is permitted between units of operation. The amount of data processed in a unit of operation depends on the particular instruction and may depend on the model and on the particular condition that causes the execution of the instruction to be interrupted.

Whenever points of interruption that include those occurring within the execution of an interruptible instruction are discussed, the term "unit of operation" is used. For a noninterruptible instruction, the entire execution consists, in effect, in the execution of one unit of operation.

When an instruction consists of a number of units of operation and an interruption occurs after some, but not all, units of operation have been completed, the instruction is said to be partially completed. In this case, the type of ending (completion, inhibition, nullification, suppression) is associated with the unit of operation. In the case of termination, the entire instruction is terminated, not just the unit of operation.

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, inhibition, nullification, or suppression of a unit of operation, all preceding units of operation have been completed, and subsequent units of operation and instructions have not been started. The main difference between these types of ending is the handling of the current unit of operation and whether the instruction address stored in the old PSW identifies the current instruction or the next sequential instruction.

At the time of an interruption, changes to register contents, which are due to be made by an interruptible vector instruction beyond the point of interruption, have not yet been made. Changes to storage locations, however, which are due to be made by an interruptible vector instruction beyond the point of interruption, may have occurred for one or more storage locations beyond the location containing the element identified by the interruption parameters, but not for any location beyond the last element specified by the instruction and not for any locations for which access exceptions exist. Changes to storage locations or register contents which are due to be made by instructions following the interrupted instruction have not yet been made at the time of interruption.

Completion: On completion of the last unit of operation of an interruptible instruction, the instruction address in the old PSW designates the next sequential instruction. The result location for the current unit of operation has been updated. It depends on the particular instruction how the operand parameters are adjusted. On completion of a unit of operation other than the last one, the instruction address in the old PSW designates the interrupted instruction or an EXECUTE instruction, as appropriate. The result location for the current unit of operation has been updated. The operand parameters are adjusted such that the execution of the interrupted instruction is resumed from the point of interruption when the old PSW stored during the interruption is made the current PSW.

Inhibition: When a unit of operation is inhibited, the instruction address in the old PSW designates the interrupted instruction or an EXECUTE instruction, as appropriate. The result location for the current unit of operation is not changed. The operand parameters are adjusted such that, if the instruction is reexecuted, execution of the interrupted instruction is resumed with the next unit of operation. Inhibition occurs only during interruptible vector instructions and is described in more detail in the publication IBM System/370 Vector Operations, SA22-7125.

Nullification: When a unit of operation is nullified, the instruction address in the old PSW designates the interrupted instruction or an EXECUTE instruction, as appropriate. The result location for the current unit of operation remains unchanged. The operand parameters are adjusted such that, if the instruction is reexecuted, execution of the interrupted instruction is resumed with the current unit of operation.

Suppression: When a unit of operation is suppressed, the instruction address in the old PSW designates the next sequential instruction. The operand parameters, however, are adjusted so as to indicate the extent to which instruction execution has been completed. If the instruction is reexecuted after the conditions causing the suppression have been removed, the execution is resumed with the current unit of operation.

Termination: When an exception which causes termination occurs as part of a unit of operation of an interruptible instruction, the entire operation is terminated, and the contents, in general, of any fields due to be changed by the instruction are unpredictable. On such an interruption, the instruction address in the old PSW designates the next sequential instruction.

The differences among the five types of ending for a unit of operation are summarized in the figure "Types of Ending for a Unit of Operation."

Unit of Operation Is	Instruction Address	Operand Parameters	Current Result Location
Completed			
Last unit of operation	Next instruction	Depends on the instruction	Changed
Any other unit of operation	Current instruction	Next unit of operation	Changed
Inhibited	Current instruction	Next unit of operation	Unchanged
Nullified	Current instruction	Current unit of operation	Unchanged
Suppressed	Next instruction	Current unit of operation	Unchanged
Terminated	Next instruction	Unpredictable	Unpredictable

Types of Ending for a Unit of Operation

Programming Notes

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

2. The amount of data processed in a unit of operation of an interruptible instruction depends on the model and may depend on the type of condition which causes the execution of the instruction to be interrupted or stopped. Thus, when an interruption occurs at the end of the current unit of operation, the length of the unit of operation may be different for different types of interruptions. Also, when the stop function is requested during the execution of an interruptible instruction, the CPU enters the stopped state at the completion of the execution of the current unit of operation. Similarly, in the instruction-step mode, only a single unit of operation is performed, but the unit of operation for the various cases of stopping may be different.

EXCEPTIONS TO NULLIFICATION AND SUPPRESSION

In certain unusual situations, the result fields of an instruction having a store-type operand are changed in spite of the occurrence of an exception which would normally result in nullification or suppression. These situations are exceptions to the general rule that the operation is treated as a no-operation when an exception requiring nullification or suppression is recognized. Each of these situations may result in the turning on of the change bit associated with the store-type operand, even though the final result in storage may appear unchanged. Depending on the particular situation, additional effects may be observable. The extent of these effects is described along with each of the situations.

All of these situations are limited to the extent that a store access does not occur and the change bit is not set when the store access is prohibited. For the CPU, a store access is prohibited whenever an access exception exists for that access, or whenever an exception exists which is of higher priority than the priority of an access exception for that access.

When, in these situations, an interruption for an exception requiring suppression occurs, the instruction address in the old PSW designates the next sequential instruction. When an interruption for an exception requiring nullification occurs, the instruction address in the old PSW designates the instruction causing the exception even though partial results may have been stored.

Storage Change and Restoration for DAT-Associated Access Exceptions

In this section, the term "DAT-associated access exceptions" is used to refer to those exceptions which may occur as part of the dynamic-address-translation process. These exceptions are page translation, segment translation, translation specification, and addressing due to a DAT-table entry being designated at a location that is not available in the configuration. The first two of these exceptions normally cause nullification, and the last two normally cause suppression. Protection exceptions, including those due to segment protection, are not considered to be DAT-associated access exceptions.

For DAT-associated access exceptions, on some models, channels may observe the effects on storage as described in the following case.

When, for an instruction having a store-type operand, a DAT-associated access exception is recognized for any operand of the instruction, that portion, if any, of the store-type operand which would not cause an exception may be changed to an intermediate value but is then restored to the original value.

The accesses associated with storage change and restoration for DAT-associated access exceptions are only observable by channels and are not observable by other CPUs in a multiprocessing configuration. Except for instructions which are defined to have multiple-access operands, the intermediate value, if any, is always equal to what would have been the final value if the DAT-associated access exception had not occurred.

Programming Notes

1. Storage change and restoration for DAT-associated access exceptions occur in two main situations:

   a. The exception is recognized for a portion of a store-type operand which crosses a page boundary, and the other portion has no access exception.

   b. The exception is recognized for one operand of an instruction having two storage operands (for example, an SS-format instruction or MOVE LONG), and the other operand, which is a store-type operand, has no access exception.

2. To avoid letting a channel observe intermediate operand values due to storage change and restoration for DAT-associated access exceptions (especially when a CCW chain is modified), the CPU program should do one of the following:

   - Operate on one storage page at a time

   - Perform preliminary testing to ensure that no exceptions occur for any of the required pages

   - Operate with DAT off

Modification of DAT-Table Entries

When a valid and attached DAT-table entry is changed to a value which would cause an exception, and when, before the TLB is cleared of entries which qualify for substitution for that entry, an attempt is made to refer to storage by using a virtual address requiring that entry for translation, the contents of any fields due to be changed by the instruction are unpredictable. Results, if any, associated with the virtual address whose DAT-table entry was changed may be placed in those real locations originally associated with the address. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable. On some machines, this situation may be reported by means of an instruction-processing-damage machine check with the delayed-access-exception bit also indicated.

Trial Execution for Editing Instructions and TRANSLATE

For the instructions EDIT, EDIT AND MARK, and TRANSLATE, the portions of the operands that are actually used in the operation may be established in a trial execution for operand accessibility that is performed before the execution of the instruction is started. This trial execution consists in an execution of the instruction in which results are not stored. If the first operand of TRANSLATE or either operand of EDIT or EDIT AND MARK is changed by another CPU or by a channel, after the initial trial execution but before completion of execution, the contents of any fields due to be changed by the instruction are unpredictable. Furthermore, it is unpredictable whether or not an interruption occurs for an access exception that was not initially applicable.

Interlocked Update for Nullification and Suppression

When an exception which is defined to cause suppression or nullification is recognized for an instruction with a store-type operand, an interlocked-update reference which does not change the contents of the location may occur for that portion, if any, of the store-type operand for which no access exception exists. The interlocked-update reference can occur only if the priority of the exception is equal to or lower than the priority of an access exception for the store-type operand.

When the exception is a specification exception for a store-type operand which requires alignment on integral boundaries, the interlocked-update reference which may occur is limited to the single byte at the location designated by the operand address.

Programming Note

The update appears to be an interlocked-update reference as observed by other CPUs. It is not interlocked as observed by channels. Examples of when an interlocked-update reference may occur to the destination-operand location in storage are:

- Specification exception for an odd register number for COMPARE DOUBLE AND SWAP
- Data exception for an invalid decimal sign for ADD DECIMAL
- Decimal-divide exception for DIVIDE DECIMAL

DUAL-ADDRESS-SPACE CONTROL

The dual-address-space (DAS) facility consists of a number of interrelated functions. Some of these functions are described in this chapter, specifically in the sections "DAS-Authorization Mechanisms" and "PC-Number Translation." Additionally, address spaces, ASN translation, and ASN authorization are described in Chapter 3, "Storage"; DAS tracing in Chapter 4, "Control"; interruptions in Chapter 6, "Interruptions"; and the instructions in Chapter 10, "Control Instructions."

A complete list of the functions, control-register fields, and instructions that are part of DAS is included in Appendix D, "Facilities."

SUMMARY

These major functions are provided:

1. Two address spaces for immediate use by the program
2. Means for changing to other spaces
3. Three instructions for moving information
4. A table-based subroutine-linkage mechanism
5. The use of multiple access keys for key-controlled protection by problem programs
6. Aids for program-problem analysis

Additionally, control and authority mechanisms are incorporated to control these functions.

These functions are intended for use by programs considered to be semiprivileged, that is, programs which are executed in the problem state but which may be authorized to use additional capabilities. The authorization mechanisms provided with DAS are described in the section "DAS Authorization Mechanisms" in this chapter.

The 11 instructions which are included as part of DAS are described in Chapter 10, "Control Instructions." DAS includes the privileged instruction LOAD ADDRESS SPACE PARAMETERS and the following semiprivileged instructions:

EXTRACT PRIMARY ASN
EXTRACT SECONDARY ASN
INSERT ADDRESS SPACE CONTROL
INSERT VIRTUAL STORAGE KEY
MOVE TO PRIMARY
MOVE TO SECONDARY
MOVE WITH KEY
PROGRAM CALL
PROGRAM TRANSFER
SET ADDRESS SPACE CONTROL
SET SECONDARY ASN

In addition, when DAS is installed, two instructions which are not part of DAS are changed to be semiprivileged. These instructions are:

INSERT PSW KEY
SET PSW KEY FROM ADDRESS

The changes to the operation of these two instructions are under the control of mode bits in the PSW or in control registers. Whenever a program in the problem state attempts to execute any of the 13 instructions at a time when the required control registers have not been set up, a program exception is indicated which is also available on machines without DAS.

DAS FUNCTIONS

Using Two Address Spaces

Primary and Secondary Address Spaces: DAS makes two address spaces available for use by a semiprivileged program. The use of control register 1 to contain the designation of a segment table for one address space, called the primary address space, is the same as when DAS is not installed. Control register 1 is used when translating primary virtual addresses. For the other address space, called the secondary address space, a segment-table designation is contained in control register 7. Control register 7 is used when translating secondary virtual addresses. DAT applies in the same way to both address spaces.

Address-Space Control: When the address-space-control bit, bit 16 of the EC-mode PSW, is one and DAT is on, the CPU is said to be in the secondary-space mode. When the CPU is in the secondary-space mode, those operand addresses defined to be logical refer to the secondary address space. When the CPU is in the secondary-space mode, it is unpredictable whether instructions are fetched from the primary address space or from the secondary address space. Programs which are executed in this mode are expected to reside in a portion of an address space which is shared between the primary address space and the secondary address space.

The instruction SET ADDRESS SPACE CONTROL provides the semiprivileged program with the capability of selecting either the primary-space mode or the secondary-space mode when DAT is on. Since logical addresses are translated as primary virtual addresses when the CPU is in the primary-space mode and as secondary virtual addresses when the CPU is in the secondary-space mode, the semiprivileged program can use the entire set of unprivileged and semiprivileged instructions to access information in either of the two address spaces. The instruction INSERT ADDRESS SPACE CONTROL provides the program with the ability to inspect the state of the address-space-control bit.

In addition to the function of accessing operands in one address space or the other, the instructions MOVE TO PRIMARY and MOVE TO SECONDARY provide a means of moving data from either of the two address spaces to the other.

Changing to Other Spaces

Address-Space Numbers: DAS provides for changing both the primary address space and the secondary address space. Each address space is designated by a 16-bit value, called the address-space number, or ASN. The ASN can be used as a primary ASN (PASN) or a secondary ASN (SASN). These two values are not used directly to access an address space but are used as symbolic identifiers of the address space.

Bits 16-31 of control register 4 contain the PASN. The PASN can be loaded by means of a PROGRAM CALL with space switching, a PROGRAM TRANSFER with space switching, or LOAD ADDRESS SPACE PARAMETERS. The PASN can be inspected by EXTRACT PRIMARY ASN. When the PASN is loaded by means of the DAS instructions, the corresponding segment-table-designation (STD) value is placed in the primary segment-table designation (PSTD), bits 0-31 of control register 1. The PASN can also be loaded by means of LOAD CONTROL, in which case no translation occurs to convert the PASN to an STD value.

Bits 16-31 of control register 3 contain the SASN. The SASN can be loaded by means of the SET SECONDARY ASN instruction and LOAD ADDRESS SPACE PARAMETERS. The SASN can be inspected by EXTRACT SECONDARY ASN. When the SASN is loaded by means of the DAS instructions, the corresponding STD value is placed in the secondary segment-table designation (SSTD), bits 0-31 of control register 7. The SASN can also be loaded by means of LOAD CONTROL, in which case no translation occurs to convert the SASN to an STD value.

Address-Space-Number Translation: By using the instructions SET SECONDARY ASN and PROGRAM TRANSFER, the semiprivileged program can specify, by reference to a general register containing an ASN, a particular address space which is to be accessed. The ASN specified by the program is used in a table-lookup process, which locates the address-space-control parameters that in turn are used to permit controlled access to the address space. The table lookup includes an authorization test to ensure that the program is authorized to use the specified address space. The table lookup, including the authorization test and the conversion to system-usable form, is called ASN translation. The same table lookup, but without the authorization test, is performed by the PROGRAM CALL instruction on the ASN specified in the entry-table entry. The instruction LOAD ADDRESS SPACE PARAMETERS also uses ASN translation.

To obtain the segment-table designation and other information for the new address space, the ASN is translated by using a set of tables whose origin is contained in control register 14. A two-level lookup is used. The ASN value is partitioned into two indexes. The first index selects an entry in the table designated by control register 14, called the ASN first table, or AFT. This entry designates another table, called the ASN second table, or AST, an entry of which is selected by the second index. An entry in the second table contains several parameters about the new address space. The information in a second-table entry includes:

- A validity indicator, generally used to indicate whether the associated address space is immediately accessible. This is useful for managing unassigned numbers and swapped-out spaces.
- The origin and length of a table which provides control over whether three of the DAS instructions are authorized to use the new ASN. This table is called the authority table (AT).
- The authorization index (AX), or level, of the new space.
- The origin and length of the segment table to be used by DAT when the new address space is accessed.
- A control over whether a signal, in the form of a space-switch-event program interruption, is given for two of the DAS instructions after a change to a new primary address space is completed.
- The origin of a set of tables which describe the entry points associated with a new primary space. These tables are used by the linkage mechanism provided with DAS. A two-level table structure is provided. The first level is the linkage table (LT), whose entries provide the origins of entry tables (ET).

<u>Changing the Secondary Address Space</u>: The SET SECONDARY ASN instruction causes the secondary address space to be changed to the address space associated with the ASN specified by the instruction. The ASN itself is placed in control register 3 and is called the secondary ASN, or SASN. The ASN is translated to obtain the segment-table designation for the space. This designation is placed in control register 7 as the secondary segment-table designation (SSTD). Instruction execution is disallowed if the translation is not authorized. The translation is authorized by a bit in the authority table at an offset determined by the authorization index in control register 4. The instruction LOAD ADDRESS SPACE PARAMETERS also can change the secondary address space.

Moving Information

DAS provides three instructions for moving information under the control of two access keys.

The instructions MOVE TO PRIMARY and MOVE TO SECONDARY permit the semiprivileged program to move data from either of the two current address spaces to the other. These instructions are defined such that a second access key can be specified in addition to the PSW key. The PSW key in these two instructions is used as the access key for the storage references to the primary address space. Accesses to the secondary address space are made by using a key specified in a general register designated by the instruction. Thus, the semiprivileged program can use the instruction to move data between a calling program area and the semiprivileged-program area and to specify the appropriate key to be used in each area.

A third move instruction, MOVE WITH KEY, gives a semiprivileged program the capability of moving information between a caller-specified area and a semiprivileged-program area in the same address space. The instruction uses the PSW key for the store accesses associated with the first operand and uses a program-specified key for the fetch accesses associated with the second operand. Thus, a semiprivileged program may set up the PSW key and specify the source key so as to provide appropriate authority checking on a caller-specified address whether it be a source or a target.

For all three move instructions, the number of bytes to be moved is expressed as a true length. A zero length is allowed, with no movement performed. Up to 256 bytes are moved each time one of these instructions is executed, and a condition code is set to indicate whether the number of bytes moved did or did not exhaust the true length. These capabilities make the instructions suitable for use in a simple program to move any number of bytes. This is particularly useful when the number of bytes to be moved must be calculated by the program.

Transferring Program Control

DAS permits programs operating at different levels of authority to be linked directly without the use of the SUPERVISOR CALL or MONITOR CALL instruction. The instructions PROGRAM CALL and PROGRAM TRANSFER provide a protected mechanism for transferring control between programs operating at different levels, or the same level, of control.

The PROGRAM CALL instruction specifies a 20-bit index, the PC number, which is used to locate the information associated with the program to be called. This information, called the entry information, includes an authorization key mask, an entry key mask to be ORed into the PSW-key mask in control register 3, the information to be loaded into the problem-state and instruction-address portions of the current PSW, and a parameter which is made available to the called program in general register 4. The entry information can also cause an optional space-switching operation to occur. The space-switching operation is specified when a nonzero address-space number (ASN) is provided as part of the entry information. When space switching occurs, the operation is called PROGRAM CALL with space switching (PC-ss). When no space switching occurs, the operation is called PROGRAM CALL to current primary (PC-cp).

The information associated with the program to be called is obtained by means of a two-level lookup:

- The first lookup consists in indexing into the linkage table to obtain a linkage-table entry, which contains an entry-table address.

- The second lookup consists in indexing into the entry table to obtain an entry-table entry, which contains the entry information.

Since the information loaded into the PSW and control registers is obtained from tables set up by the control program, system integrity is maintained because the problem program cannot load arbitrary values. The current values of the PSW-key mask and the PASN are saved in general register 3. The problem-state status and instruction address are saved in general register 14.

A program can use PC-ss to call a program in another address space. In addition to isolating programs in address spaces, this operation provides for a change to a higher level of privilege and authority. Thus, the called program is entered with an authorization index that can permit access to address spaces which are not authorized to the caller, and with a different linkage table. The called program can then perform services for the calling program by having easy access to these other address spaces, without the requirement that the calling program also have access to these address spaces, and it may use program services which are not available to the calling program. A hierarchy of control can be established and the integrity of the address spaces maintained.

PROGRAM TRANSFER may be used to restore the information saved by PROGRAM CALL. It ANDs information into the PSW-key mask in control register 3 and loads the problem-state status and instruction address into the current PSW. However, PROGRAM TRANSFER cannot be used to change from the problem to the supervisor state. Like PROGRAM CALL, PROGRAM TRANSFER is described in terms of two cases: PROGRAM TRANSFER with space switching (PT-ss) and PROGRAM TRANSFER to current primary (PT-cp). PT-ss occurs when the specified ASN is different from the current PASN.

PT-ss provides the return function to be used by a program which has been called by means of PC-ss. The authorization checking provided on PT-ss permits a table structure to be set up which prohibits a program from increasing its authority. PT-ss can also be used to transfer control from one address space to another of the same authority.

PROGRAM CALL and PROGRAM TRANSFER are valid only when the CPU is in the primary-space mode. They cause a special-operation exception to be recognized when the CPU is in the secondary-space mode or the real mode.

Handling Storage Keys and the PSW Key

The handling of keys is facilitated by instructions for changing and extracting the PSW key in the problem state. A semiprivileged instruction is provided for obtaining the storage key associated with a virtual-storage location.

INSERT PSW KEY, which is changed by DAS to be semiprivileged, permits a semiprivileged program to save the current PSW key for later restoration.

INSERT VIRTUAL STORAGE KEY permits the semiprivileged program to determine the storage key associated with any particular virtual-storage location. It may be used, for example, when one program, with authority to more than one key, calls another program and passes the address of a location to be used as either an input or output buffer. The called program must determine the key needed to access the buffer.

INSERT VIRTUAL STORAGE KEY is also useful to the control program since the instruction uses a virtual rather than a real address. The sequence LOAD REAL ADDRESS followed by INSERT STORAGE KEY or INSERT STORAGE KEY EXTENDED does not necessarily produce a valid result if the program is enabled for interruptions or operating in a multiprocessing configuration. This could be the case, for example, in a multiprocessing configuration if another CPU executed INVALIDATE PAGE TABLE ENTRY, followed by a reassignment of the page.

SET PSW KEY FROM ADDRESS, which is changed by DAS to be semiprivileged, provides the semiprivileged program with the capability of changing the PSW key, under control of the PSW-key mask, and thus permits the program to access different data areas protected by different keys.

Increased flexibility in key handling is controlled by a 16-bit PSW-key mask in control register 3. The PSW-key mask permits the semiprivileged program to operate with more than one key without having authorization to all keys. This mask controls the semiprivileged-program use of keys in MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH KEY, and SET PSW KEY FROM ADDRESS. Each bit position corresponds to a key value. The bit in the mask must be one in order for the corresponding key to be used.

Program-Problem Analysis

To aid program-problem analysis, the option is provided of having a trace entry made implicitly for three DAS instructions. When tracing is activated, a trace entry is made each time PROGRAM CALL, PROGRAM TRANSFER, or SET SECONDARY ASN is executed.

As a further analysis aid, PROGRAM CALL and PROGRAM TRANSFER are also recognized for PER purposes as successful branching events. Additionally, for these two instructions, the space-switch-event-control bit is provided both in control register 1 and in the second-table entry used during ASN translation. When either bit is one, a program interruption for a space-switch event occurs at the completion of the instruction. The effect is to provide for an interruption when a primary-space switch occurs, allowing recognition that a space has been entered, left, or both.

DAS AUTHORIZATION MECHANISMS

The DAS authorization mechanisms which are described in this section permit the control program to establish the degree of function which is provided to a particular semiprivileged program. (A summary of the authorization mechanisms is given in the figure "Summary of DAS Authorization Mechanisms.") The DAS authorization mechanisms are intended for use by programs considered to be semiprivileged, that is, programs which are executed in the problem state but which may be authorized to use additional capabilities. With these authorization controls, a hierarchy of programs may be established, with programs at a higher level having a greater degree of privilege or authority than programs at a lower level. The range of functions available at each level, and the ability to transfer control from a lower to a higher level, are specified in tables which are managed by the control program.

Programming Note

The DAS authorization mechanisms are defined such that if zeros are placed in the previously unassigned control-register positions, a problem program attempting to use the semiprivileged instructions causes a privileged-operation or special-operation exception to be recognized.

Mode Requirements

Most of the DAS instructions can be executed only with DAT on. PROGRAM CALL and PROGRAM TRANSFER are valid only in the primary-space mode. When a DAS instruction is executed in an invalid translation mode, a special-operation exception is recognized.

PROGRAM TRANSFER specifies a new value for the problem-state bit in the PSW. If a program in the problem state attempts to execute PROGRAM TRANSFER and set the supervisor state, a privileged-operation exception is recognized.

Extraction-Authority Control

The extraction-authority-control bit is located in bit position 4 of control register 0. In the problem state, bit 4 must be one to allow completion of these instructions:

EXTRACT PRIMARY ASN
EXTRACT SECONDARY ASN
INSERT ADDRESS SPACE CONTROL
INSERT PSW KEY
INSERT VIRTUAL STORAGE KEY

Otherwise, a privileged-operation exception is recognized. The extraction-authority-control bit is not examined in the supervisor state.

PSW-Key Mask

The PSW-key mask consists of bits 0-15 in control register 3. These bits are used in the problem state to control which keys and entry points are authorized for the program. The PSW-key mask is modified by PROGRAM CALL and PROGRAM TRANSFER and is loaded by LOAD ADDRESS SPACE PARAMETERS. The PSW-key mask is used in the problem state to control the following:

- The PSW-key values that can be set by means of the instruction SET PSW KEY FROM ADDRESS.
- The PSW-key values that are valid for the three move instructions that specify a second access key: MOVE TO PRIMARY, MOVE TO SECONDARY, and MOVE WITH KEY.
- The entry points which can be called by means of PROGRAM CALL. In this case, the PSW-key mask is ANDed with the authorization key mask in the entry-table entry, and, if the result is zero, the program is not authorized.

When an instruction in the problem state attempts to use a key not authorized by the PSW-key mask, a privileged-operation exception is recognized. The same action is taken when an instruction in the problem state attempts to call an entry not authorized by the PSW-key mask. The PSW-key mask is not examined in the supervisor state, all keys and entry points being valid.

Secondary-Space Control

Bit 5 of control register 0 is the secondary-space-control bit. This bit provides a mechanism whereby the control program can indicate whether or not the secondary segment table has been established. Bit 5 must be one to allow completion of these instructions:

MOVE to PRIMARY
MOVE TO SECONDARY
SET ADDRESS SPACE CONTROL

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

Subsystem-Linkage Control

Bit 0 of control register 5 is the subsystem-linkage-control bit. Bit 0 must be one to allow completion of these instructions:

PROGRAM CALL
PROGRAM TRANSFER

Otherwise, a special-operation exception is recognized. The subsystem-linkage-control bit is examined in both the problem and supervisor states and controls both the space-switching and current-primary versions of the instructions.

ASN-Translation Control

Bit 12 of control register 14 is the ASN-translation-control bit. This bit provides a mechanism whereby the control program can indicate whether ASN translation may occur while a particular program is being executed. Bit 12 must be one to allow completion of these instructions:

LOAD ADDRESS SPACE PARAMETERS
SET SECONDARY ASN
PROGRAM CALL with space switching
PROGRAM TRANSFER with space switching

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

Authorization Index

The authorization index is contained in bits 0-15 of control register 4. The authorization index is associated with the primary address space and is loaded along with the PASN when PROGRAM CALL with space switching, PROGRAM TRANSFER with space switching, or LOAD ADDRESS SPACE PARAMETERS is executed. The authorization index is used to determine whether a program is authorized to establish a particular address space. A program may be authorized to establish the address space as a secondary-address space, as a primary-address space, or both. The authorization index is examined in both the problem and supervisor states.

Associated with each address space is an authority table. The authorization index is used to select an entry in the authority table. Each entry contains two bits, which indicate whether the program with that authorization index is permitted to establish the address space as a primary address space, as a secondary address space, or both.

The instruction SET SECONDARY ASN with space switching uses the authorization index to test the secondary-authority bit in the authority-table entry to determine if the address space can be established as a secondary address space. The tested bit must be one; otherwise, a secondary-authority exception is recognized.

The instruction PROGRAM TRANSFER with space switching uses the authorization index to test the primary authority bit in the authority-table entry to determine if the address space can be established as a primary address space. The tested bit must be one; otherwise, a primary-authority exception is recognized.

The instruction PROGRAM CALL with space switching causes a new authorization index to be loaded from the ASN-second-table entry. This permits the program which is called to be given an authorization index which authorizes it to access more address spaces than those authorized for the calling program.

	Mode Requirement		Authorization Mechanism								Space-Switch-Event-Control Bit (CR1.31)
			Subsystem-Linkage Control (CR5.0)	Secondary-Space Control (CR0.5)	ASN-Translation-Control (CR14.12)		Extraction-Authority Control (CR0.4)	PSW-Key Mask (CR3.0-15)		Authorization Index (CR4.0-15)	
Instr	Priv Op	Trans Mode			Uncd	Cond		Bit Test	AND AKM[1]		
EPAR		SO-PS					Q				
ESAR		SO-PS					Q				
IAC		SO-PS					Q				
IPK							Q				
IVSK		SO-PS					Q				
LASP	P				SO					CC	CC
MVCP		SO-PS		SO				Q			
MVCS		SO-PS		SO				Q			
MVCK								Q			
PC-cp		SO-P	SO						Q		
PC-ss		SO-P	SO			SO			Q		X
PT-cp	Q[2]	SO-P	SO								
PT-ss	Q[2]	SO-P	SO			SO				PA	X
SAC		SO-PS		SO							
SPKA								Q			
SSAR-cp		SO-PS			SO						
SSAR-ss		SO-PS			SO					SA	

Explanation:

[1] The PSW-key mask is ANDed with the authorization key mask in the entry-table entry.

[2] The exception is recognized on an attempt to set the supervisor state when in the problem state.

CC Space-switch-event-control bit and authorization index tests cause a condition code to be set.

CRx.y Control register x, bit position y.

P Privileged-operation exception for privileged instruction.

PA Authority checked in both the problem and supervisor states; violation causes a primary-authority exception.

Q Privileged-operation exception for semiprivileged instruction. Authority checked only in the problem state.

SA Authority checked in both the problem and supervisor states; violation causes a secondary-authority exception.

SO Authority checked in both the problem and supervisor states; violation causes a special-operation exception.

SO-P CPU must be in the primary-space mode; if the CPU is in the secondary-space mode or in the real mode, a special-operation exception is recognized in both the problem and supervisor states.

SO-PS CPU must be in the primary-space mode or the secondary-space mode; if the CPU is in the real mode, a special-operation exception is recognized in both the problem and supervisor states.

X When bit 31 of control register 1 is one, a space-switch event is recognized. The operation is completed. The event is recognized in both the problem and supervisor states.

Summary of DAS Authorization Mechanisms

PC-NUMBER TRANSLATION

PC-number translation is the process of translating the 20-bit PC number to locate an entry-table entry as part of the execution of the PROGRAM CALL instruction. To perform this translation, the 20-bit PC number is divided into two fields. Bits 12-23 are the linkage index (LX), and bits 24-31 are the entry index (EX). The effective address, from which the PC-number is taken, has the following format:

////////////	LX	EX
0	12	24 31

The translation is performed by means of two tables: a linkage table and an entry table. Both of these tables reside in real storage. The linkage-table designation resides in control register 5. The entry table is designated by means of a linkage-table entry.

PC-NUMBER TRANSLATION CONTROL

PC-number translation is controlled by means of the linkage-table designation in control register 5. The register has the following format:

V		Linkage-Table Origin	LTL
0	1	8	25 31

Subsystem-Linkage Control (V): Bit 0 of control register 5 is the subsystem-linkage-control bit. Bit 0 must be one to allow completion of these instructions:

PROGRAM CALL
PROGRAM TRANSFER

Otherwise, a special-operation exception is recognized. The system-linkage-control bit is examined in both the problem and the supervisor states and controls both the space-switching and current-primary versions of the instructions.

Linkage-Table Origin: Bits 8-24 of control register 5, with seven zeros appended on the right, form a 24-bit real address that designates the beginning of the linkage table. With extended real addressing, the linkage-table origin is still a 24-bit real address and is extended on the left with zeros.

Linkage-Table Length (LTL): Bits 25-31 of control register 5 specify the length of the linkage table in units of 128 bytes, thus making the length of the linkage table variable in multiples of 32 four-byte entries. The length of the linkage table, in units of 128 bytes, is one more than the value in bit positions 25-31. The linkage-table length is compared against the leftmost seven bits of the linkage-index portion of the PC number to determine whether the linkage index designates an entry within the linkage table.

PC-NUMBER TRANSLATION TABLES

The PC-number translation process consists in a two-level lookup using two tables: a linkage table and an entry table. These tables reside in real storage.

Linkage-Table Entries

The entry fetched from the linkage table has the following format:

I	0000000	Entry-Table Origin	ETL
0	1	8	26 31

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

LX Invalid Bit (I): Bit 0 controls whether the entry table associated with the linkage-table entry is available.

When the bit is zero, PC-number translation proceeds by using the linkage-table entry. When the bit is one, an LX-translation exception is recognized.

Entry-Table Origin: Bits 8-25, with six zeros appended on the right, form a 24-bit real address that designates the beginning of the entry table. With extended real addressing, the entry-table origin is still a 24-bit real address and is extended on the left with zeros.

Entry-Table Length (ETL): Bits 26-31 specify the length of the entry table in units of 64 bytes, thus making the entry table variable in multiples of four 16-byte entries. The length of the entry table, in units of 64 bytes, is one more than the value in bit positions 26-31. The entry-table length is compared against the leftmost six bits of the entry index to determine whether the entry index designates an entry within the entry table.

Bits 1-7 of the linkage-table entry must be zeros; otherwise, a PC-translation-specification exception is recognized.

Entry-Table Entries

The entry fetched from the entry table is 16 bytes in length and has the following format:

```
┌─────────────────┬─────────────────┐
│  Auth Key Mask  │       ASN       │
└─────────────────┴─────────────────┘
0                 16              31

┌────────┬──────────────────────┬─┐
│00000000│   Entry Instr Addr   │P│
└────────┴──────────────────────┴─┘
32       40                      63

┌─────────────────────────────────┐
│         Entry Parameter         │
└─────────────────────────────────┘
64                              95

┌─────────────────┬─────────────────┐
│ Entry Key Mask  │/////////////////│
└─────────────────┴─────────────────┘
96                112            127
```

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

Authorization Key Mask: Bits 0-15 are used to verify whether the program issuing the PROGRAM CALL instruction, when in the problem state, is authorized to call this entry point. The authorization key mask and the current PSW-key mask in control register 3 are ANDed, and the result is checked for all zeros. If the result is all zeros, a privileged-operation exception is recognized. The test is not performed in the supervisor state.

ASN: Bits 16-31 specify whether a PC-ss or PC-cp is to occur. When bits 16-31 are zeros, a PC-cp is specified. When bits 16-31 are not all zeros, a PC-ss is specified, and the bits contain the ASN that replaces the primary ASN.

Entry Instruction Address: Bits 40-62, with a zero appended on the right, form the instruction address which replaces the instruction address in the PSW as part of the PROGRAM CALL operation.

Entry Problem State (P): Bit 63 replaces the problem-state bit, bit 15 of the current PSW, as part of the PROGRAM CALL operation.

Entry Parameter: Bits 64-95 are placed in general register 4.

Entry Key Mask: Bits 96-111 are ORed into the PSW-key mask in control register 3 as part of the PROGRAM CALL operation.

Bits 32-39 of the entry-table entry must be zeros; otherwise, a PC-translation-specification exception is recognized.

Programming Note

The entry parameter is intended to provide the called program with an address which can be depended upon and used as the basis of addressability in locating necessary information which may be environment-dependent. The parameter may be appropriately changed for each environment by setting up different entry tables. The alternative -- obtaining this information from the calling program -- may require extensive validity checking or may present an integrity exposure.

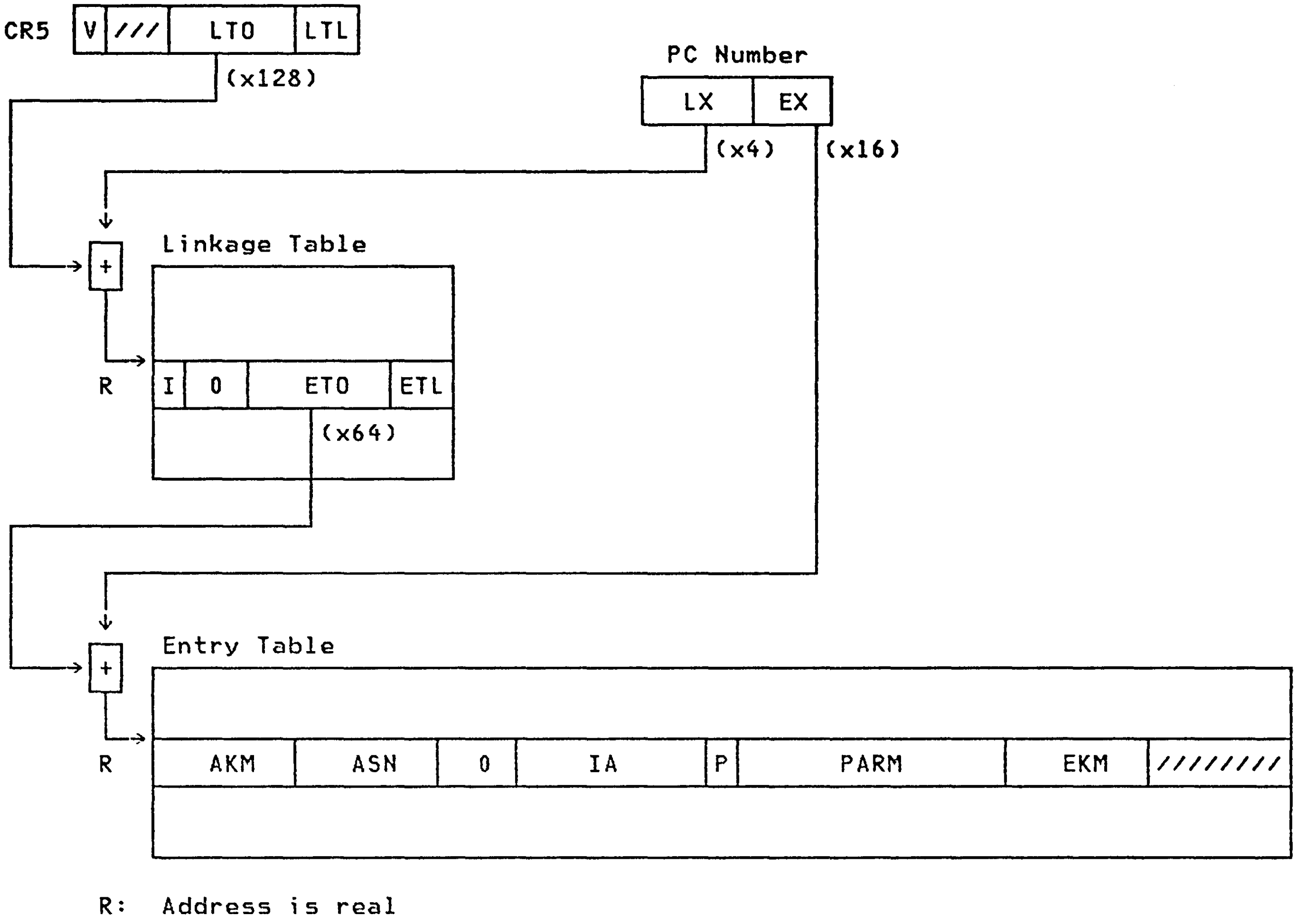
PC-NUMBER-TRANSLATION PROCESS

The translation of the PC number is performed by means of a linkage table and entry table both of which reside in real storage.

For the purposes of PC-number translation, the 20-bit PC number is divided into two parts: the leftmost 12 bits are called the linkage index (LX), and the rightmost eight bits are called the entry index (EX). The LX is used to select an entry from the linkage table, the starting address and length of which are specified by the contents of the linkage-table designation in control register 5. This entry designates the entry table to be used. The EX field of the PC number is then used to select an entry from the entry table.

When, for the purposes of PC-number translation, accesses are made to main storage to fetch entries from the linkage table and entry table, key-controlled protection does not apply.

The PC-number-translation process is shown in the figure "PC-Number Translation."



PC-Number Translation

<u>Linkage-Table</u> <u>Lookup</u>

The linkage-index (LX) portion of the PC number, in conjunction with the linkage-table origin, is used to select an entry from the linkage table.

The 24-bit real address of the linkage-table entry is obtained by appending seven zeros on the right to the contents of bit positions 8-24 of control register 5 and adding the linkage index, with two rightmost and 10 leftmost zeros appended. A carry, if any, into bit position 7 is ignored. With extended real addressing, this 24-bit real address is extended on the left with zeros; thus, the linkage table can wrap from 2^{24} - 1 to zero.

As part of the linkage-table-lookup process, the leftmost seven bits of the linkage index are compared against the linkage-table length, bits 25-31 of control register 5, to establish whether the addressed entry is within the linkage table. If the value in the linkage-table-length field is less than the value in the seven leftmost bits of the linkage index, an LX-translation exception is recognized.

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

Bit 0 of the linkage-table entry specifies whether the entry table corresponding to the linkage index is available. This bit is inspected, and, if it is one, an LX-translation exception is recognized.

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

Entry-Table Lookup

The entry-index (EX) portion of the PC number, in conjunction with the entry-table origin contained in the linkage-table entry, is used to select an entry from the entry table.

The 24-bit real address of the entry-table entry is obtained by appending six zeros on the right to the entry-table origin and adding the entry index, with four rightmost and 12 leftmost zeros appended. A carry, if any, into bit position 7 is ignored. With extended real addressing, this 24-bit real address is extended on the left with zeros. Thus, the entry table can wrap from 2^{24} - 1 to zero.

As part of the entry-table-lookup process, the six leftmost bits of the entry index are compared against the entry-table length, bits 26-31 of the linkage-table entry, to establish whether the addressed entry is within the table. If the value in the entry-table length field is less than the value in the six leftmost bits of the entry index, an EX-translation exception is recognized.

The 16-byte entry-table entry is fetched by using the real address. The entry appears to be fetched word-concurrent as observed by other CPUs, with the leftmost word fetched first. The order in which the remaining three words are fetched is unpredictable. The fetch access is not subject to protection. When the storage address which is generated for fetching the entry-table entry designates a location which is not available in the configuration, an addressing exception is recognized, and the operation is suppressed.

The use that is made of the information fetched from the entry-table entry is described in the definition of the PROGRAM CALL instruction.

Recognition of Exceptions during PC-Number Translation

The exceptions which can be encountered during the PC-number-translation process and their priority are described in the definition of the PROGRAM CALL instruction.

SEQUENCE OF STORAGE REFERENCES

The following sections describe the effects of overlapped operation and of piecemeal execution of a CPU program as that execution is observed in storage. Except for the section "Interlocks for Virtual-Storage References," the effects described in these sections are observable only when two or more CPUs or channels are in simultaneous execution and access common storage locations. Thus, in most cases, the program must take into account the effects which are described in these sections only for those cases in which the program interacts with another CPU or a channel.

Conceptual Sequence

Conceptually, the CPU processes instructions one at a time, with the execution of one instruction preceding the execution of the following instruction. The execution of the instruction designated by a successful branch follows the execution of the branch. Similarly, an interruption takes place between instructions or, for interruptible instructions, between units of operation of such instructions.

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

Overlapped Operation of Instruction Execution

Each operation of instruction execution appears to the program itself to be performed sequentially, with the current instruction being fetched after the preceding operation is completed and before the execution of the current operation is begun. This appearance is maintained even though the storage-implementation characteristics and overlap of instruction execution with storage accessing may cause actual processing to be different. The results generated are those that would have been obtained had the operations been performed in the conceptual sequence. Thus, it is possible for an instruction to modify the next succeeding instruction in storage. However, in certain situations involving dynamic address translation, where different virtual addresses map to the same real address, the copies of prefetched instructions are not necessarily changed. Also, when a vector-facility instruction is executed that causes storing into a location from which subsequent instructions have been prefetched, the copies of the prefetched instructions are not necessarily changed.

In simple models in which operations are not overlapped, the conceptual and actual sequences are essentially the same. However, in more complex machines, overlapped operation, buffering of operands and results, and execution times which are comparable to the propagation delays between units can cause the actual sequence to differ considerably from the conceptual sequence. In these machines, special circuitry is employed to detect dependencies between operations and ensure that the results obtained, as observed by the CPU which generates them, are those that would have been obtained if the operations had been performed in the conceptual sequence. However, other CPUs and channels may, unless otherwise constrained, observe a sequence that differs from the conceptual sequence.

Divisible Instruction Execution

It can normally be assumed that the execution of each instruction occurs as an indivisible event. However, in actual operation, the execution of an instruction consists in a series of discrete steps. Depending on the instruction, operands may be fetched and stored in a piecemeal fashion, and some delay may occur between fetching operands and storing results. As a consequence, intermediate or partially completed results may be observable by other CPUs and by channels.

When a program interacts with the operation on another CPU or a channel, the program may have to take into consideration that a single operation may consist in a series of storage references, that a storage reference may in turn consist in a series of accesses, and that the conceptual and observed sequences of these accesses may differ.

Storage references associated with instruction execution are of the following types: instruction fetches, DAT-table fetches, and storage-operand references. For the purpose of describing the sequence of storage references, accesses to storage in order to perform ASN translation, PC-number translation, and tracing are considered to be storage-operand references.

Programming Note

The sequence of execution of a CPU may differ from the simple conceptual definition in the following ways:

- As observed by the CPU itself, instructions may appear to be prefetched when different effective addresses are used. (See the section "Interlocks for Virtual-Storage References" in this chapter.)

- As observed by other CPUs and by channels, the execution of an instruction may appear to be performed as a sequence of piecemeal steps. This is described for each type of storage reference in the following sections.

- As observed by other CPUs and by channels, the storage-operand accesses associated with one instruction are not necessarily performed in the conceptual sequence. (See the section "Relation between Operand Accesses" in this chapter.)

- As observed by channels, in certain unusual situations, the contents of storage may appear to change and then be restored to the original value. (See the section "Storage Change and Restoration for DAT-Associated Access Exceptions" in this chapter.)

INTERLOCKS FOR VIRTUAL-STORAGE REFERENCES

As described in the previous section, CPU operation appears to be performed sequentially as observed by the CPU itself; the results stored by one instruction appear to be completed before the next instruction is fetched. This appearance is maintained in overlapped machines by means of special circuitry to detect accesses to a common location by comparing effective addresses.

For purposes of this definition, the term "effective address" is used to denote the address before translation, if any, regardless of whether the address is virtual, real, or absolute. If two effective addresses have the same value, the effective addresses are said to be the same even though one may be real or in a different address space.

When all accesses to a main-storage location are made by using the same effective address, then the above rule appears to be strictly maintained, as observed by the CPU itself. When different effective addresses are used to access the common location, the above rule does not hold in two cases:

1. For some instructions, the definition specifies the results which must be obtained for overlapping operands. This definition is specified in terms of the sequence of the storage accesses; that is, the results of some or all of the stores of one operand must be placed in storage before some parts or all parts of the other operand are fetched. When the store and

the fetch are performed by means of different effective addresses, then the operand may appear to be fetched before the store.

2. When an instruction changes the contents of a main-storage location from which a conceptually subsequent instruction is to be executed, either directly or by means of EXECUTE, and when different effective addresses are used to designate that location for storing the result and fetching the instruction, the instruction may appear to be fetched before the store occurs. This does not occur if an intervening operation causes the prefetched instructions to be discarded. A definition of when prefetched instructions must be discarded is included in the section "Instruction Fetching" in this chapter.

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

Programming Note

A single main-storage location can be accessed by more than one address in several ways:

1. The DAT tables may be set up such that multiple addresses in a single address space, or virtual addresses in different address spaces, map to a single real address.

2. The translation of logical, instruction, and virtual addresses may be changed by loading the DAT parameters in the control registers, by changing the address-space-control bit in the PSW, or, for logical and instruction addresses, by turning DAT on or off.

3. Certain instructions use real addresses, and the instructions MOVE TO PRIMARY and MOVE TO SECONDARY access two address spaces.

4. Accesses to storage for the purpose of storing and fetching information for interruptions is performed by means of real addresses, and, for the store-status function, by means of absolute addresses, whereas accesses by the program may be by means of virtual addresses.

5. The real-to-absolute mapping may be changed by means of the SET PREFIX instruction or a reset.

6. A main-storage location may be accessed by channels by means of an absolute address and by the CPU by means of a real or a virtual address.

7. A main-storage location may be accessed by another CPU by means of one type of address and by this CPU by means of a different type of address.

8. The CPU updates the interval timer by means of a real address, and the program may access the location by means of a virtual address.

The primary purpose of this section is to describe the effects caused in case 1 above.

For case 2, no effect is observable because prefetched instructions are discarded when the translation parameters are changed and the delay of stores by a CPU is not observable by the CPU itself.

For case 3, for those instructions which fetch by using real addresses (for example, LOAD REAL ADDRESS), no effect is observable because only operand accesses between instructions are involved. All instructions that store by using a real address or that store into another address space cause prefetched instructions to be discarded, and no effect is observable.

Cases 4 and 5 are situations which are defined to cause serialization, with the result that prefetched instructions are discarded. In these cases, no effect is observable.

The handling of cases 6 and 7 involves accesses as observed by other CPUs and by channels and is covered in the following sections in this chapter.

For case 8, the effect of updating the interval timer is observable only if an instruction is fetched from real location 80 or 82 by using a virtual address which is not 80 or 82, respectively.

INSTRUCTION FETCHING

Instruction fetching consists in fetching the one, two, or three halfwords designated by the instruction address in the current PSW. The immediate field of an instruction is accessed as part of an instruction fetch. If, however, an instruction designates a storage operand at the location occupied by the instruction itself, the location is accessed both as an instruction and as a storage operand. The fetch of the target instruction of EXECUTE is considered to be an instruction fetch.

The bytes of an instruction may be fetched piecemeal and are not necessarily accessed in a left-to-right direction. The instruction may be fetched multiple times for a single execution; for example, it may be fetched for testing the addressability of operands or for inspection of PER events, and it may be refetched for actual execution.

Instructions are not necessarily fetched in the sequence in which they are conceptually executed and are not necessarily fetched each time they are executed. In particular, the fetching of an instruction may precede the storage-operand references for an instruction that is conceptually earlier. The instruction fetch occurs prior to all storage-operand references for all instructions that are conceptually later.

An instruction may be prefetched by using a virtual address only when the associated DAT table entries are attached and valid or when entries which qualify for substitution for the table entries exist in the TLB. An instruction that has been prefetched may be interpreted for execution only for the same virtual address for which the instruction was prefetched.

No limit is established on the number of instructions which may be prefetched, and multiple copies of the contents of a single storage location may be fetched. As a result, the instruction executed is not necessarily the most recently fetched copy. Storing caused by other CPUs and by channels does not necessarily change the copy of prefetched instructions. However, if a store that is conceptually earlier is made by the same CPU using the same effective address as that by which the instruction is subsequently fetched, the updated information is obtained.

All copies of prefetched instructions are discarded when:

- A serializing function is performed.

- The CPU enters the operating state.

- The CPU changes from DAT on to DAT off or from DAT off to DAT on.

- A change is made to a translation parameter in control register 0 or 1 when DAT is on.

- DAS is installed and the CPU changes from one to the other of the primary-space mode and secondary-space mode.

- DAS is installed, and a change is made to a translation parameter in control register 7 when DAT is on.

Programming Notes

1. As observed by a CPU itself, its own instruction prefetching is not normally apparent; the only exception occurs when multiple virtual addresses in a single address space, or virtual addresses in different address spaces, map to a single real address. This is described in the section "Interlocks for Virtual-Storage References" in this chapter.

2. The following are some effects of instruction prefetching on one CPU as observed by other CPUs and by channels.

   It is possible for one CPU to prefetch the contents of a storage location, after which another CPU or a channel can change the storage location and then set a flag to indicate that the change has been made. Subsequently, the first CPU can test and find the flag set, branch to the modified location, and execute the original prefetched contents.

   It is possible, if another CPU or a channel concurrently modifies the instruction, for one CPU to recognize the changes to some but not all bit positions of an instruction.

   It is possible for one CPU to prefetch an instruction and subsequently, before the instruction is executed, for another CPU to change the storage key. As a result, the first CPU may appear to execute instructions from a protected storage location. However, the copy of the instructions executed is the copy prefetched before the location was protected.

DAT-TABLE FETCHES

The fetching of dynamic-address-translation (DAT) table entries may occur as follows:

1. A DAT-table entry may be prefetched into the translation-lookaside

buffer (TLB) and used from the TLB without refetching from storage, until the entry is cleared by an INVALIDATE PAGE TABLE ENTRY, PURGE TLB, or SET PREFIX instruction or by CPU reset. DAT-table entries are not necessarily fetched in the sequence conceptually called for; they may be fetched at any time they are attached and valid, including during the execution of conceptually previous instructions.

2. All bytes of a DAT-table entry appear to be fetched concurrently, as observed by other CPUs. However, the reference to the entry may appear to access a single byte at a time, as observed by channels.

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-table entry.

4. A DAT-table entry may be fetched for each use of the address, including any trial execution, and for each reference to each byte of each operand.

5. The DAT page-table-entry fetch precedes the reference to the page. When no copy of the page-table entry is in the TLB, the fetch of the associated segment-table entry precedes the fetch of the page-table entry.

STORAGE-KEY ACCESSES

References to the storage key are handled as follows:

1. Whenever a reference to storage is made and key-controlled protection applies to the reference, the four access-control bits and the fetch-protection bit associated with the storage location are inspected concurrently with the reference to the storage location.

2. When storing is performed, the change bit is set in the associated storage key concurrently with the store operation.

3. The instructions SET STORAGE KEY and SET STORAGE KEY EXTENDED cause all seven bits to be set concurrently in the storage key. The access to the storage key for SET STORAGE KEY and SET STORAGE KEY EXTENDED follows the sequence rules for storage-operand store references and is a single-access reference. For SET STORAGE KEY EXTENDED, the two keys in a double-key 4K-byte block are not necessarily accessed concurrently.

4. The instructions INSERT STORAGE KEY and INSERT STORAGE KEY EXTENDED provide a consistent image of bits 0-6 of the storage key. Similarly, the instructions INSERT VIRTUAL STORAGE KEY and TEST PROTECTION provide a consistent image of bits 0-4 of the storage key. The access to the storage key for all of these instructions follows the sequence rules for storage-operand fetch references and is a single-access reference. For INSERT STORAGE KEY EXTENDED, the two keys in a double-key 4K-byte block are not necessarily accessed concurrently.

5. The instructions RESET REFERENCE BIT and RESET REFERENCE BIT EXTENDED modify only the reference bit. All other bits of the storage key remain unchanged. The reference bit and change bit are examined concurrently to set the condition code. The access to the storage key for RESET REFERENCE BIT and RESET REFERENCE BIT EXTENDED follows the sequence rules for storage-operand update references. The reference bit is the only bit which is updated. For RESET REFERENCE BIT EXTENDED, the two keys in a double-key 4K-byte block are not necessarily accessed concurrently.

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.

The change bit may be set in cases when no storing has occurred. See the section "Exceptions to Nullification and Suppression" in this chapter.

STORAGE-OPERAND REFERENCES

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

During the execution of an instruction, all or some of the storage operands for that instruction may be fetched, intermediate results may be maintained for subsequent modification, and final results may be temporarily held prior to placing them in storage. Stores caused by other CPUs and by channels do not necessarily affect these intermediate results.

Storage-operand references are of three types: fetches, stores, and updates.

Storage-Operand Fetch References

When the bytes of a storage operand participate in the instruction execution only as a source, the operand is called a fetch-type operand, and the reference to the location is called a storage-operand fetch reference. A fetch-type operand is identified in individual instruction definitions by indicating that the access exception is for fetch.

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

The storage-operand fetch references of one instruction occur after those of all preceding instructions and before those of subsequent instructions, as observed by other CPUs and by channels. The operands of any one instruction are fetched in the sequence specified for that instruction.

Storage-Operand Store References

When the bytes of a storage operand participate in the instruction execution only as a destination, to the extent of being replaced by the result, the operand is called a store-type operand, and the reference to the location is called a storage-operand store reference. A store-type operand is identified in individual instruction definitions by indicating that the access exception is for store.

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

The CPU may delay placing results in storage. There is no defined limit on the length of time that results may remain pending before they are stored.

This delay does not affect the sequence in which results are placed in storage. The results of one instruction are placed in storage after the results of all preceding instructions have been placed in storage and before any results of the succeeding instructions are stored, as observed by other CPUs and by channels. The results of any one instruction are stored in the sequence specified for that instruction.

The CPU does not fetch operands or DAT-table entries from a storage location until all information destined for that location by the CPU has been stored. Prefetched instructions may appear to be updated before the information appears in storage.

The stores are necessarily completed only as a result of a serializing operation and before the CPU enters the stopped state.

Storage-Operand Update References

In some instructions, the storage-operand location participates both as a source and as a destination. In these cases, the reference to the location consists first in a fetch and subsequently in a store. The operand is called an update-type operand, and the combination of the two accesses is referred to as an update reference. Instructions such as MOVE ZONES, TRANSLATE, OR (OC, OI), and ADD DECIMAL cause an update to the first-operand location. An update-type operand is identified in the individual instruction definition by indicating that the access exception is for both fetch and store.

For most instructions which have update-type operands, the fetch and store accesses associated with an update reference do not necessarily occur one immediately after the other, and it is possible for other CPUs and channels to make fetch and store accesses to the same location during this time. Such an update reference is sometimes called a noninterlocked-update storage reference.

For certain special instructions, the update reference is interlocked against certain accesses by other CPUs. Such an update reference is called an interlocked-update reference. The fetch and store accesses associated with an interlocked-update reference do not necessarily occur one immediately after the other, but all store accesses and the fetch and store accesses associated with interlocked-update references by other CPUs are prevented from occurring at the same location between the fetch and the store accesses of an interlocked-update reference. Accesses by channels may occur to the location during the interlock period.

The storage-operand update references for the following instructions appear to be an interlocked-update reference as observed by other CPUs. The instructions TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP perform an interlocked-update reference. On models in which the STORE CHARACTERS UNDER MASK instruction with a mask of zero fetches and stores the byte designated by the second-operand address, the fetch and store accesses are an interlocked-update reference. For DAS tracing, the current-entry-control word in the trace-table-entry header is changed by means of an interlocked-update reference.

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

Programming Notes

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

2. The instructions TEST AND SET, COMPARE AND SWAP, and COMPARE DOUBLE AND SWAP facilitate updating of a common storage field by two or more CPUs. To ensure that no changes are lost, all CPUs must use an instruction providing an interlocked-update reference. In addition, the program must ensure that channels do not store into the same storage location since such stores may occur between the fetch and store portions of an interlocked-update reference.

3. Only those bytes which are included in the result field of both operations are considered to be part of the common main-storage location. However, all bits within a common byte are considered to be common even if the bits modified by the two operations do not overlap. As an example, if (1) one CPU executes the instruction OR (OC) with a length of 1 and the value 80 hex in the second-operand location and (2) the other CPU executes AND (NC) with a length of 1 and the value FE hex in the second-operand location, and (3) the first operand of both instructions is the same byte, then the result of one of the updates can be lost.

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

   Depending on the model, an access to store information may be performed, for example, in the following cases:

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

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

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

STORAGE-OPERAND CONSISTENCY

Single-Access References

A fetch reference is said to be a single-access reference if the value is fetched in a single access to each byte of the data field. In the case of overlapping operands, the location may be accessed once for each operand. A store-type reference is said to be a single-access reference if a single store access occurs to each byte location within the data field. An update reference is said to be single-access if both the fetch and store accesses are each single-access.

Except for the accesses associated with multiple-access references and the stores associated with storage change and restoration for DAT-associated access exceptions, all storage-operand references are single-access references.

Multiple-Access References

In some cases, multiple accesses may be made to all or some of the bytes of a storage operand. The following cases may involve multiple-access references:

1. The storage operands of the following instructions: CONVERT TO BINARY, CONVERT TO DECIMAL, MOVE INVERSE, MOVE WITH OFFSET, PACK, TRANSLATE, TEST BLOCK, and UNPACK.

2. The stores into that portion of the first operand of MOVE LONG which is filled with padding bytes.

3. The storage operands of the decimal instructions.

4. The stores into a DAS-trace entry.

5. The storage operands of vector-facility instructions.

6. The stores associated with the stop-and-store-status SIGNAL PROCESSOR order.

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

Programming Notes

1. When multiple fetch or store accesses are made to a single byte that is being changed by another CPU or by a channel, the result is not necessarily limited to that which could be obtained by fetching or storing the bits individually. For example, the execution of MULTIPLY DECIMAL may consist in repetitive additions and subtractions, each of which causes the second operand to be fetched from storage and the first operand to be updated in storage.

2. When CPU instructions which make multiple-access references are used to modify storage locations being simultaneously accessed by another CPU or by a channel, multiple store accesses to a single byte by the CPU may result in intermediate values being observed by the other CPU or by the channel. To avoid these intermediate values (for example, when modifying a CCW chain), only instructions making single-access references should be used.

Block-Concurrent References

For some references, the accesses to all bytes within a halfword, word, or doubleword are specified to appear to be block-concurrent as observed by other CPUs. These accesses do not necessarily appear to channels to include more than a byte at a time. The halfword, word, or doubleword is referred to in this section as a block. When a fetch-type reference is specified to appear to be concurrent within a block, no store access to the block by another CPU is permitted during the time that bytes contained in the block are being fetched. Accesses to the bytes within the block by channels may occur between the fetches. When a store-type reference is specified to appear to be concurrent within a block, no access to the block, either fetch or store, is permitted by another CPU during the time that the bytes within the block are being stored. Accesses to the bytes in the block by channels may occur between the stores.

Consistency Specification

For all instructions in the S format and RX format, with the exception of EXECUTE, CONVERT TO DECIMAL, and CONVERT TO BINARY, when the operand is addressed on a boundary which is integral to the size of the operand, the storage-operand references appear to be block-concurrent as observed by other CPUs.

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

The instructions LOAD MULTIPLE and STORE MULTIPLE, when the operand starts on a word boundary, and the instructions COMPARE LOGICAL (CLC), COMPARE LOGICAL CHARACTERS UNDER MASK, INSERT CHARACTERS UNDER MASK, and STORE CHARACTERS UNDER MASK access their storage operands in a left-to-right direction, and all bytes accessed within each doubleword appear to be accessed concurrently as observed by other CPUs.

The instructions LOAD CONTROL and STORE CONTROL access the storage operand in a left-to-right direction, and all bytes accessed within each word appear to be accessed concurrently as observed by other CPUs.

When destructive overlap does not exist, the operands of MOVE (MVC), MOVE WITH

KEY, MOVE TO PRIMARY, and MOVE TO SECONDARY are accessed as follows:

1. The first operand is accessed in a left-to-right direction, and all bytes accessed within a doubleword appear to be accessed concurrently as observed by other CPUs.

2. The second operand is accessed 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 be fetched concurrently as observed by other CPUs. Thus, if the first and second operands begin on the same byte offset within a doubleword, the second operand appears to be fetched doubleword-concurrent. If the offsets within a doubleword differ by 4, the second operand appears to be fetched word-concurrent as observed by other CPUs.

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

The operands for MOVE LONG appear to be accessed doubleword-concurrent as observed by other CPUs when all of the following are true:

- Both operands start on doubleword boundaries and are an integral number of doublewords in length.

- The operands do not overlap.

- The nonpadding part of the operation is being executed.

The operands for COMPARE LOGICAL LONG appear to be accessed doubleword-concurrent as observed by other CPUs when both operands start on doubleword boundaries and are an integral number of doublewords in length.

For EXCLUSIVE OR (XC), the operands are processed in a left-to-right direction, and, when the first and second operands coincide, all bytes accessed within a doubleword appear to be accessed concurrently as observed by other CPUs.

Programming Note

In the case of EXCLUSIVE OR (XC) designating operands which coincide exactly, the bytes within the field may appear to be accessed as many as three times, by two fetches and one store: once as the fetch portion of the first operand update, once as the second-operand fetch, and then once as the store portion of the first-operand update. Each of the three accesses appears to be doubleword-concurrent as observed by other CPUs, but the three accesses do not necessarily appear to occur one immediately after the other. One or both fetch accesses may be omitted since the instruction can be completed without fetching the operands.

RELATION BETWEEN OPERAND ACCESSES

As observed by other CPUs and by channels, storage-operand fetches associated with one instruction execution appear to precede all storage-operand references for conceptually subsequent instructions. A storage-operand store specified by one instruction appears to precede all storage-operand stores specified by conceptually subsequent instructions, but it does not necessarily precede storage-operand fetches specified by conceptually subsequent instructions. However, a storage-operand store appears to precede a conceptually subsequent storage-operand fetch from the same main-storage location.

When an instruction has two storage operands both of which cause fetch references, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. When the two operands overlap, the common locations may be fetched independently for each operand.

When an instruction has two storage operands the first of which causes a store and the second a fetch reference, it is unpredictable how much of the second operand is fetched before the results are stored. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

When an instruction has two storage operands the first of which causes an update reference and the second a fetch reference, it is unpredictable which operand is fetched first, or how much of one operand is fetched before the other operand is fetched. Similarly, it is unpredictable how much of the result is processed before it is returned to storage. In the case of destructively overlapping operands, the portion of the second operand which is common to the first is not necessarily fetched from storage.

Programming Note

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

When the same storage location is designated by two operand addresses of an instruction, and another CPU or a channel causes the contents of the location to change during execution of the instruction, the old and new values of the location may be used simultaneously. For example, comparison of a field to itself may yield a result other than equal, or EXCLUSIVE-ORing of a field with itself may yield a result other than zero.

OTHER STORAGE REFERENCES

The restart, program, supervisor-call, external, input/output, and machine-check PSWs appear to be accessed doubleword-concurrent as observed by other CPUs. These references appear to occur after the conceptually previous unit of operation and before the conceptually subsequent unit of operation. The relationship between the new-PSW fetch, the old-PSW store, and the interruption-code store is unpredictable.

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

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

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

SERIALIZATION

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

CPU SERIALIZATION

All interruptions and the execution of certain instructions cause a serialization of CPU operations. A serialization operation consists in completing all conceptually previous storage accesses by the CPU, as observed by other CPUs and by channels, before the conceptually subsequent storage accesses occur. Serialization affects the sequence of all CPU accesses to storage and to the storage keys, except for those associated with DAT-table-entry fetching.

| Serialization is performed by CPU reset, all interruptions, and by the execution of the following instructions:

- The general instructions BRANCH ON CONDITION (BCR) with the M_1 and R_2 field containing all ones and all zeros, respectively, and COMPARE AND SWAP, COMPARE DOUBLE AND SWAP, STORE CLOCK, SUPERVISOR CALL, and TEST AND SET.
- LOAD PSW, SET STORAGE KEY, and SET STORAGE KEY EXTENDED.
- All I/O instructions, CONNECT CHANNEL SET, and DISCONNECT CHANNEL SET.
- PURGE TLB and SET PREFIX, which also cause the translation-lookaside buffer to be cleared of entries.
- SIGNAL PROCESSOR, READ DIRECT, and WRITE DIRECT.
- INVALIDATE PAGE TABLE ENTRY.
- TEST BLOCK.
- MOVE TO PRIMARY, MOVE TO SECONDARY, PROGRAM CALL, PROGRAM TRANSFER, SET ADDRESS SPACE CONTROL, and SET SECONDARY ASN.
- The DAS-tracing function causes serialization to be performed before the trace action and after completion of the trace action.

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

1. All conceptually previous storage accesses by the CPU are completed

as observed by other CPUs and by channels. This includes all conceptually previous stores and changes to the storage keys.

2. The normal function associated with the serializing operation is performed. In the case of instruction execution, operands are fetched, and the storing of results is completed. The exceptions are LOAD PSW and SET PREFIX, in which the operand may be fetched before previous stores have been completed, and interruptions, in which the interruption code and associated fields may be stored prior to the serialization. The fetching of the serializing instruction occurs before the execution of the instruction and may precede the execution of previous instructions, but may not precede the completion of any previous serializing operation. In the case of an interruption, the old PSW, the interruption code, and other information, if any, are stored, and the new PSW is fetched, but not necessarily in that sequence.

3. Finally, instruction fetch and operand accesses for conceptually subsequent operations may begin.

A serializing function affects the sequence of storage accesses that are under the control of the CPU in which the serializing function takes place. It does not affect the sequence of storage accesses under the control of other CPUs and of channels.

Programming Notes

1. The following are some effects of a serializing operation:

   a. When the execution of an instruction changes the contents of a storage location that is used as a source of a following instruction and when different addresses are used to designate the same absolute location for storing the result and fetching the instruction, a serializing operation following the change ensures that the modified instruction is executed.

   b. When a serializing operation takes place, other CPUs and channels observe instruction and operand fetching and result storing to take place in the sequence established by the serializing operation.

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

CHANNEL-PROGRAM SERIALIZATION

Serialization of a channel program occurs as follows:

1. All storage accesses and storage-key accesses by the channel program follow initiation of the execution of START I/O or START I/O FAST RELEASE, or, if suspended, RESUME I/O, as observed by CPUs and by other channels. This includes all accesses for the CAW, CCWs, IDAWs, and data.

2. All storage accesses and storage-key accesses by the channel program are completed, as observed by CPUs and by other channels, before the CSW is stored indicating termination of the operation at the subchannel.

3. If a CCW contains a PCI flag or a suspend flag which is one, all storage accesses and storage-key accesses due to CCWs preceding it in the CCW chain are completed, as observed by CPUs and by other channels, before the CSW is stored indicating the PCI or suspended condition.

The serialization of a channel program does not affect the sequence of storage accesses or storage-key accesses caused by other channel programs or by another CPU program.

CHAPTER 6. INTERRUPTIONS

The interruption mechanism permits the CPU to change its state as a result of conditions external to the configuration, within the configuration, or within the CPU itself. To permit fast response to conditions of high priority and immediate recognition of the type of condition, interruption conditions are grouped into six classes: external, input/output, machine check, program, restart, and supervisor call.

INTERRUPTION ACTION

An interruption consists in storing the current PSW as an old PSW, storing information identifying the cause of the interruption, and fetching a new PSW. Processing resumes as specified by the new PSW.

The old PSW stored on an interruption normally contains the address of the instruction that would have been executed next had the interruption not occurred, thus permitting resumption of the interrupted program. For program and supervisor-call interruptions, the information stored also contains a code that identifies the length of the last-executed instruction, thus permitting the program to respond to the cause of the interruption. In the case of some program conditions for which the normal response is reexecution of the instruction causing the interruption, the instruction address directly identifies the instruction last executed.

Except for restart, an interruption can occur only when the CPU is in the operating state. The restart interruption can occur with the CPU in either the stopped or operating state.

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

Source Identification	Interruption Code		PSW-Mask Bits		Mask Bits in Ctrl Registers	ILC Set	Execution of Instruction Identified by Old PSW
			EC	BC	Reg, Bit		
MACHINE CHECK (old PSW 48, new PSW 112)	Locations 232-239[1]						
Exigent condition			13	13		u	terminated or nullified[2]
Repressible cond			13	13	14, 4-7	u	unaffected[2]
SUPERVISOR CALL (old PSW 32, new PSW 96)	Locations 138-139 in the EC mode and 34-35 in the BC mode						
Instruction bits	00000000 ssssssss					1,2	completed
PROGRAM (old PSW 40, new PSW 104)	Locations 142-143 in the EC mode and 42-43 in the BC mode						
	Binary	Hex[3]					
Operation	00000000 p0000001	0001				1,2,3	suppressed
Privileged oper	00000000 p0000010	0002				1,2,3	suppressed
Execute	00000000 p0000011	0003				2	suppressed
Protection	00000000 p0000100	0004				0,1,2,3	suppressed or terminated
Addressing	00000000 p0000101	0005				0,1,2,3	suppressed or terminated
Specification	00000000 p0000110	0006				0,1,2,3	suppressed or completed
Data	00000000 p0000111	0007				2,3	suppressed or terminated
Fixed-pt overflow	xxxxxxxx p0001000	0008	20	36		1,2	completed
Fixed-point divide	00000000 p0001001	0009				1,2	suppressed or completed
Decimal overflow	00000000 p0001010	000A	21	37		2,3	completed
Decimal divide	00000000 p0001011	000B				2,3	suppressed
Exponent overflow	xxxxxxxx p0001100	000C				1,2	completed
Exponent underflow	xxxxxxxx p0001101	000D	22	38		1,2	completed
Significance	xxxxxxxx p0001110	000E	23	39		1,2	completed
Floating-pt divide	xxxxxxxx p0001111	000F				1,2	suppressed or inhibited[4]
Segment transl	00000000 p0010000	0010				1,2,3	nullified
Page translation	00000000 p0010001	0011				1,2,3	nullified
Translation spec	00000000 p0010010	0012				1,2,3	suppressed
Special operation	00000000 p0010011	0013			0, 1	2	suppressed
ASN-transl spec	00000000 p0010111	0017				2	suppressed
Vector operation[4]	00000000 p0011001	0019				2,3	nullified
Space-switch event	00000000 p0011100	001C			1, 31	2	completed
Unnormalized operand[4]	xxxxxxxx p0011110	001E				2	inhibited[4]
PC-transl spec	00000000 p0011111	001F				2	suppressed
AFX translation	00000000 p0100000	0020				2	nullified
ASX translation	00000000 p0100001	0021				2	nullified
LX translation	00000000 p0100010	0022				2	nullified
EX translation	00000000 p0100011	0023				2	nullified
Primary authority	00000000 p0100100	0024				2	nullified
Secondary auth	00000000 p0100101	0025				2	nullified
Monitor event	00000000 p1000000	0040			8, 16-31	2	completed
PER event	xxxxxxxx 1nnnnnnn[5]	0080	1	*	9, 0-3@	0,1,2,3	completed[6]

Interruption Action (Part 1 of 3)

Source Identification	Interruption Code		PSW-Mask Bits		Mask Bits in Ctrl Registers	ILC Set	Execution of Instruction Identified by Old PSW
			EC	BC	Reg, Bit		
EXTERNAL (old PSW 24, new PSW 88)	Locations 134-135 in the EC mode and 26-27 in the BC mode						
	Binary	Hex[3]					
Interval timer	00000000 1eeeeeee	0080	7	7	0, 24	u	unaffected
Interrupt key	00000000 e1eeeeee	0040	7	7	0, 25	u	unaffected
External signal 2	00000000 ee1eeeee	0020	7	7	0, 26	u	unaffected
External signal 3	00000000 eee1eeee	0010	7	7	0, 26	u	unaffected
External signal 4	00000000 eeee1eee	0008	7	7	0, 26	u	unaffected
External signal 5	00000000 eeeee1ee	0004	7	7	0, 26	u	unaffected
External signal 6	00000000 eeeeee1e	0002	7	7	0, 26	u	unaffected
External signal 7	00000000 eeeeeee1	0001	7	7	0, 26	u	unaffected
Malfunction alert	00010010 00000000	1200	7	7	0, 16	u	unaffected
Emergency signal	00010010 00000001	1201	7	7	0, 17	u	unaffected
External call	00010010 00000010	1202	7	7	0, 18	u	unaffected
TOD-clock sync chk	00010000 00000011	1003	7	7	0, 19	u	unaffected
Clock comparator	00010000 00000100	1004	7	7	0, 20	u	unaffected
CPU timer	00010000 00000101	1005	7	7	0, 21	u	unaffected
Service signal	00100100 00000001	2401	7	7	0, 22	u	unaffected
INPUT/OUTPUT (old PSW 56, new PSW 120)	Locations 186-187 in the EC mode and 58-59 in the BC mode						
Channel 0	00000000 dddddddd		6	0	2, 0[7]	u	unaffected
Channel 1	00000001 dddddddd		6	1	2, 1[7]	u	unaffected
Channel 2	00000010 dddddddd		6	2	2, 2[7]	u	unaffected
Channel 3	00000011 dddddddd		6	3	2, 3[7]	u	unaffected
Channel 4	00000100 dddddddd		6	4	2, 4[7]	u	unaffected
Channel 5	00000101 dddddddd		6	5	2, 5[7]	u	unaffected
Channel 6 & up	cccccccc dddddddd		6	6	2, 6+	u	unaffected
RESTART (old PSW 8, new PSW 0)	Locations 2-3 in the BC mode						
Restart key	00000000 00000000[8]					u	unaffected

Interruption Action (Part 2 of 3)

Explanation:

Locations for the old PSWs, new PSWs, and interruption codes are real locations.

[1] A model-independent machine-check interruption code of 64 bits is stored at real locations 232-239. In the BC mode, the contents of real locations 50-51 are unpredictable.

[2] The effect of the machine-check condition is indicated by bits in the machine-check-interruption code. The setting of these bits indicates the extent of the damage and whether the unit of operation is nullified, terminated, or unaffected.

[3] The interruption code in the column labeled "Hex" is the hex code for the basic interruption; this code does not show the effects of concurrent interruption conditions represented by e, n, p, or x in the column labeled "Binary."

[4] Vector-operation and unnormalized-operand exceptions are associated with the vector facility. "Inhibited" is a type of ending which occurs only for instructions associated with the vector facility. These are described in the publication IBM System/370 Vector Operations, SA22-7125.

[5] When the interruption code indicates a PER event, an ILC of 0 may be stored only when bits 8-15 of the interruption code are 10000110 (PER, specification).

[6] The unit of operation is completed, unless a program exception concurrently indicated causes the unit of operation to be inhibited, nullified, suppressed, or terminated.

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

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

+ Plus the following bits in the control register. One mask bit is provided for each installed channel; the bit position matches the channel address.

* In the BC mode, PER is disabled.

@ Additional masks in control register 9, bit positions 16-31, provide detailed control over the source of PER general-register-alteration events which are masked by control register 9, bit 3.

c Channel-address bits.

d Device-address bits.

e If one, the bit indicates another concurrent external-interruption condition.

n A possible nonzero code, indicating another concurrent program-interruption condition.

p If one, the bit indicates a concurrent PER-event interruption condition.

s Bits of the I field of SUPERVISOR CALL.

u Unpredictable in the BC mode; not stored in the EC mode.

x Exception-extension code. This field is described in the publication IBM System/370 Vector Operations, SA22-7125. This field is set to zero except by vector instructions.

Interruption Action (Part 3 of 3)

INTERRUPTION CODE

The six classes of interruptions (external, I/O, machine check, program, restart, and supervisor call) are distinguished by the storage locations at which the old PSW is stored and from which the new PSW is fetched. For most classes, the causes are further identified by an interruption code and, for some classes, by additional information placed in permanently assigned real storage locations during the interruption. (See also the section "Assigned Storage Locations" in Chapter 3, "Storage.") For external, I/O, program, and supervisor-call interruptions, the interruption code consists of 16 bits. In the BC mode, the interruption code is zero in the PSW stored by the store-status function and is unpredictable when the PSW is displayed.

For external interruptions in the EC mode, the interruption code is stored at real locations 134-135. In the BC mode, the interruption code is placed in the old PSW. A parameter may be stored at real locations 128-131, or a CPU address may be stored at real locations 132-133.

For I/O interruptions in the EC mode, the interruption code, which contains the I/O address, is stored at real locations 186-187. In the BC mode, the interruption code is placed in the old PSW. Additional information is provided by the contents of the channel-status word (CSW) stored at real location 64. Further information may be provided by the limited channel logout stored at real locations 176-179 and by a full channel logout stored in the fixed-logout area (real locations 256-351) or in the I/O-extended-logout area.

For machine-check interruptions, the interruption code consists of 64 bits and is stored at real locations 232-239. Additional information for identifying the cause of the interruption and for recovering the state of the machine may be provided by the contents of the machine-check failing-storage address, the external-damage code, the region code, and the contents of the fixed-logout, extended-logout, and machine-check-save areas. (See Chapter 11, "Machine-Check Handling.")

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

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

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

ENABLING AND DISABLING

By means of mask bits in the current PSW and in control registers, the CPU may be enabled or disabled for all external, I/O, and machine-check interruptions and for some program interruptions. When a mask bit is one, the CPU is enabled for the corresponding class of interruptions, and these interruptions can occur.

When a mask bit is zero, the CPU is disabled for the corresponding interruptions. The conditions that cause I/O interruptions remain pending. External-interruption conditions either remain pending or persist until the cause is removed. Machine-check-interruption conditions, depending on the type, are ignored, remain pending, or cause the CPU to enter the check-stop state. The disallowed program-interruption conditions are ignored, except that some causes are indicated also by the setting of the condition code. The setting of the significance and exponent-underflow program-mask bits affects the manner in which floating-point operations are completed when the corresponding condition occurs.

The CPU is always enabled for program interruptions for which mask bits are not provided, as well as the supervisor-call and restart interruptions.

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

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

Programming Notes

1. Mask bits in the PSW provide a means of disallowing all maskable interruptions; thus, subsequent interruptions can be disallowed by the new PSW introduced by an interruption. Furthermore, the mask bits can be used to establish a hierarchy of interruption priorities, where a condition in one class can interrupt the program handling a condition in another class but not vice versa. To prevent an interruption-handling routine from being interrupted before the necessary housekeeping steps are performed, the new PSW must disable the CPU for further interruptions within the same class or within a class of lower priority.

2. Because the mask bits in control registers are not changed as part of the interruption procedure, these masks cannot be used to prevent an interruption immediately after a previous interruption in the same class. The mask bits in control registers provide a means for selectively enabling the CPU for some sources and disabling it for others within the same class.

HANDLING OF FLOATING INTERRUPTION CONDITIONS

An interruption condition which can be presented to any CPU in the configuration is called a floating interruption condition. The condition is presented to the first CPU in the configuration which is enabled for the corresponding interruption and which can accept the interruption, and then the condition is cleared and not presented to any other CPU in the configuration. A CPU cannot accept the interruption when it is in the check-stop state, has an invalid prefix, is in a string of program interruptions due to a specification exception of the type which is recognized early, is executing a READ DIRECT instruction, or is in the stopped state. However, a CPU with the rate control set to instruction step can accept the interruption when the start key is activated.

Service signal and certain machine-check conditions are floating interruption conditions.

INSTRUCTION-LENGTH CODE

The instruction-length code (ILC) occupies two bit positions and provides the length of the last instruction executed. It permits identifying the instruction causing the interruption when the instruction address in the old PSW designates the next sequential instruction. The ILC is provided also by the BRANCH AND LINK instructions.

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

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

For supervisor-call and program interruptions, a nonzero ILC identifies in halfwords the length of the instruction that was last executed. Whenever an instruction is executed by means of EXECUTE, instruction-length code 2 is set to indicate the length of EXECUTE and not that of the target instruction.

The value of a nonzero instruction-length code is related to the leftmost two bits of the instruction. The value does not depend on whether the operation code is assigned or on whether the instruction is installed. The following table summarizes the meaning of the instruction-length code:

ILC		Instr Bits	Instruction
Decimal	Binary	0-1	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 instruction address stored in the old PSW does not identify the instruction causing the interruption.

An ILC of 0 occurs when a specification exception due to a PSW-format error is recognized as part of early exception recognition and the PSW has been introduced by LOAD PSW or an interruption. (See the section "Exceptions Associated with the PSW" later in this chapter.) In the case of LOAD PSW, the instruction address of LOAD PSW or EXECUTE has been replaced by the instruction address of the new PSW. When the invalid PSW is introduced by an interruption, the PSW-format error cannot be attributed to an instruction.

On some models without the translation facility, an ILC of 0 occurs also when an addressing exception or a protection exception is recognized for 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 instruction address of the instruction causing the exception nor the length of the last-executed instruction is made available to the program. This type of interruption is sometimes referred to as an imprecise program interruption.

In the case of LOAD PSW and the supervisor-call interruption, a PER event may be indicated concurrently with a specification exception having an ILC of 0.

ILC on Instruction-Fetching Exceptions

When a program interruption occurs because of an exception that prohibits access to the instruction, the instruction-length code cannot be set on the basis of the first two bits of the instruction. As far as the significance of the ILC for this case is concerned, the following two situations are distinguished:

1. When an odd instruction address causes a specification exception to be recognized or when an addressing, protection, or translation-specification exception is encountered on fetching an instruction, the ILC is set to 1, 2, or 3, indicating the multiple of 2 by which the instruction address has been incremented. It is unpredictable whether the instruction address is incremented by 2, 4, or 6. By reducing the instruction address in the old PSW by the number of halfword locations indicated in the ILC, the instruction address originally appearing in the PSW may be obtained.

2. When a segment-translation or page-translation exception is recognized while fetching an instruction, including the target instruction of EXECUTE, the ILC is arbitrarily set to 1, 2, or 3. In this case, the operation is nullified, and the instruction address is not incremented.

The ILC is not necessarily related to the first two bits of the instruction when the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword. The ILC may be arbitrarily set to 1, 2, or 3 in these cases. The instruction address is or is not updated, as described in situations 1 and 2 above.

When any exceptions other than segment translation or page translation are encountered on fetching the target instruction of EXECUTE, the ILC is 2.

Programming Notes

1. A nonzero instruction-length code for a program interruption indicates the number of halfword locations by which the instruction address in the program old PSW must be reduced to obtain the instruction address of the last instruction executed, unless one of the following situations exists:

   a. The interruption is caused by an exception resulting in nullification.

   b. An interruption for a PER event occurs before the execution of an interruptible instruction is completed, and no other program-interruption condition is indicated concurrently.

   c. The interruption is caused by a PER event due to LOAD PSW or a branch or linkage instruction, including SUPERVISOR CALL (but not including MONITOR CALL).

   d. The interruption is caused by an access exception encountered in fetching an instruction, and the instruction address has been introduced into the PSW by a means other than sequential operation (by a branch instruction, LOAD PSW, an interruption, or conclusion of an IPL sequence).

   e. The interruption is caused by a specification exception because of an odd instruction address.

   f. The interruption is caused by an early specification exception or by an access exception encountered in fetching an instruction, and changes have been made to a parameter that controls the relation between instruction addresses and real addresses. The relation between instruction addresses and real addresses can be changed without introducing an entire new PSW by switching from the real mode, primary-space mode, or secondary-space mode to a different mode, or by changing one or more of the translation parameters in control registers 0, 1, and 7. The early specification exception can be caused by executing STORE THEN OR SYSTEM MASK or SET SYSTEM MASK, which switches to or from the real mode while introducing invalid values in bit positions 0-7 of an EC-mode PSW.

   For situations a and b above, the instruction address in the PSW is not incremented, and the instruction designated by the instruction address is the same as the last one executed. These situations are the only ones in which the instruction address in the old PSW identifies the instruction causing the exception.

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

For situation f, the instruction address in the PSW has not been replaced, but the corresponding real address after the change may be different.

2. The instruction-length code (ILC) is redundant when a PER event is indicated since the PER address in the word at real location 152 identifies the instruction causing the interruption (or the EXECUTE instruction, as appropriate). Similarly, the ILC is redundant when the operation is nullified, since in this case the instruction address in the PSW is not incremented. If the ILC value is required in this case, it can be derived from the operation code of the instruction identified by the old PSW.

EXCEPTIONS ASSOCIATED WITH THE PSW

Exceptions associated with erroneous information in the current PSW may be recognized when the information is introduced into the PSW or may be recognized as part of the execution of the next instruction. Errors in the PSW which are specification-exception conditions are called PSW-format errors.

Early Exception Recognition

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

- The EC mode is specified (PSW bit 12 is one) in a CPU that does not have the translation facility installed.

- Bit position 16 of an EC-mode PSW is one, and DAS is not installed.

- A one is introduced into an unassigned bit position of an EC-mode PSW (that is, any of bit positions 0, 2-4, 17, or 24-39).

The interruption occurs regardless of whether the wait state is specified. If the invalid PSW causes the CPU to become enabled for a pending I/O, external, or machine-check interruption, the program interruption occurs instead, and the pending interruption is subject to the mask bits of the new PSW introduced by the program interruption. If the EC mode is not present, bits 0-15 and 34-63 of the invalid PSW are stored unchanged in the corresponding bit positions of the program old PSW, and the interruption code and instruction-length code are stored in 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 error conditions, the instruction-length code is set to 0, and the newly introduced PSW, except for the interruption code and the instruction-length code in the BC mode, is stored unmodified as the old PSW. When one of the above error conditions is introduced by execution of SET SYSTEM MASK or STORE THEN OR SYSTEM MASK, the instruction-length code is set to 2, and the instruction address is incremented by 4. The PSW containing the invalid value introduced into the system-mask field is stored as the old PSW.

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

Late Exception Recognition

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

- A specification exception is recognized due to an odd instruction address in the PSW (PSW bit 63 is one).

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

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

If an I/O, external, or machine-check-interruption condition is pending and the PSW causes the CPU to be enabled for that condition, the corresponding interruption occurs, and the PSW is not inspected for exceptions which are recognized late. Similarly, a PSW specifying the wait state is not inspected for exceptions which are recognized late.

Programming Notes

1. The execution of LOAD ADDRESS SPACE PARAMETERS, LOAD PSW, PROGRAM CALL, PROGRAM TRANSFER, SET PREFIX, SET SECONDARY ASN, SET SYSTEM MASK, STORE THEN AND SYSTEM MASK, and STORE THEN OR SYSTEM MASK is suppressed on an addressing or protection exception, and hence the program old PSW provides information concerning the program causing the exception.

2. When the first halfword of an instruction can be fetched but an access exception is recognized on fetching the second or third halfword, the ILC is not necessarily related to the operation code.

3. If the new PSW introduced by an interruption contains a PSW-format error, a string of interruptions may occur. (See the section "Priority of Interruptions" in this chapter.)

EXTERNAL INTERRUPTION

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

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

The source of the interruption is identified in the interruption code. When the old PSW specifies the EC mode, the interruption code is stored at real locations 134-135. 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.

Additionally, for the malfunction-alert, emergency-signal, and external-call conditions, a 16-bit CPU address is associated with the source of the interruption and is stored at real locations 132-133 in both the EC and BC modes. When the CPU address is stored, bit 6 of the interruption code is set to one. For all other conditions, no CPU address is stored, and bit 6 of the interruption code is set to zero. When bit 6 is zero and the old PSW specifies the EC mode, zeros are stored at real locations 132-133. When bit 6 is zero and the old PSW specifies the BC mode, the contents of real locations 132-133 remain unchanged.

For the service-signal interruption, a 32-bit parameter is associated with the interruption and is stored at real locations 128-131 in both the EC and BC modes. Bit 2 of the external-interruption code indicates that a parameter has been stored. When bit 2 is zero, the contents of real locations 128-131 remain unchanged.

External-interruption conditions are of two types: those for which an interruption-request condition is held pending, and those for which the condition directly requests the interruption. Clock comparator, CPU timer, and TOD-clock sync check are conditions which directly request external interruptions. If a condition which directly requests an external interruption is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and if the condition persists, more than one interruption may result from a single occurrence of the condition.

When several interruption requests for a single source are generated before the interruption occurs, and the interruption condition is of the type which is held pending, only one request for that source is preserved and remains pending.

An external interruption for a particular source can occur only when the CPU is enabled for interruption by that source. The external interruption occurs at the completion of a unit of operation. The external mask, PSW bit 7, and external subclass-mask bits in control register 0 control whether the CPU is enabled for a particular source. Each source for an external interruption has a subclass-mask bit assigned to it, and the source can cause an interruption only when the external-mask bit is one and the corresponding subclass-mask bit is one. The use of the subclass-mask bits does not depend on whether the CPU is in the EC or BC mode.

When the CPU becomes enabled for a pending external-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

More than one source may present a request for an external interruption at the same time. When the CPU becomes enabled for more than one concurrently pending request, the interruption occurs for the pending condition or conditions having the highest priority.

The priorities for external-interruption requests in descending order are as follows:

Interval timer, interrupt key,
  external signals 2-7
Malfunction alert

Emergency signal
External call
TOD-clock sync check
Clock comparator
CPU timer
Service signal

The interval timer, interrupt key, and the external signals 2-7 are of equal priority; if more than one of these conditions is pending and allowed, the conditions are indicated concurrently. All other requests are honored one at a time. When more than one emergency-signal request exists at a time or when more than one malfunction-alert request exists at a time, the request associated with the smallest CPU address is honored first.

CLOCK COMPARATOR

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

1. The TOD clock is in the set or not-set state, and the value of the clock comparator is less than the value in the compared portion of the TOD clock, both compare values being considered unsigned binary integers.

2. The clock comparator is installed, and the TOD clock is in the error or not-operational state.

If the condition responsible for the request is removed before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from a single occurrence of the condition.

When the TOD clock accessed by a CPU is set or changes state, interruption conditions, if any, that are due to the clock comparator may or may not be recognized for up to 1.048576 seconds after the change.

The subclass-mask bit is in bit position 20 of control register 0. This bit is initialized to zero.

The clock-comparator condition is indicated by an external-interruption code of 1004 hex.

CPU TIMER

An interruption request for the CPU timer exists whenever the CPU-timer value is negative (bit 0 of the CPU timer is one). If the value is made positive before the request is honored, the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may occur from a single occurrence of the condition.

When the TOD clock accessed by a CPU is set or changes state, interruption conditions, if any, that are due to the CPU timer may or may not be recognized for up to 1.048576 seconds after the change.

The subclass-mask bit is in bit position 21 of control register 0. This bit is initialized to zero.

The CPU-timer condition is indicated by an external-interruption code of 1005 hex.

EMERGENCY SIGNAL

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

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

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

The emergency-signal condition is indicated by an external-interruption code of 1201 hex. The address of the CPU that executed the SIGNAL PROCESSOR instruction is stored at real locations 132-133.

EXTERNAL CALL

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

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

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

The external-call condition is indicated by an external-interruption code of 1202 hex. The address of the CPU that executed the SIGNAL PROCESSOR instruction is stored at real locations 132-133.

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.

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

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

External signals 2-7 are indicated by setting to one interruption-code bits 10-15, respectively. Bits 0-7 are set to zeros, and bits 8 and 9 are set to zeros unless set to ones for other conditions that are concurrently indicated.

Programming Notes

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

2. The pattern presented in bit positions 10-15 of the interruption code depends on the pattern received before the interruption occurs. 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 interruption code for the external interruption resulting from the earliest signals. These late signals, if not included in the interruption code, cause another interruption to occur.

INTERRUPT KEY

An interruption request for the interrupt key is generated when the operator activates that key. The request is preserved and remains pending in the CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

When the interrupt key is activated while the CPU is in the load state, it depends on the model whether an interruption request is generated or the condition is lost.

The subclass-mask bit is in bit position 25 of control register 0. This bit is initialized to one.

The interrupt-key condition is indicated by setting bit 9 in the interruption code to one and by setting bits 0-7 to zeros. Bits 8 and 10-15 are zeros unless set to ones for other conditions that are concurrently indicated.

INTERVAL TIMER

An interruption request for the interval timer is generated when the interval timer is decremented from a positive number or zero to a negative number. The request is preserved and remains pending in the CPU until it is cleared. The pending request is cleared when it causes an interruption and by CPU reset.

When the TOD clock accessed by a CPU is set or changes state, interruption conditions, if any, that are due to the interval timer may or may not be recognized for up to 1.048576 seconds after the change.

The subclass-mask bit is in bit position 24 of control register 0. This bit is initialized to one.

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

MALFUNCTION ALERT

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

Facilities are provided for holding a separate malfunction-alert request pending in the receiving CPU for each of the other CPUs in the configuration. Removal of a CPU from the configuration does not generate a malfunction-alert condition.

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

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

SERVICE SIGNAL

An interruption request for a service signal is generated upon the completion of certain configuration-control and maintenance functions, such as those initiated by means of the model-dependent DIAGNOSE instruction. A 32-bit parameter is provided with the interruption to assist the program in determining the operation for which the interruption is reported.

Service signal is a floating interruption condition and is presented to the first CPU in the configuration which can accept the interruption. The pending request is cleared when it causes an interruption in any one of the CPUs and also by subsystem reset.

The subclass-mask bit is in bit position 22 of control register 0. This bit is initialized to zero.

The service-signal condition is indicated by an external-interruption code of 2401 hex. A 32-bit parameter is stored at real locations 128-131.

TOD-CLOCK SYNC CHECK

The TOD-clock-sync-check condition indicates that more than one TOD clock exists in the configuration, and that the rightmost 32 bits of the clocks are not running in synchronism.

An interruption request for a TOD-clock sync check exists when the TOD clock accessed by this CPU is running (that is, the clock is in the set or not-set state), the clock accessed by any other CPU in the configuration is running, and bits 32-63 of the two clocks do not match. When a clock is set or changes state, or when 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 TOD 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; if the condition is recognized, it may continue to persist up to 1.048576 seconds after both clocks have been running with the rightmost 32 bits matching. However, in this case, the condition does not persist if one of the TOD clocks is removed from the configuration.

When more than one CPU shares a TOD clock, only the CPU with the smallest CPU address among those sharing the clock indicates a TOD-clock-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 subclass-mask bit is in bit position 19 of control register 0. This bit is initialized to zero.

The TOD-clock-sync-check condition is indicated by an external-interruption code of 1003 hex.

I/O INTERRUPTION

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

A request for an I/O interruption may occur at any time, and more than one request may occur at the same time. The requests are preserved and remain pending in channels or devices until accepted by the CPU, or until cleared by some other means, such as subsystem reset.

The I/O interruption occurs at the completion of a unit of operation. Priority is established among requests so that only one interruption request is processed at a time. For more details, see the section "Input/Output Interruptions" in Chapter 13, "Input/Output Operations."

When the CPU becomes enabled for I/O interruptions and a channel has established priority for a pending I/O-interruption condition, the interruption occurs at the completion of the instruction execution or interruption that causes the enabling.

An I/O interruption causes the old PSW to be stored at real location 56, a channel-status word to be stored at real location 64, and a new PSW to be fetched from real location 120. Upon detection of equipment errors, additional information may be stored in the form of a limited channel logout at real locations 176-179, and in the form of a full channel logout at real locations 256-351 or in the I/O-extended-logout area starting at the absolute location designated by the contents of real locations 173-175.

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

A nonzero value for the measurement byte is part of the start-I/O-fast-queuing facility. When this facility is not installed, zeros are stored at this location.

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

The channel-mask bits in control register 2 start at bit position 0 and extend for at least as many contiguous bit positions as required to control interruptions from the channel with the greatest installed channel address which may be connected to this CPU. 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. Installed channel-mask bits are initialized to one; the state of the remaining bits in control register 2 is unpredictable.

When the current PSW specifies the EC mode, each channel is controlled by the I/O-mask bit, PSW bit 6, and by 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. The channel causing the interruption must be a member of a channel set which is connected to this CPU.

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

MACHINE-CHECK INTERRUPTION

The machine-check interruption is a means for reporting to the program the occurrence of equipment malfunctions. Information is provided to assist the program in determining the source of the fault and extent of the damage.

A machine-check interruption causes the old PSW to be stored at real location 48 and a new PSW to be fetched from real location 112. When the old PSW specifies the BC mode, the contents of the interruption-code and ILC fields in the old PSW are unpredictable.

The cause and severity of the malfunction are identified by a 64-bit machine-check-interruption code stored at real locations 232-239. Further information identifying the cause of the interruption and the location of the fault may be stored at real locations 216-511 and in the area starting with the real location designated by the contents of control register 15.

The interruption action and the storing of the associated information are under the control of PSW bit 13 and bits in control register 14. See Chapter 11, "Machine-Check Handling," for more detailed information.

PROGRAM INTERRUPTION

Program interruptions are used to report exceptions and events which occur during execution of the program.

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

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

Except for PER events, the condition causing the interruption is indicated by a coded value placed in the rightmost seven bit positions of the interruption code. Only one condition at a time can be indicated. Bits 0-7 of the interruption code are set to zeros.

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

When there is a corresponding mask bit, a program interruption can occur only when that mask bit is one. The program mask in the PSW controls four of the exceptions, 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, a hierarchy of masks control interruptions due to PER events. When any controlling mask bit is zero, the condition is ignored; the condition does not remain pending.

Programming Notes

1. When the new PSW for a program interruption has a PSW-format error or causes an exception to be recognized in the process of instruction fetching, a string of program interruptions may occur. See the section "Priority of Interruptions" in this chapter for a description of how such strings are terminated.

2. Some of the conditions indicated as program exceptions may be recognized also by a channel, in which case the exception is indicated in the channel-status word.

EXCEPTION-EXTENSION CODE

When an arithmetic exception is recognized during execution of an interruptible vector instruction, a nonzero exception-extension code is stored in bits 0-7 of the program-interruption code. This code is set to a nonzero value only for arithmetic exceptions occurring during the execution of vector instructions. For more details, see the publication IBM System/370 Vector Operations, SA22-7125.

PROGRAM-INTERRUPTION CONDITIONS

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

Addressing Exception

An addressing exception is recognized when the CPU attempts to reference a main-storage location that is not available in the configuration. A main-storage location is not available in the configuration when the location is not installed, when the storage unit is not in the configuration, or when power is off in the storage unit. An address designating a storage location that is not available in the configuration is referred to as invalid.

The operation is suppressed when the address of the instruction is invalid. Similarly, the operation is suppressed when the address of the target instruction of EXECUTE is invalid. Also, the unit of operation is suppressed when an addressing exception is encountered in accessing a table entry. The table entries to which the rule applies are entries for the segment table, page table, linkage table, entry table, ASN first table, ASN second table, authority table, trace-table designation, trace-table-entry header, and CPU-identity byte. Addressing exceptions result in suppression when they are encountered for references to the segment table and page table, in both implicit references for dynamic address translation and references associated with the execution of LOAD REAL ADDRESS and TEST PROTECTION. Except for some specific instructions whose execution is suppressed, the operation is terminated for an operand address that can be translated but designates an unavailable location. See the figure "Summary of Action for Addressing and Protection Exceptions."

For termination, changes may occur only to result fields. In this context, the term "result field" includes the condition code, registers, and any storage locations that are provided and that are designated to be changed by the instruction. Therefore, if an instruction is due to change only the contents of a field in storage, and every byte of the field is in a location that is not available in the configuration, the operation is suppressed. When part of an operand location is available in the configuration and part is not, storing may be performed in the part that is available in the configuration.

When an addressing exception occurs during the fetching of an instruction or during the fetching of a DAT table entry associated with an instruction fetch, it is unpredictable whether the ILC is 1, 2, or 3. When the exception is associated with fetching the target of EXECUTE, the ILC is 2.

In all cases of addressing exceptions not associated with instruction fetching, the ILC is 1, 2, or 3, indicating the length of the instruction that caused the reference. However, on some models without the translation facility, an ILC of 0 occurs when an addressing exception is recognized for a store-type reference.

An addressing exception is indicated by a program-interruption code of 0005 hex (or 0085 hex if a concurrent PER event is indicated).

	Action on		
Exception	Table-Entry Fetch[1]	Instruction Fetch	Operand Reference
Addressing exception	Suppress	Suppress	Suppress for IPTE, LASP, LPSW, SCKC, SPT, SPX, SSM, STNSM, STOSM, TPROT, and DAS tracing.[2] Terminate for all others.[3]
Protection exception for key-controlled protection	--	Suppress	Suppress for IPTE, LASP, LPSW, SCKC, SPT, SPX, SSM, STNSM, and STOSM. Terminate for all others.[3]
Protection exception for segment protection	--	--	Suppress for STNSM, STOSM, and DAS tracing.[2] Terminate for all others.[3]
Protection exception for low-address protection	--	--	Suppress for IPTE, STNSM, STOSM, and DAS tracing.[2] Terminate for all others.[3]

Explanation:

-- Not applicable.

[1] Table entries include segment table, page table, linkage table, entry table, ASN first table, ASN second table, authority table, trace-table designation, trace-table-entry header, and CPU-identity byte.

[2] The following instructions may cause an entry to be made in the trace table when DAS tracing is active: PC, PT, and SSAR. The stores into the current-entry-control word and the trace entry are subject to addressing, segment-protection, and low-address-protection exceptions. The operation is suppressed for these exceptions.

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

Summary of Action for Addressing and Protection Exceptions

AFX-Translation Exception

An AFX-translation exception is recognized when, during ASN translation in PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), or SET SECONDARY ASN with space switching (SSAR-ss), bit 0 of the ASN-first-table entry used is not zero.

The ASN being translated is stored at real locations 146-147, and real locations 144-145 are set to zeros.

The operation is nullified.

The instruction-length code is 2.

The AFX-translation exception is indicated by a program-interruption code of 0020 hex (or 00A0 hex if a concurrent PER event is indicated).

ASN-Translation-Specification Exception

An ASN-translation-specification exception is recognized during ASN translation in LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), or SET SECONDARY ASN with space switching (SSAR-ss) when either:

1. Bit positions 1-7 and 28-31 of a valid ASN-first-table entry do not contain zeros.

2. Bit positions 1-7, 30, 31, 60-63, and 97-103 of a valid ASN-second-table entry do not contain zeros.

The operation is suppressed.

The instruction-length code is 2 or 3.

The ASN-translation-specification exception is indicated by a program-interruption code of 0017 hex (or 0097 hex if a concurrent PER event is indicated).

ASX-Translation Exception

An ASX-translation exception is recognized when, during ASN translation in PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), or SET SECONDARY ASN with space switching (SSAR-ss), bit 0 of the ASN-second-table entry used is not zero.

The ASN being translated is stored at real locations 146-147, and real locations 144-145 are set to zeros.

The operation is nullified.

The instruction-length code is 2.

The ASX-translation exception is indicated by a program-interruption code of 0021 hex (or 00A1 hex if a concurrent PER event is indicated).

Data Exception

A data exception is recognized when any of the following is true:

1. The sign or digit codes of operands in the decimal instructions (described in Chapter 8, "Decimal Instructions") or in CONVERT TO BINARY are invalid.

2. The operand fields in ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, and SUBTRACT DECIMAL overlap in a way other than with coincident rightmost bytes; or operand fields in ZERO AND ADD overlap, and the rightmost byte of the second operand is to the right of the rightmost byte of the first operand.

3. The multiplicand in MULTIPLY DECIMAL has an insufficient number of leftmost zeros.

The action taken for a data exception depends on whether a sign code is invalid. The operation is suppressed when a sign code is invalid, regardless of whether any other condition causing the exception exists; when no sign code is invalid, the operation is terminated.

For all instructions other than EDIT and EDIT AND MARK, when the operation is terminated, the contents of the sign position in the rightmost byte of the result field either remain unchanged or are set to the preferred sign code; the contents of the remainder of the result field are unpredictable.

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

The instruction-length code is 2 or 3.

The data exception is indicated by a program-interruption code of 0007 hex (or 0087 hex if a concurrent PER event is indicated).

Programming Notes

1. The definition for data exception permits termination when digit codes are invalid but no sign code is invalid. On some models, valid digit codes may be placed in the result field even if the original contents were invalid. Thus it is possible, after a data exception occurs, for all fields to contain valid codes.

2. 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 is located in the numeric portion of the result field.

3. When, after a program interruption for data exception, a sign code is found to be invalid, the operation has been suppressed if both of the following conditions are met:

   a. The invalid sign of the source field is not located in the numeric portion of the result field.

   b. The invalid sign code is in a position specified by the instruction to be checked for a valid sign. (This condition excludes the first operand of ZERO AND ADD, both operands of EDIT, and EDIT AND MARK.)

Decimal-Divide Exception

A decimal-divide exception is recognized when in decimal division the divisor is zero or the quotient exceeds the specified data-field size.

The decimal-divide exception is indicated only if the sign codes of both the divisor and dividend are valid and only if the digit or digits used in establishing the exception are valid.

The operation is suppressed.

The instruction-length code is 2 or 3.

The decimal-divide exception is indicated by a program-interruption code of 000B hex (or 008B hex if a concurrent PER event is indicated).

Decimal-Overflow Exception

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

The interruption may be disallowed by the decimal-overflow mask (PSW bit 21 in the EC mode and PSW bit 37 in the BC mode).

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

The instruction-length code is 2 or 3.

The decimal-overflow exception is indicated by a program-interruption code of 000A hex (or 008A hex if a concurrent PER event is indicated).

Execute Exception

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

The operation is suppressed.

The instruction-length code is 2.

The execute exception is indicated by a program-interruption code of 0003 hex (or 0083 hex if a concurrent PER event is indicated).

Exponent-Overflow Exception

An exponent-overflow exception is recognized when the result characteristic of a floating-point operation exceeds 127 and the result fraction is not zero.

The operation is completed. The fraction is normalized, and the sign and fraction of the result remain correct. The result characteristic is made 128 smaller than the correct characteristic.

The instruction-length code is 1 or 2.

The exponent-overflow exception is indicated by a program-interruption code of XX0C hex (or XX8C hex if a concurrent PER event is indicated), where XX is the exception-extension code.

Exponent-Underflow Exception

An exponent-underflow exception is recognized when the result characteristic of a floating-point operation is less than zero and the result fraction is not zero. For an extended-format floating-point result, exponent underflow is indicated only when the high-order characteristic underflows.

The interruption may be disallowed by the exponent-underflow mask (PSW bit 22 in the EC mode and PSW bit 38 in the BC mode).

The operation is completed. The exponent-underflow mask also affects the result of the operation. When the mask bit is zero, the sign, characteristic, and fraction are set to zero, making the result a true zero. When the mask bit is one, the fraction is normalized, the characteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct.

The instruction-length code is 1 or 2.

The exponent-underflow exception is indicated by a program-interruption code of XX0D hex (or XX8D hex if a concurrent PER event is indicated), where XX is the exception-extension code.

EX-Translation Exception

An EX-translation exception is recognized during PC-number translation in PROGRAM CALL when the entry-table entry indicated by the entry-table-index part of the PC number is beyond the length of the entry table as designated by the linkage-table entry.

The PC number is stored in bit positions 12-31 of the word at real location 144, and the leftmost 12 bits of the word are set to zeros.

The operation is nullified.

The instruction-length code is 2.

The EX-translation exception is indicated by a program-interruption code of 0023 hex (or 00A3 hex if a concurrent PER event is indicated).

Fixed-Point-Divide Exception

A fixed-point-divide exception is recognized when in signed binary division the divisor is zero or when the quotient in signed binary division or the result of CONVERT TO BINARY cannot be expressed as a 32-bit signed binary integer.

In the case of division, the operation is suppressed. The execution of CONVERT TO BINARY is completed by ignoring the leftmost bits that cannot be placed in the register.

The instruction-length code is 1 or 2.

The fixed-point-divide exception is indicated by a program-interruption code of 0009 hex (or 0089 hex if a concurrent PER event is indicated).

Fixed-Point-Overflow Exception

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

The interruption may be disallowed by the fixed-point-overflow mask (PSW bit 20 in the EC mode and PSW bit 36 in the BC mode).

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

The instruction-length code is 1 or 2.

The fixed-point-overflow exception is indicated by a program-interruption code of XX08 hex (or XX88 hex if a concurrent PER event is indicated), where XX is the exception-extension code.

Floating-Point-Divide Exception

A floating-point-divide exception is recognized when in floating-point division the divisor has a zero fraction.

The operation is suppressed.

The instruction-length code is 1 or 2.

The floating-point-divide exception is indicated by a program-interruption code of XX0F hex (or XX8F hex if a concurrent PER event is indicated), where XX is the exception-extension code.

LX-Translation Exception

An LX-translation exception is recognized during PC-number translation in PROGRAM CALL when either:

1. The linkage-table entry indicated by the linkage-table-index part of the PC number is beyond the length of the linkage table as designated by control register 5.

2. Bit 0 of the linkage-table entry is not zero.

The PC number is stored in bit positions 12-31 of the word at real location 144, and the leftmost 12 bits of the word are set to zeros.

The operation is nullified.

The instruction-length code is 2.

The LX-translation exception is indicated by a program-interruption code of 0022 hex (or 00A2 hex if a concurrent PER event is indicated).

Monitor Event

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

Control Register 8

	Monitor Masks
	16 31

The monitor-mask bits, bits 16-31 of control register 8, correspond to monitor classes 0-15, respectively. Any number of monitor-mask bits may be on at a time; together they specify the classes of monitor events that are monitored at that time. The mask bits are initialized to zeros.

When MONITOR CALL is executed and the corresponding monitor-mask bit is one, a program interruption for monitor event occurs.

The monitor event can occur in both the EC and BC modes.

Additional information is stored at real locations 148-149 and 156-159. The format of the information stored at these locations is the same in the EC and BC modes and is as follows:

Real Locations 148-149

00000000	Monitor Class No.
0	8 15

Real Locations 156-159

00000000	Monitor Code
0	8 31

The contents of bit positions 8-15 of the MONITOR CALL instruction are stored at real location 149 and constitute the monitor-class number. Zeros are stored at real location 148. The effective address specified by the B_1 and D_1 fields of the instruction forms the monitor code, which is stored at real locations 157-159. Zeros are stored at real location 156.

The operation is completed.

The instruction-length code is 2.

The monitor event is indicated by a program-interruption code of 0040 hex (or 00C0 hex if a concurrent PER event is indicated).

Operation Exception

An operation exception is recognized when the CPU attempts to execute an instruction with an invalid operation code. The operation code may be unassigned, or the instruction with that operation code may not be installed on the CPU.

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

1. When the first eight bits of an instruction have the value B2, A4, A5, A6, E4, or E5 hex, or have the value 9C hex and the suspend-and-resume facility is installed, the first 16 bits form the operation code.

2. In all other cases, the first eight bits alone form the operation code.

The operation is suppressed.

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

The operation exception is indicated by a program-interruption code of 0001 hex (or 0081 hex if a concurrent PER event is indicated).

Programming Notes

1. Some models may offer instructions not described in this publication, such as those provided for assists or as part of special or custom features. Consequently, operation codes not described in this publication do not necessarily cause an operation exception to be recognized. Furthermore, these instructions may cause modes of operation to be set up or may otherwise alter the machine so as to affect the execution of subsequent instructions. To avoid causing such an operation, an instruction with an operation code not described in this publication

should be executed only when the specific function associated with the operation code is desired.

2. The operation code 00, with a two-byte instruction format, currently is not assigned. It is improbable that this operation code will ever be assigned.

3. In the case of I/O instructions with hex values 9D, 9E, 9F, and, on machines without the suspend-and-resume facility, 9C, 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. On machines with the suspend-and-resume facility, all 16 bits are checked for op codes beginning 9C hex.

   To ensure that presently written programs operate correctly if and when the I/O operation codes (9D, 9E, and 9F) are extended further to provide for new functions, only zeros should be placed in the unassigned bit positions in the second op-code byte. In accordance with these recommendations, the operation codes for the I/O instructions are shown as 9C00, 9C01, 9D00, etc.

Page-Translation Exception

A page-translation exception is recognized when either:

1. The page-table entry indicated by the page-index portion of a virtual address is outside the page table.

2. The page-invalid bit is one.

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

The segment-index and page-index portion of the virtual address causing the exception is stored at real locations 145-147. When DAS is installed, bit 0 of real location 144 is set to zero if the virtual address was relative to the primary address space, or it is set to one if the virtual address was relative to the secondary address space. When DAS is not installed, bit 0 of real location 144 is set to zero. Bits 1-7 of real location 144 are set to zeros. When 2K-byte pages are used, the rightmost 11 bits of the address stored are unpredictable; when 4K-byte pages are used, the rightmost 12 bits of the address stored are unpredictable.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during a reference to the target of EXECUTE, the ILC is 2.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The page-translation exception is indicated by a program-interruption code of 0011 hex (or 0091 hex if a concurrent PER event is indicated).

PC-Translation-Specification Exception

A PC-translation-specification exception is recognized during PC-number translation in PROGRAM CALL when bit positions 1-7 of a valid linkage-table entry do not contain zeros or when bit positions 32-39 of the entry-table entry are not all zeros.

The operation is suppressed.

The instruction-length code is 2.

The PC-translation-specification exception is indicated by a program-interruption code of 001F hex (or 009F hex if a concurrent PER event is indicated).

PER Event

A PER event is recognized when the CPU is enabled for PER and one or more of these events occur.

The PER mask, bit 1 of the EC-mode PSW, controls whether the CPU is enabled for PER. PER is disallowed in the BC mode. When the PER mask is zero, or in the BC mode, PER events are not recognized. When the bit is one, PER events are recognized, subject to the PER-event-mask bits in control register 9.

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

Additional information identifying the event is stored at real locations 150-155.

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

The PER event is indicated by setting bit 8 of the program-interruption code to one.

See the section "Program-Event Recording" in Chapter 4, "Control," for a detailed description of the PER event and the associated interruption information.

Primary-Authority Exception

A primary-authority exception is recognized during ASN authorization in PROGRAM TRANSFER with space switching (PT-ss) when either:

1. The authority-table entry indicated by the authorization index in control register 4 is beyond the length of the authority table designated by the ASN-second-table entry.

2. The primary-authority bit indicated by the authorization index is zero.

The ASN being translated is stored at real locations 146-147, and real locations 144-145 are set to zeros.

The operation is nullified.

The instruction-length code is 2.

The primary-authority exception is indicated by a program-interruption code of 0024 hex (or 00A4 hex if a concurrent PER event is indicated).

Privileged-Operation Exception

A privileged-operation exception is recognized when any of the following is true:

1. Execution of a privileged instruction is attempted in the problem state.

2. The value of the rightmost bit of the general register designated by the R_2 field of the PROGRAM TRANSFER instruction is zero and would cause the PSW problem-state bit to change from the problem state (one) to the supervisor state (zero).

3. In the problem state, the key value specified by the second operand of the SET PSW KEY FROM ADDRESS instruction corresponds to a zero PSW-key-mask bit in control register 3.

4. In the problem state, the key value specified by the rightmost byte of the register designated by the R_3 field of the MOVE WITH KEY instruction corresponds to a zero PSW-key-mask bit in control register 3.

5. In the problem state, the key value specified by the rightmost byte of the register designated by the R_3 field of the instructions MOVE TO PRIMARY and MOVE TO SECONDARY corresponds to a zero PSW-key-mask bit in control register 3.

6. In the problem state, any of the instructions

   EXTRACT PRIMARY ASN
   EXTRACT SECONDARY ASN
   INSERT ADDRESS SPACE CONTROL
   INSERT PSW KEY
   INSERT VIRTUAL STORAGE KEY

   is encountered, and the extraction-authority control, bit 4 of control register 0, is zero.

7. In the problem state, the result of ANDing the authorization key mask (AKM) with the PSW-key mask in control register 3 during PROGRAM CALL produces a result of zero.

The operation is suppressed.

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

The privileged-operation exception is indicated by a program-interruption code of 0002 hex (or 0082 hex if a concurrent PER event is indicated).

Protection Exception

A protection exception is recognized when any of the following is true:

1. Key-Controlled Protection: The CPU attempts to access a storage location that is protected against the type of reference, and the access key does not match the storage key.

2. Low-Address Protection: The CPU attempts a store that is subject to low-address protection, the effective address is in the range 0-511, and the low-address protection control, bit 3 of control register 0, is one.

3. Segment Protection: The CPU attempts to store, with DAT on, into a segment which has the segment-protection bit set to one.

The operation is suppressed when the location of the instruction is protected against fetching. Similarly, the operation is suppressed when the location of the target instruction of 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 figure "Summary of Action for Protection and Addressing Exceptions," which is included in the section "Addressing Exception" in this chapter.

For termination, changes may occur only to result fields. In this context, the term "result field" includes condition code, registers, and storage locations, if any, which are due to be changed by the instruction. However, no change is made to a storage location when a reference to that location causes a protection exception. Therefore, if an instruction is due to change only the contents of a field in storage, and every byte of that field would cause a protection exception, the operation is suppressed. When termination occurs on fetching, the protected information is not loaded into an addressable register nor moved to another storage location.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2.

For a protected operand location, the instruction-length code (ILC) is 1, 2, or 3, indicating the length of the instruction that caused the reference. However, on some models without the translation facility, an ILC of 0 occurs when a protection exception is recognized for a store-type reference.

The protection exception is indicated by a program-interruption code of 0004 hex (or 0084 hex if a concurrent PER event is indicated).

Secondary-Authority Exception

A secondary-authority exception is recognized during ASN authorization in SET SECONDARY ASN with space switching (SSAR-ss) when either:

1. The authority-table entry indicated by the authorization index in control register 4 is beyond the length of the authority table designated by the ASN-second-table entry.
2. The secondary-authority bit indicated by the authorization index is zero.

The ASN being translated is stored at real locations 146-147, and real locations 144-145 are set to zeros.

The operation is nullified.

The instruction-length code is 2.

The secondary-authority exception is indicated by a program-interruption code of 0025 hex (or 00A5 hex if a concurrent PER event is indicated).

Segment-Translation Exception

A segment-translation exception is recognized when either:

1. The segment-table entry indicated by the segment-index portion of a virtual address is outside the segment table.
2. The segment-invalid bit is one.

The exception is recognized as part of the execution of 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 and TEST PROTECTION, in which case the condition is indicated by the setting of the condition code.

The segment-index and page-index portion of the virtual address causing the exception is stored at real locations 145-147. When DAS is installed, bit 0 of real location 144 is set to zero if the virtual address was relative to the primary address space, or it is set to one if the virtual address was relative to the secondary address space. When DAS is not installed, bit 0 of real location 144 is set to zero. Bits 1-7 of real location 144 are set to zeros. When 2K-byte pages are used, the rightmost 11 bits of the address stored are unpredictable; when 4K-byte pages are used, the rightmost 12 bits of the address stored are unpredictable.

The unit of operation is nullified.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The segment-translation exception is indicated by a program-interruption code of 0010 hex (or 0090 hex if a concurrent PER event is indicated).

Significance Exception

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

The interruption may be disallowed by the significance mask (PSW bit 23 in the EC mode and PSW bit 39 in the BC mode).

The operation is completed. The significance mask also affects the result of the operation. When the mask bit is zero, the operation is completed by replacing the result with a true zero. When the mask bit is one, the operation is completed without further change to the characteristic of the result.

The instruction-length code is 1 or 2.

The significance exception is indicated by a program-interruption code of XX0E hex (or XX8E hex if a concurrent PER event is indicated), where XX is the exception-extension code.

Space-Switch Event

A space-switch event is recognized at the completion of a PROGRAM CALL with space switching (PC-ss) or a PROGRAM TRANSFER with space switching (PT-ss) when any of the following is true:

1. The space-switch-event-control bit, bit 31 of control register 1, is one before the operation.

2. The space-switch-event-control bit is one after the operation.

3. A PER event is reported.

The old PASN, which is in the right half of control register 4 before the execution of the instruction PC-ss or PT-ss, is stored at real locations 146-147. The old space-switch-event-control bit is placed in bit position 0 and zeros are placed in bit positions 1-15 at real locations 144-145.

The operation is completed.

The instruction-length code is 2.

The space-switch event is indicated by a program-interruption code of 001C hex (or 009C hex if a concurrent PER event is indicated).

Programming Notes

1. The space-switch event permits the control program to gain control whenever a program enters or leaves a particular address space. The space-switch-event-control bit is loaded into control register 1, along with the remaining bits of the primary segment-table designation, whenever control register 1 is loaded.

2. The space-switch event may be useful in obtaining programmed authorization checking, in causing additional trace information to be recorded, or in enabling or disabling the CPU for PER or tracing.

3. Bit 95 of the ASN-second-table entry (ASTE) is loaded into bit position 31 of control register 1 as part of the PC-ss and PT-ss operations. If bit 95 of the ASTE for a particular address space is set to one, then a space-switch event is recognized when a program enters or leaves the address space by means of either a PC-ss or a PT-ss.

4. The occurrence of a space-switch event at the completion of a PC-ss or PT-ss when any PER event is indicated permits the control program to determine the address space from which the instruction causing the PER event was fetched.

Special-Operation Exception

A special-operation exception is recognized when any of the following is true:

1. Execution of SET SYSTEM MASK is attempted in the supervisor state and the SSM-suppression control, bit 1 of control register 0, is one.

2. Execution of any of the following instructions is attempted with DAT off:

   EXTRACT PRIMARY ASN
   EXTRACT SECONDARY ASN
   INSERT ADDRESS SPACE CONTROL
   INSERT VIRTUAL STORAGE KEY
   MOVE TO PRIMARY
   MOVE TO SECONDARY
   SET ADDRESS SPACE CONTROL
   SET SECONDARY ASN

3. Execution of PROGRAM CALL or PROGRAM TRANSFER is attempted, and

the CPU is not in the primary-space mode.

4. Execution of LOAD ADDRESS SPACE PARAMETERS, PROGRAM CALL with space switching (PC-ss), PROGRAM TRANSFER with space switching (PT-ss), or SET SECONDARY ASN (SSAR-cp or SSAR-ss) is attempted, and the ASN-translation control, bit 12 of control register 14, is zero.

5. Execution of PROGRAM CALL or PROGRAM TRANSFER is attempted and, the subsystem-linkage control, bit 0 of control register 5, is zero.

6. Execution of SET ADDRESS SPACE CONTROL, MOVE TO PRIMARY, or MOVE TO SECONDARY is attempted, and the secondary-space control, bit 5 of control register 0, is zero.

7. The storage-key 4K-byte-block facility is installed; execution of the instruction INSERT STORAGE KEY, RESET REFERENCE BIT, or SET STORAGE KEY is attempted; and the storage-key-exception control, bit 7 of control register 0, is zero.

The operation is suppressed.

The instruction-length code is 1, 2, or 3, and indicates the length of the instruction causing the exception.

The special-operation exception is indicated by a program-interruption code of 0013 hex (or 0093 hex if a concurrent PER event is indicated).

Specification Exception

A specification exception is recognized when any of the following is true:

1. A one is introduced into an unassigned bit position of an EC-mode PSW (that is, any of bit positions 0, 2-4, 17, or 24-39). This is handled as an early PSW specification exception.

2. A PSW is introduced in which the EC mode is specified (PSW bit 12 is one) in a CPU that does not have the translation facility installed. This is handled as an early PSW specification exception.

3. A one is introduced into an EC-mode PSW bit position, other than in the I/O-mask or program-mask field, specifying a mode or facility that is not installed in the CPU. For example, bit 16 is one, and DAS is not installed. This is handled as an early PSW specification exception.

4. The PSW contains an odd instruction address.

5. An operand address does not designate an integral boundary in an instruction requiring such integral-boundary designation.

6. An odd-numbered general register is designated by an R field of an instruction that requires an even-numbered register designation.

7. A floating-point register other than 0, 2, 4, or 6 is designated for a short or long operand, or a floating-point register other than 0 or 4 is designated for an extended operand.

8. The multiplier or divisor in decimal arithmetic exceeds 15 digits and sign.

9. The length of the first-operand field is less than or equal to the length of the second-operand field in decimal multiplication or division.

10. Bit positions 8-11 of MONITOR CALL do not contain zeros.

11. Bits 20-22 of the second-operand address of SET ADDRESS SPACE CONTROL are not all zeros.

12. The leftmost eight bits of the general register designated by the R_2 field of PROGRAM TRANSFER are not zeros.

13. Execution of PROGRAM CALL, PROGRAM TRANSFER, or SET SECONDARY ASN is attempted with DAS tracing enabled, and (1) bits 29-31 of the trace-table designation contained in the word at logical location 84 are not all zeros, or (2) the new value of bits 27-31 of the trace-table-entry header would not be zero.

14. The storage address in INSERT STORAGE KEY or SET STORAGE KEY does not have zeros in the four rightmost bit positions.

The execution of the instruction identified by the old PSW is suppressed. However, for early PSW specification exceptions (causes 1-3), the operation that introduces the new PSW is completed, but an interruption occurs immediately thereafter.

Except as noted below, the instruction-length code (ILC) is 1, 2, or 3, indicating the length of the instruction causing the exception.

When the instruction address is odd (cause 4), it is unpredictable whether the ILC is 1, 2, or 3.

When the exception is recognized because of an early PSW specification exception, (causes 1-3), and the exception has been introduced by LOAD PSW or an interruption, the ILC is 0. When the exception is introduced by SET SYSTEM MASK or by STORE THEN OR SYSTEM MASK, the ILC is 2.

The specification exception is indicated by a program-interruption code of 0006 hex (or 0086 hex if a concurrent PER event is indicated).

Programming Note

See the section "Exceptions Associated with the PSW" in this chapter for a definition of when the exceptions associated with the PSW are recognized.

Translation-Specification Exception

A translation-specification exception is recognized when translation of a virtual address is attempted and any of the following is true:

1. Bit positions 8-12 of control register 0 do not contain one of the codes 01000, 01010, 10000, or 10010. When the translation facility is installed but the 1M-byte segment size is not provided, the exception is recognized when bit positions 8-12 do not contain one of the codes 01000 or 10000. On models offering only the 4K-byte page size, the exception is recognized when bit positions 8-12 do not contain the code 10000.

2. The segment-table entry used for the translation is valid and bit positions 4-7 and 29-30 in the entry do not contain zeros. (On some models, these bit positions are ignored and not checked for zeros.) When the segment-protection facility is installed, bit 29 of the segment-table entry is used to indicate segment protection and need not be zero. When the common-segment facility is installed, bit 30 is interpreted as the common-segment bit and need not be zero.

3. The page-table entry used for the translation is valid and bit position 14, when 2K-byte pages are used, or bit positions 13-14, when 4K-byte pages are used, in the entry do not contain zeros. When the extended-real-addressing facility is installed, and when 4K-byte pages are used, bit positions 13 and 14 of the page-table entry are the extended-storage-address bits and need not be zeros.

The exception is recognized only as part of the execution of an instruction using address translation, that is, when DAT is on and a logical address, instruction address, or virtual address must be translated, or when LOAD REAL ADDRESS or INVALIDATE PAGE TABLE ENTRY is executed. Cause 1 is recognized on any translation attempt; causes 2 and 3 are recognized only for table entries that are actually used.

The unit of operation is suppressed.

When the exception occurs during fetching of an instruction, it is unpredictable whether the ILC is 1, 2, or 3. When the exception occurs during the fetching of the target of EXECUTE, the ILC is 2.

When the exception occurs during a reference to an operand location, the instruction-length code (ILC) is 1, 2, or 3 and indicates the length of the instruction causing the exception.

The translation-specification exception is indicated by a program-interruption code of 0012 hex (or 0092 hex if a concurrent PER event is indicated).

Programming Note

When a translation-specification exception is recognized in the process of translating an instruction address, the operation is suppressed. In this case, the instruction-length code (ILC) is needed to derive the address of the instruction, as the instruction address in the old PSW has been incremented by the amount indicated by the ILC. In the case of segment-translation and page-translation exceptions, the operation is nullified, the instruction address in the old PSW identifies the instruction, and the ILC may be arbitrarily set to 1, 2, or 3.

Unnormalized-Operand Exception

An unnormalized-operand exception is recognized when, in a vector floating-point divide or multiply operation, a source-operand element has a nonzero fraction with a leftmost hexadecimal digit of zero. For more details, see the publication *IBM System/370 Vector Operations*, SA22-7125.

The unit of operation is inhibited.

The instruction-length code is 2.

The unnormalized-operand exception is indicated by a program-interruption code of XX1E hex (or XX9E hex if a concurrent PER event is indicated), where XX is the exception-extension code.

Vector-Operation Exception

A vector-operation exception is recognized when a vector-facility instruction is executed while bit 14 of control register 0 is zero on a CPU which has the vector facility installed and available. The vector-operation exception is also recognized when a vector-facility instruction is executed and the vector facility is not installed or available on this CPU, but the facility can be made available to the program either on this CPU or another CPU in the configuration.

When a vector-facility instruction is executed, and the vector facility is not installed on any CPU which is or can be placed in the configuration, it depends on the model whether a vector-operation exception or an operation exception is recognized.

The operation is nullified when the vector-operation exception is recognized.

The instruction-length code is 2 or 3.

The vector-operation exception is indicated by a program-interruption code of 0019 hex (or 0099 hex if a concurrent PER event is indicated).

COLLECTIVE PROGRAM-INTERRUPTION NAMES

For the sake of convenience, certain program exceptions are grouped together under a single collective name. These collective names are used when it is necessary to refer to the complete set of exceptions, such as in instruction definitions. Three collective names are used:

Access exceptions
ASN-translation exceptions
Trace exceptions

The individual exceptions and their priorities are listed in the section "Multiple-Program-Interruption Conditions" in this chapter.

RECOGNITION OF ACCESS EXCEPTIONS

The figure "Handling of Access Exceptions" summarizes the conditions that can cause access exceptions and the action taken when they are encountered.

Condition	Translation for Virtual Address of LRA		Translation and Access for Logical Address of TPROT		Translation and Access for Any Other Address	
	Indication	Action	Indication	Action	Indication	Action
Control-register-0 contents[1]						
Invalid encoding of bits 8-12	TS	Suppress	-[2]	-[2]	TS	Suppress
Segment-table entry						
Segment-table-length violation	cc3	Complete	cc3	Complete	ST	Nullify
Entry protected against fetching	-	-	-	-	-	-
Invalid address of entry	A	Suppress	A	Suppress	A	Suppress
I bit on	cc1	Complete	cc3	Complete	ST	Nullify
One in a bit position which is checked for zero[3]	TS	Suppress	TS	Suppress	TS	Suppress
Page-table entry						
Page-table-length violation	cc3	Complete	cc3	Complete	PT	Nullify
Entry protected against fetching	-	-	-	-	-	-
Invalid address of entry	A	Suppress	A	Suppress	A	Suppress
I bit on	cc2	Complete	cc3	Complete	PT	Nullify
One in a bit position which is checked for zero[3]	TS	Suppress	TS	Suppress	TS	Suppress
Access for instruction fetch						
Location protected	-	-	-	-	P	Suppress
Invalid address	-	-	-	-	A	Suppress
Access for operands						
Location protected	-	-	cc set[4]	Complete	P	Term.*
Invalid address	-	-	A	Suppress	A	Term.*

Explanation:

- The condition does not apply.

* Action is to terminate except where otherwise specified in this publication.

[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; when DAT is on, it is recognized during translation of the instruction address, and, when DAT is off, it is only recognized during execution of INVALIDATE PAGE TABLE ENTRY or for translation of the operand address of LOAD REAL ADDRESS.

[2] A translation-specification exception cannot occur for the logical address of TEST PROTECTION because this exception would have been recognized during the instruction fetch for the instruction.

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

[4] The condition code is set as follows:

0 Operand location not protected.
1 Fetches permitted, but stores not permitted.
2 Neither fetches nor stores permitted.

A Addressing exception.
cc1 Condition code 1 set.
cc2 Condition code 2 set.
cc3 Condition code 3 set.
P Protection exception.
PT Page-translation exception.
ST Segment-translation exception.
TS Translation-specification exception.

Handling of Access Exceptions

Any access exception is recognized as part of the execution of the instruction with which the exception is associated. An access exception is not recognized when the CPU attempts to prefetch from an unavailable location or detects some other access-exception condition, but a branch instruction or an interruption changes the instruction sequence such that the instruction is not executed.

Every instruction can cause an access exception to be recognized because of instruction fetch. Additionally, access exceptions associated with instruction execution may occur because of an access to an operand in storage.

An access exception due to fetching an instruction is indicated when the first instruction halfword cannot be fetched without encountering the exception. When the first halfword of the instruction has no access exceptions, access exceptions may be indicated for additional halfwords according to the instruction length specified by the first two bits of the instruction; however, when the operation can be performed without accessing the second or third halfwords of the instruction, it is unpredictable whether the access exception is indicated for the unused part. Since the indication of access exceptions for instruction fetch is common to all instructions, it is not covered in the individual instruction definitions.

Except where otherwise indicated in the individual instruction description, the following rules apply for exceptions associated with an access to an operand location. For a fetch-type operand, access exceptions are necessarily indicated only for that portion of the operand which is required for completing the operation. It is unpredictable whether access exceptions are indicated for those portions of a fetch-type operand which are not required for completing the operation. For a store-type operand, access exceptions are recognized for the entire operand even if the operation could be completed without the use of the inaccessible part of the operand. In situations where the value of a store-type operand is defined to be unpredictable, it is unpredictable whether an access exception is indicated.

Whenever an access to an operand location can cause an access exception to be recognized, the word "access" is included in the list of program exceptions in the description of the instruction. This entry also indicates which operand can cause the exception to be recognized and whether the exception is recognized on a fetch or store access to that operand location. Access exceptions are recognized only for the portion of the operand as defined by each particular instruction.

MULTIPLE PROGRAM-INTERRUPTION CONDITIONS

Except for PER events, only one program-interruption condition is indicated with a program interruption. The existence of one condition, however, does not preclude the existence of other conditions. When more than one program-interruption condition exists, only the condition having the highest priority is identified in the interruption code.

With two conditions 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 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 figure "Priority of Program-Interruption Conditions" lists the priorities of all program-interruption conditions other than PER events and exceptions associated with DAS. All exceptions associated with references to storage for a particular instruction halfword or a particular operand byte are grouped as a single entry called "access." The figure "Priority of Access Exceptions" lists the priority of access exceptions for a single access. Thus, the second figure specifies which of several exceptions, encountered either in the access of a particular portion of an instruction or in any particular access associated with an operand, has highest priority, and the first figure specifies the priority of this condition in relation to other conditions detected in the operation. Similarly, the priorities for exceptions occurring as part of ASN translation and tracing are covered in the figures "Priority of ASN-Translation Exceptions" and "Priority of Trace Exceptions," respectively.

For some instructions, the priority is shown in the individual instruction description.

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

To understand the use of the table, consider an example involving the instruction ADD DECIMAL, which is a six-byte instruction. Assume that the first four bytes of the instruction can be accessed but that the instruction crosses a boundary so that an addressing exception exists for the last two bytes. Additionally, assume that the first operand addressed by the instruction contains invalid decimal digits and is in a location that can be fetched from, but not stored into, because of key-controlled protection. The three exceptions which could result from attempted execution of the ADD DECIMAL are:

Priority Number	Exception
7.B	Access exceptions for third instruction halfword.
8.B	Access exceptions (operand 1).
8.D	Data exception.

Since the first inequality (7≠8) is between numeric characters, the addressing exception would be indicated. If, however, the entire ADD DECIMAL instruction can be fetched, and only the second two exceptions listed above exist, then the inequality (B≠D) is between alphabetic characters, and it is unpredictable whether the protection exception or the data exception would be indicated.

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).

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

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

3. Access exceptions for first halfword of EXECUTE.[2]

4. Access exceptions for second halfword of EXECUTE.[2]

5. Specification exception due to target instruction of EXECUTE not being specified on halfword boundary.[2]

6. Access exceptions for first instruction halfword.

7.A Access exceptions for second instruction halfword.[3]

7.B Access exceptions for third instruction halfword.[3]

7.C.1 Vector-operation exception.

7.C.2 Operation exception.

7.C.3 Privileged-operation exception for privileged instructions.

7.C.4 Execute exception.

7.C.5 Special-operation exception.

7.D Specification exception caused by an uninstalled instruction that has an assigned operation code (for example, an uninstalled floating-point instruction designating an odd floating-point register).

8.A Specification exception due to conditions other than those included in 2, 5, and 7.D above.

8.B[4] Access exceptions for an access to an operand in storage.[5]

8.C[4] Access exceptions for any other access to an operand in storage.[5]

8.D Data exception.[6]

8.E Decimal-divide exception.[7]

9. Events other than PER events, exceptions which result in completion, and the following exceptions: fixed-point divide, floating-point divide, and unnormalized operand. Either these exceptions and events are mutually exclusive or their priority is specified in the corresponding definitions.

Priority of Program-Interruption Conditions (Part 1 of 2)

Explanation:

Numbers indicate priority, with "1" being the highest priority; letters indicate no priority.

[1] PSW errors which cause an immediate interruption may be introduced by a new PSW loaded as a result of an interruption or by the instructions LOAD PSW, SET SYSTEM MASK, and STORE THEN OR SYSTEM MASK. The priority shown in the chart is for a PSW error introduced by an interruption and may also be considered as the priority for a PSW error introduced by the previous instruction. The error is introduced only if the instruction encounters no other exceptions. The resulting interruption has a higher priority than any interruption caused by the instruction which would have been executed next; it has lower priority, however, than any interruption caused by the instruction which introduced the erroneous PSW.

[2] Priorities 3, 4, and 5 are for the EXECUTE instruction, and priorities starting with 6 are for the target instruction. When no EXECUTE is encountered, priorities 3, 4, and 5 do not apply.

[3] Separate accesses may occur for each halfword of an instruction. The second instruction halfword is accessed only if bits 0-1 of the instruction are not both zeros. The third instruction halfword is accessed only if bits 0-1 of of the instruction are both ones. Access exceptions for one of these halfwords are not necessarily recognized if the instruction can be completed without use of the contents of the halfword or if an exception of lower priority can be determined without the use of the halfword.

[4] As in instruction fetching, separate accesses may occur for each portion of an operand. Each of these accesses is of equal priority, and the two entries 8.B and 8.C are listed to represent the relative priorities of exceptions associated with any two of these accesses. Access exceptions for INSERT STORAGE KEY, INSERT STORAGE KEY EXTENDED, INSERT VIRTUAL STORAGE KEY, INVALIDATE PAGE TABLE ENTRY, LOAD REAL ADDRESS, RESET REFERENCE BIT, RESET REFERENCE BIT EXTENDED, SET STORAGE KEY, SET STORAGE KEY EXTENDED, and TEST PROTECTION are also included in 8.B.

[5] 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.

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

[7] The exception can be indicated only if the digits used in establishing the exception, and also the signs, were fetched without encountering an access exception, only if the signs are valid, and only if the digits used in establishing the exception are valid.

Priority of Program-Interruption Conditions (Part 2 of 2)

Access Exceptions

The access exceptions consist of those exceptions which can be encountered while using an absolute, instruction, logical, real, or virtual address to access storage. Thus, with DAT on, the exceptions are:

1. Translation specification
2. Segment translation
3. Page translation
4. Addressing
5. Protection (key-controlled, segment, and low-address)

With DAT off, the exceptions are:

1. Addressing
2. Protection (key-controlled and low-address)

Additionally, the instructions LOAD REAL ADDRESS and INVALIDATE PAGE TABLE ENTRY can encounter a translation-specification exception even with DAT off.

A. Protection exception (low-address protection) due to a store-type operand reference with an effective address in the range 0-511.

B.1. Translation-specification exception due to invalid encoding of bits 8-12 of control register 0.[1]

B.2. Segment-translation exception due to segment-table entry being outside table.[2]

B.3. Addressing exception for access to segment-table entry.[3]

B.4. Segment-translation exception due to I bit in segment-table entry having the value one.[2]

B.5. Translation-specification exception due to invalid ones in segment-table entry.[3]

B.6.A. Protection exception (segment protection) due to a store-type operand reference to a virtual address which is protected against stores.[4]

B.6.B.1 Page-translation exception due to page-table entry being outside table.[2]

B.6.B.2 Addressing exception for access to page-table entry.[1]

B.6.B.3 Page-translation exception due to I bit in page-table entry having the value one.[2]

B.6.B.4 Translation-specification exception due to invalid ones in page-table entry.[3]

B.6.B.5 Addressing exception for access to instruction or operand.

B.7. Protection exception (key-controlled protection) due to attempt to access a protected instruction or operand location.

Explanation:

[1] Not applicable when DAT is off, except for execution of INVALIDATE PAGE TABLE ENTRY and for translation of operand address of LOAD REAL ADDRESS.

[2] Not applicable when DAT is off; not applicable to operand addresses for LOAD REAL ADDRESS and TEST PROTECTION.

[3] Not applicable when DAT is off except for translation of operand address for LOAD REAL ADDRESS.

[4] Not applicable when DAT is off.

Priority of Access Exceptions

ASN-Translation Exceptions

The ASN-translation exceptions are those exceptions which are common to the process of translating an ASN in the instructions PROGRAM CALL, PROGRAM TRANSFER, and SET SECONDARY ASN. The exceptions and the priority in which they are detected are shown in the figure "Priority of ASN-Translation Exceptions."

1. Addressing exception for access to ASN-first-table entry.

2. AFX-translation exception due to I bit (bit 0) in ASN-first-table entry being one.

3. ASN-translation-specification exception due to invalid ones (bits 1-7, 28-31) in ASN-first-table entry.

4. Addressing exception for access to ASN-second-table entry.

5. ASX-translation exception due to I bit (bit 0) in ASN-second-table entry being one.

6. ASN-translation-specification exception due to invalid ones (bits 1-7, 30, 31, 60-63, 97-103) in ASN-second-table entry.

Priority of ASN-Translation Exceptions

Trace Exceptions

The trace exceptions are those exceptions which can be encountered while performing the implicit tracing function. The exceptions, except for PER storage alteration, and their priority are shown in the figure "Priority of Trace Exceptions." PER storage alteration is recognized only if the instruction is completed.

1. Access exceptions (except for protection) for the trace-table designation at logical location 84.

2. Specification exception due to bits 29-31 of the word at trace-header address in logical location 84 not being zeros.

3.A Access exceptions (including low-address protection and segment protection) for first doubleword of trace-table-entry header.

3.B Access exceptions (except for protection) for third word of trace-table-entry header.

4. Specification exception if new value of trace-entry address in trace header would not designate a 32-byte boundary.

5. Access exceptions (including low-address protection and segment protection) for the trace entry.

Priority of Trace Exceptions

RESTART INTERRUPTION

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

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

If the CPU is in the operating state, the exchange of the PSWs occurs at the completion of the current unit of operation and after all other pending interruption conditions for which the CPU is enabled have been honored. 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 honoring any other pending interruptions.

The restart interruption is initiated by activating the restart key. When the multiprocessing facility is installed, the operation can also be initiated at the addressed CPU by executing a SIGNAL PROCESSOR instruction which specifies the restart order.

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

Programming Note

To perform a restart when the CPU is in the check-stop state, the CPU has to be reset. If the translation facility is installed, resetting with loss of the least amount of information can be accomplished by means of the system-reset-normal key, which does not clear the contents of program-addressable registers, including the control registers, but causes the channels to be reset. The program-reset SIGNAL PROCESSOR order can be used to perform a similar function.

SUPERVISOR-CALL INTERRUPTION

The supervisor-call interruption occurs when the instruction SUPERVISOR CALL is executed. The CPU cannot be disabled for the interruption, and the interruption occurs immediately upon the execution of the instruction.

The supervisor-call interruption causes the old PSW to be stored at real location 32 and a new PSW to be fetched from real location 96.

The contents of bit positions 8-15 of the SUPERVISOR CALL instruction are placed in the rightmost byte of the interruption code. The leftmost byte of the interruption code is set to zero. The instruction-length code is 1, unless the instruction was executed by means of EXECUTE, in which case the code is 2.

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

PRIORITY OF INTERRUPTIONS

During the execution of an instruction, several interruption-causing events may occur simultaneously. The instruction may give rise to a program interruption, a request for an external interruption may be received, equipment malfunctioning may be detected, an I/O-interruption request may be made, and the restart key may be activated. Instead of the program interruption, a supervisor-call interruption might occur; or both can occur if PER is active. Simultaneous interruption requests are honored in a predetermined order.

An exigent machine-check condition has the highest priority. When it occurs, the current operation is terminated or nullified. Program and supervisor-call interruptions that would have occurred as a result of the current operation may be eliminated. Any pending repressible machine-check conditions may be indicated with the exigent machine-check interruption. Every reasonable attempt is made to limit the side effects of an exigent machine-check condition, and requests for external, I/O, and restart interruptions normally remain unaffected.

In the absence of an exigent machine-check condition, interruption requests existing concurrently at the end of a unit of operation are honored, in descending order of priority, as follows:

Supervisor call
Program
Repressible machine check
External
Input/output
Restart

The processing of multiple simultaneous interruption requests consists in storing the old PSW and fetching the new PSW belonging to the interruption first honored. This new PSW is subsequently stored without the execution of any instructions, and the new PSW associated with the next interruption is fetched. Storing and fetching of PSWs continues until no more interruptions are to be serviced. The priority is reevaluated after each new PSW is loaded. Each evaluation takes into consideration any additional interruptions which may have become pending. Additionally, external and I/O interruptions, as well as machine-check interruptions due to repressible conditions, occur only if the current PSW at the instant of evaluation indicates that the CPU is interruptible for the cause.

Instruction execution is resumed using the last-fetched PSW. The order of executing interruption subroutines is, therefrre, the reverse of the order in which the PSWs are fetched.

If the new PSW for a program interruption does not specify the wait state and has an odd instruction address, or causes an access exception to be recognized, another program interruption occurs. Since this second interruption introduces the same unacceptable PSW, a string of interruptions is established. These program exceptions are recognized as part of the execution of the following instruction, and the string may be broken by an external, I/O, machine-check, or restart interruption or by the stop function.

If the new PSW for a program interruption contains a one in an unassigned bit position of an EC-mode PSW, or if it specifies the EC mode in a CPU that does not have the EC mode, 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 string of interruptions.

A string of interruptions for other interruption classes can also exist if the new PSW allows the interruption which has just occurred. These include machine-check interruptions, external interruptions, and I/O interruptions due to PCI conditions generated because of CCWs which form a loop. Furthermore, a string of interruptions involving more than one interruption class can exist. For example, assume that the CPU timer is negative and the CPU-timer subclass mask is one. If the external new PSW has a one in an unassigned bit position in the EC mode, and the program new PSW is enabled for external interruptions, then a string of interruptions occurs, alternating between external and program. Even more complex strings of interruptions are possible. As long as more interruptions must be serviced, the string of interruptions cannot be broken by employing the stop function; CPU reset is required.

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

On some models, when an excessive string 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 occur before the CPU is placed in the stopped state. When the CPU is in the stopped state, restart has the highest priority.

Programming Note

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

CHAPTER 7. GENERAL INSTRUCTIONS

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

DATA FORMAT

The general instructions treat data as being of four types: signed binary integers, unsigned binary integers, unstructured logical data, and decimal data. Data is treated as decimal by the conversion, packing, and unpacking instructions. Decimal data is described in Chapter 8, "Decimal Instructions."

The general instructions manipulate data which resides in general registers or in storage or is introduced from the instruction stream. Some general instructions operate on data which resides in the PSW or the TOD clock.

In a storage-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. Special rules apply to the operands of MOVE LONG and MOVE INVERSE.

BINARY-INTEGER REPRESENTATION

Binary integers are treated as signed or unsigned.

In an unsigned binary integer, all bits are used to express the absolute value of the number. When two unsigned binary integers of different lengths are added, the shorter number is considered to be extended on the left with zeros.

In some operations, the result is achieved by the use of the one's complement of the number. The one's complement of a number is obtained by inverting each bit of the number, including the sign.

For signed binary integers, the leftmost bit represents the sign, which is followed by the numeric field. Positive numbers are represented in true binary notation with the sign bit set to zero. When the value is zero, all bits are zeros, including the sign bit. Negative numbers are represented in two's-complement binary notation with a one in the sign-bit position.

Specifically, a negative number is represented by the two's complement of the positive number of the same absolute value. The two's complement of a number is obtained by forming the one's complement of the number, adding a value of one in the rightmost bit position, allowing a carry into the sign position, and ignoring any carry out of the sign position.

This number representation can be considered the rightmost portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number are zeros. When the number is negative, these bits are ones. Therefore, when a signed operand must be extended with bits on the left, the extension is achieved by setting these bits equal to the sign bit of the operand.

The notation for signed binary integers does not include a negative zero. It has a number range in which, for a given length, the set of negative nonzero numbers is one larger than the set of positive nonzero numbers. The maximum positive number consists of a sign bit of zero followed by all ones, whereas the maximum negative number (the negative number with the greatest absolute value) consists of a sign bit of one followed by all zeros.

A signed binary integer of either sign, except for zero and the maximum negative number, can be changed to a number of the same magnitude but opposite sign by forming its two's complement. Forming the two's complement of a number is equivalent to subtracting the number from zero. The two's complement of zero is zero.

The two's complement of the maximum negative number cannot be represented in the same number of bits. When an operation, such as LOAD COMPLEMENT, attempts to produce the two's complement of the maximum negative number, the result is the maximum negative number, and a fixed-point-overflow exception is recognized. An overflow does not result, however, when the maximum negative number is complemented as an intermediate result but the final result is within the representable range. An example of this case is a subtraction of the maximum negative number from -1. The product of two maximum negative numbers of a given length is representable as a positive number of double that length.

In discussions of signed binary integers in this publication, a signed binary integer includes the sign bit. Thus, the expression "32-bit signed binary integer" denotes an integer with 31 numeric bits and a sign bit, and the expression "64-bit signed binary integer" denotes an integer with 63 numeric bits and a sign bit.

In an arithmetic operation, a carry out of the numeric field of a signed binary integer is carried into the sign bit. However, in algebraic left-shifting, the sign bit does not change even if significant numeric bits are shifted out.

Programming Notes

1. An alternate way of forming the two's complement of a signed binary integer is to invert all bits to the left of the rightmost one bit, leaving the rightmost one bit and all zero bits to the right of it unchanged.

2. The numeric bits of a signed binary integer may be considered to represent a positive value, with the sign representing a value of either zero or the maximum negative number.

BINARY ARITHMETIC

SIGNED BINARY ARITHMETIC

Addition and Subtraction

Addition of signed binary integers is performed by adding all bits of each operand, including the sign bits. When one of the operands is shorter, the shorter operand is considered to be extended on the left to the length of the longer operand by propagating the sign-bit value.

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

Fixed-Point Overflow

A fixed-point-overflow condition exists for signed binary addition or subtraction when the carry out of the sign-bit position and the carry out of the leftmost numeric bit position disagree. Detection of an overflow does not affect the result produced by the addition. In mathematical terms, signed addition and subtraction produce a fixed-point overflow when the result is outside the range of representation for signed binary integers. Specifically, for ADD and SUBTRACT, which operate on 32-bit signed binary integers, there is an overflow when the proper result would be greater than or equal to $+2^{31}$ or less than -2^{31}. The actual result placed in the general register after an overflow differs from the proper result by 2^{32}. A fixed-point overflow causes a program interruption if allowed by the program mask.

The instructions SHIFT LEFT SINGLE and SHIFT LEFT DOUBLE produce an overflow when the result is outside the range of representation for signed binary integers. The actual result differs from that for addition and subtraction in that the sign of the result remains the same as the original sign.

UNSIGNED BINARY ARITHMETIC

Addition of unsigned binary integers is performed by adding all bits of each operand. When one of the operands is shorter, the shorter operand is considered to be extended on the left with zeros. Unsigned binary arithmetic is used in address arithmetic for adding the X, B, and D fields. (See the section "Address Generation" in Chapter 5, "Program Execution.") It is also used to obtain the addresses of the function bytes in TRANSLATE and TRANSLATE AND TEST. Furthermore, unsigned binary arithmetic is used on 32-bit unsigned binary integers by ADD LOGICAL and SUBTRACT LOGICAL. Given the same two operands, ADD and ADD LOGICAL produce the same 32-bit result. The instructions differ only in the interpretation of this result. ADD interprets the result as a signed binary integer and inspects it for sign, magnitude, and overflow to set the condition code accordingly. ADD LOGICAL interprets the result as an unsigned binary integer and sets the condition code according to whether the result is zero and whether there was a carry out of bit position 0. Such a carry is not considered an overflow, and no program interruption for overflow can occur for ADD LOGICAL.

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

Programming Notes

1. Logical addition and subtraction may be used to perform arithmetic on multiple-precision binary-integer operands. Thus, for multiple-precision addition, ADD LOGICAL can be used to add the corresponding parts of the operands beginning with the lowest-order parts. If the condition code indicates a carry, a value of one should be added to the sum of the next-higher-order parts. If the multiple-precision operands are signed, ADD should be used on the highest-order parts. The condition code then indicates any overflow or the proper sign and magnitude of the entire result; an overflow is also indicated by a program interruption for fixed-point overflow if allowed by the program mask. If the multiple-precision operands are unsigned, ADD LOGICAL should be used throughout.

2. Another use for ADD LOGICAL is to increment values representing binary counters, which are allowed to wrap around from all ones to all zeros without indicating overflow.

SIGNED AND LOGICAL COMPARISON

Comparison operations determine whether two operands are equal or not and, for most operations, which of two unequal operands is the greater (high). Signed-binary-comparison operations are provided which treat the operands as signed binary integers, and logical-comparison operations are provided which treat the operands as unsigned binary integers or as unstructured data.

COMPARE and COMPARE HALFWORD are signed-binary-comparison operations. These instructions are equivalent to SUBTRACT and SUBTRACT HALFWORD without replacing either operand, the resulting difference being used only to set the condition code. The operations permit comparison of numbers of opposite sign which differ by 2^{31} or more. Thus, unlike SUBTRACT, COMPARE cannot cause overflow.

Logical comparison of two operands is performed byte by byte, in a left-to-right sequence. The operands are equal when all their bytes are equal. When the operands are unequal, the comparison result is determined by a left-to-right comparison of corresponding bit positions in the first unequal pair of bytes: the zero bit in the first unequal pair of bits indicates the low operand, and the one bit the high operand. Since the remaining bit and byte positions do not change the comparison, it is not necessary to continue comparing unequal operands beyond the first unequal bit pair.

INSTRUCTIONS

The general instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of General Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

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

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designations for the assembler language are shown with each instruction. For LOAD AND TEST, for example, LTR is the mnemonic and R_1, R_2 the operand designation.

Name	Mne- monic	Characteristics											Op 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 AND SAVE	BASR	RR		BS						B	R		0D
BRANCH AND SAVE	BAS	RX		BS						B	R		4D
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 C. 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			AI	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

Summary of General Instructions (Part 1 of 3)

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

Summary of General Instructions (Part 2 of 3)

Explanation:

¢	Causes serialization and checkpoint synchronization.
¢[1]	Causes serialization and checkpoint synchronization when the M_1 and R_2 fields contain all ones and all zeros, respectively.
$	Causes serialization.
A	Access exceptions for logical addresses.
AI	Access exceptions for instruction address.
B	PER branch event.
BS	Branch-and-save facility.
C	Condition code is set.
D	Data exception.
EX	Execute exception.
GM	Instruction execution includes the implied use of general registers 1 and 2.
IF	Fixed-point-overflow exception.
II	Interruptible instruction.
IK	Fixed-point-divide exception.
L	New condition code is loaded.
MI	Move-inverse facility.
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 facility.

Summary of General Instructions (Part 3 of 3)

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 at the first-operand location. The operands and the sum are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

Access (fetch, operand 2 of A only)
Fixed-point overflow

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

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

<u>Resulting Condition Code</u>:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

<u>Program Exceptions</u>:

Access (fetch, operand 2)
Fixed-point overflow

<u>Programming Note</u>

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

ADD LOGICAL

```
ALR    R1,R2      [RR]

 ____________________
| '1E'   | R1 | R2 |
0        8    12   15
```

```
AL     R1,D2(X2,B2)      [RX]

 ______________________________________
| '5E'   | R1 | X2 | B2 |      D2      |
0        8    12   16   20             31
```

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

<u>Resulting Condition Code</u>:

0 Result zero; no carry
1 Result not zero; no carry
2 Result zero; carry
3 Result not zero; carry

<u>Program Exceptions</u>:

Access (fetch, operand 2 of AL only)

AND

```
NR     R1,R2      [RR]

 ____________________
| '14'   | R1 | R2 |
0        8    12   15
```

```
N      R1,D2(X2,B2)      [RX]

 ______________________________________
| '54'   | R1 | X2 | B2 |      D2      |
0        8    12   16   20             31
```

```
NI     D1(B1),I2        [SI]

 _____________________________________
| '94'   |   I2    | B1 |     D1      |
0        8         16   20            31
```

```
NC     D1(L,B1),D2(B2)        [SS]

 ____________________________/_______/__
| 'D4'   |    L    | B1 | D1 | B2 | D2 |
0        8         16   20   32   36   47
```

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

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

For AND (NC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

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

<u>Resulting Condition Code</u>:

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

<u>Program Exceptions</u>:

Access (fetch, operand 2, N and NC; fetch and store, operand 1, NI and NC)

Programming Notes

1. An example of the use of the AND instruction is given in Appendix A.

2. The AND instruction may be used to set a bit to zero.

3. Accesses to the first operand of AND (NI) and AND (NC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, the instruction AND cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

BRANCH AND LINK

BALR R_1,R_2 [RR]

```
┌─────────┬────┬────┐
│  '05'   │ R1 │ R2 │
└─────────┴────┴────┘
0         8    12   15
```

BAL $R_1,D_2(X_2,B_2)$ [RX]

```
┌─────────┬────┬────┬────┬───────────┐
│  '45'   │ R1 │ X2 │ B2 │    D2     │
└─────────┴────┴────┴────┴───────────┘
0         8    12   16   20          31
```

Information from the current PSW, including the updated instruction address, is loaded as link information at the first-operand location. Subsequently, the instruction address is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, bits 8-31 of general register R_2 are used as the branch address; however, when the R_2 field is zero, the operation is performed without branching. The branch address is computed before general register R_1 is changed.

The link information consists of the instruction-length code (ILC), the condition code (CC), the program mask bits, and the updated instruction address, arranged in the following format:

```
┌───┬──┬────┬──────────────────────┐
│   │  │Prog│                      │
│ILC│CC│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

1. An example of the use of the BRANCH AND LINK instruction is given in Appendix A.

2. When the R_2 field in the RR format is zero, the link information is loaded without branching.

3. When BRANCH AND LINK is the target instruction of EXECUTE, the instruction-length code is 2.

4. The format and the contents of the link information do not depend on whether the PSW specifies the EC or BC mode. In both modes, the link information is in the format of the rightmost 32 bit positions of the BC-mode PSW.

BRANCH AND SAVE

BASR R_1,R_2 [RR]

```
┌─────────┬────┬────┐
│  '0D'   │ R1 │ R2 │
└─────────┴────┴────┘
0         8    12   15
```

BAS $R_1,D_2(X_2,B_2)$ [RX]

```
┌─────────┬────┬────┬────┬───────────┐
│  '4D'   │ R1 │ X2 │ B2 │    D2     │
└─────────┴────┴────┴────┴───────────┘
0         8    12   16   20          31
```

The updated instruction address, with eight zeros appended on the left, is saved as link information at the first-operand location. Subsequently, the instruction address is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, bits 8-31 of general register R_2 are used as the branch address; however, when the R_2 field is zero, the operation is performed without branching. The branch address is computed before general register R_1 is changed.

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the branch-and-save facility is not installed)

Programming Notes

1. An example of the use of the BRANCH AND SAVE instruction is given in Appendix A.

2. The BRANCH AND SAVE instruction (BAS and BASR) may be used in place of the BRANCH AND LINK instruction (BAL and BALR) when it is desired to obtain the instruction address without the instruction-length code, program mask, and condition code.

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 instruction address in the current PSW is replaced by the branch address if the condition code has one of the values 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, bits 8-31 of general register R_2 are used as the branch address; however, when the R_2 field is zero, the operation is performed without branching.

The M_1 field is used as a four-bit mask. The four condition codes (0, 1, 2, and 3) correspond, left to right, with the four bits of the mask, as follows:

Condition Code	Instruction Bit No. of Mask	Mask Position Value
0	8	8
1	9	4
2	10	2
3	11	1

The current condition code is used to select the corresponding mask bit. If the mask bit selected by the condition code is one, the branch is successful. If the mask bit selected is zero, normal instruction sequencing proceeds with the next sequential instruction.

When the M_1 and R_2 fields of BRANCH ON CONDITION (BCR) are all ones and all zeros, respectively, a serialization and checkpoint-synchronization function is performed.

Condition Code: The code remains unchanged.

Program Exceptions: None.

Programming Notes

1. An example of the use of the BRANCH ON CONDITION instruction is given in Appendix A.

2. When a branch is to depend on more than one condition, the pertinent condition codes are specified in the mask as the sum of their mask position values. A mask of 12, for example, specifies that a branch is to be made when the condition code is 0 or 1.

3. When all four mask bits are zeros or when the R_2 field in the RR format contains zero, the branch instruction is equivalent to a no-operation. When all four mask bits are ones, that is, the mask value is 15, the branch is unconditional unless the R_2 field in the RR format is zero.

4. Execution of BCR 15,0 (that is, an instruction with a value of 07F0 hex) may result in significant performance degradation. To ensure optimum performance, the program should avoid use of BCR 15,0 except in cases when the serialization or the checkpoint-synchronization function is actually required.

5. Note that the relation between the RR and RX formats in branch-address specification is not the same as in operand-address specification. For branch instructions in the RX format, the branch address is the

address specified by X_2, B_2, and D_2; in the RR format, the branch address is contained in the register designated by R_2. For operands, the address specified by X_2, B_2, and D_2 is the operand address, but the register designated by R_2 contains the operand, not the operand address.

BRANCH ON COUNT

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

A one is subtracted from the first operand, and the result is placed at the first-operand location. The first operand and result are treated as 32-bit binary integers, with overflow ignored. When the result is zero, normal instruction sequencing proceeds with the updated instruction address. When the result is not zero, the instruction address in the current PSW is replaced by the branch address.

In the RX format, the second-operand address is used as the branch address. In the RR format, bits 8-31 of general register R_2 are used as the branch address; however, when the R_2 field is zero, the operation is performed without branching. The branch address is computed before general register R_1 is changed.

Condition Code: The code remains unchanged.

Program Exceptions: None.

Programming Notes

1. An example of the use of the BRANCH ON COUNT instruction is given in Appendix A.

2. The first operand and result can be considered as either signed or unsigned binary integers since the result of a binary subtraction is the same in both cases.

3. An initial count of one results in zero, and no branching takes place; an initial count of zero results in -1 and causes branching to be executed; an initial count of -1 results in -2 and causes branching to be executed; and so on. In a loop, branching takes place each time the instruction is executed until the result is again zero. Note that, because of the number range, an initial count of -2^{31} results in a positive value of $2^{31} - 1$.

4. 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

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 with a compare value. The result of the comparison determines whether branching occurs. Subsequently, the sum is placed at the first-operand location. The second-operand address is used as a branch address. The R_3 field designates registers containing the increment and the compare value.

For BRANCH ON INDEX HIGH, when the sum is high, the instruction address in the current PSW is replaced by the branch address. When the sum is low or equal, normal instruction sequencing proceeds with the updated instruction address.

For BRANCH ON INDEX LOW OR EQUAL, when the sum is low or equal, the instruction address in the current PSW is replaced by the branch address. When the sum is high, normal instruction sequencing proceeds with the updated instruction address.

When the R_3 field is even, it designates a pair of registers; the contents of the even and odd registers of the pair are used as the increment and the compare value, respectively. When the R_3 field is odd, it designates a single register, the contents of which are used as both the increment and the compare value.

For purposes of the addition and comparison, all operands and results are treated as 32-bit signed binary integers. Overflow caused by the addition is ignored.

The original contents of the compare-value register are used as the compare value even when that register is also specified to be the first-operand location. The branch address is computed before general register R_1 is changed.

The sum is placed at the first-operand location, regardless of whether the branch is taken.

Condition Code: The code remains unchanged.

Program Exceptions: None.

Programming Notes

1. Several examples of the use of the BRANCH ON INDEX HIGH and BRANCH ON INDEX LOW OR EQUAL instructions are given in Appendix A.

2. The word "index" in the names of these instructions indicates that one of the major purposes is the incrementing and testing of an index value. The increment, being a signed binary integer, may be used to increase or decrease the value in general register R_1 by an arbitrary amount.

COMPARE

CR R_1,R_2 [RR]

'19'	R_1	R_2
0	8	12 15

C $R_1,D_2(X_2,B_2)$ [RX]

'59'	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The first operand is compared with the second operand, and the result is indicated in the condition code. The operands are treated as 32-bit signed binary integers.

Resulting Condition Code:

0 Operands equal
1 First operand low
2 First operand high
3 --

Program Exceptions:

Access (fetch, operand 2 of C only)

COMPARE AND SWAP

CS $R_1,R_3,D_2(B_2)$ [RS]

'BA'	R_1	R_3	B_2	D_2
0	8	12	16	20 31

COMPARE DOUBLE AND SWAP

CDS $R_1,R_3,D_2(B_2)$ [RS]

'BB'	R_1	R_3	B_2	D_2
0	8	12	16	20 31

The first and second operands are compared. If they are equal, the third operand is stored at the second-operand location. If they are unequal, the second operand is loaded into the first-operand location. The result of the comparison is indicated in the condition code.

For COMPARE AND SWAP, the first and third operands are 32 bits in length, with each operand occupying a general register. The second operand is a word in storage.

For COMPARE DOUBLE AND SWAP, the first and third operands are 64 bits in length, with each operand occupying an even-odd pair of general registers. The second operand is a doubleword in storage.

When an equal comparison occurs, the third operand is stored at the second-operand location. The fetch of the second operand for purposes of comparison and the store into the second-operand location appear to be a block-concurrent interlocked-update reference as observed by other CPUs.

When the result of the comparison is unequal, the second-operand location remains unchanged. However, on some models, the value may be fetched and subsequently stored back unchanged at the second-operand location. This update appears to be a block-concurrent interlocked-update reference as observed by other CPUs.

A serialization function is performed before the operand is fetched and again after the operation is completed.

The second operand of COMPARE AND SWAP must be designated on a word boundary. The R_1 and R_3 fields for COMPARE DOUBLE AND SWAP must each designate an even register, and the second operand for the CDS instruction must be designated on a doubleword boundary. Otherwise, a specification exception is recognized.

Resulting Condition Code:

0	First and second operands equal, second operand replaced by third operand
1	First and second operands unequal, first operand replaced by second operand
2	--
3	--

Program Exceptions:

Access (fetch and store, operand 2)
Operation (if the conditional-swapping facility is not installed)
Specification

Programming Notes

1. Several examples of the use of the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions are given in Appendix A.

2. COMPARE AND SWAP can be used by CPU programs sharing common storage areas in either a multiprogramming or multiprocessing environment. Two examples are:

   a. By performing the following procedure, a CPU program can modify the contents of a storage location even though the possibility exists that the CPU program may be interrupted by another CPU program that will update the location or that another CPU program may simultaneously update the location. First, the entire word containing the byte or bytes to be updated is loaded into a general register. Next, the updated value is computed and placed in another general register. Then COMPARE AND SWAP is executed with the R_1 field designating the register that contains the original value and the R_3 field designating the register that contains the updated value. If the update has been successful, condition code 0 is set. If the storage location no longer contains the original value, the update has not been successful, the general register designated by the R_1 field of the COMPARE AND SWAP instruction contains the new current value of the storage location, and condition code 1 is set. When condition code 1 is set, the CPU program can repeat the procedure using the new current value.

   b. COMPARE AND SWAP can be used for controlled sharing of a common storage area, including the capability of leaving a message (in a chained list of messages) when the common area is in use. To accomplish this, a word in storage can be used as a control word, with a zero value in the word indicating that the common area is not in use and that no messages exist, a negative value indicating that the area is in use and that no messages exist, and a nonzero positive value indicating that the common area is in use and that the value is the address of the most recent message added to the list. Thus, any number of CPU programs desiring to seize the area can use COMPARE AND SWAP to update the control word to indicate that the area is in use or to add messages to the list. The single CPU program which has seized the area can also safely use COMPARE AND SWAP to remove messages from the list.

3. COMPARE DOUBLE AND SWAP can be used in a manner similar to that described for COMPARE AND SWAP. In addition, it has another use. Consider a chained list, with a control word used to address the first message in the list, as described in programming note 2b above. If multiple CPU programs are to be permitted to delete messages by using COMPARE AND SWAP (and not just the single CPU program which has seized the common area), there is a possibility the list will be incorrectly updated. This would occur if, for example, after one CPU program has fetched the address of the most recent message in order to remove the message, another CPU program removes the first two messages and then adds the first message back into the chain. The first CPU program, on continuing, cannot easily detect that the list is changed. By increasing the size of the control word to a doubleword

containing both the first message address and a word with a change number that is incremented for each modification of the list, and by using COMPARE DOUBLE AND SWAP to update both fields together, the possibility of the list being incorrectly updated is reduced to a negligible level. That is, an incorrect update can occur only if the first CPU program is delayed while changes exactly equal in number to a multiple of 2^{32} take place *and* only if the last change places the original message address in the control word.

4. COMPARE AND SWAP and COMPARE DOUBLE AND SWAP do not interlock against storage accesses by channels. Therefore, the instructions should not be used to update a location at which a channel program may store, since the channel-program data may be lost.

5. For the case of a condition-code setting of 1, COMPARE AND SWAP and COMPARE DOUBLE AND SWAP may or may not, depending on the model, cause any of the following to occur for the second-operand location: a PER storage-alteration event may be recognized; a protection exception for storing may be recognized; and, provided no access exception exists, the change bit may be set to one.

COMPARE HALFWORD

CH $R_1,D_2(X_2,B_2)$ [RX]

'49'	R_1	X_2	B_2	D_2
0	8	12	16	20 31

The first operand is compared with the second operand, and the result is indicated in the condition code. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand is treated as a 32-bit signed binary integer.

Resulting Condition Code:

0 Operands equal
1 First operand low
2 First operand high
3 --

Program Exceptions:

Access (fetch, operand 2)

Programming Note

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

COMPARE LOGICAL

CLR R_1,R_2 [RR]

'15'	R_1	R_2
0	8	12 15

CL $R_1,D_2(X_2,B_2)$ [RX]

'55'	R_1	X_2	B_2	D_2
0	8	12	16	20 31

CLI $D_1(B_1),I_2$ [SI]

'95'	I_2	B_1	D_1
0	8	16	20 31

CLC $D_1(L,B_1),D_2(B_2)$ [SS]

'D5'	L	B_1	D_1	B_2	D_2
0	8	16	20	32	36 47

The first operand is compared with the second operand, and the result is indicated in the condition code.

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the fields is reached. For COMPARE LOGICAL (CL) and COMPARE LOGICAL (CLC), access exceptions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte.

Resulting Condition Code:

0 Operands equal
1 First operand low
2 First operand high
3 --

Program Exceptions:

Access (fetch, operand 2, CL and CLC; fetch, operand 1, CLI and CLC)

Programming Notes

1. Examples of the use of the COMPARE LOGICAL instruction are given in Appendix A.

2. COMPARE LOGICAL treats all bits of each operand alike as part of a field of unstructured logical data. For COMPARE LOGICAL (CLC), the comparison may extend to field lengths of 256 bytes.

COMPARE LOGICAL CHARACTERS UNDER MASK

CLM $R_1,M_3,D_2(B_2)$ [RS]

```
 ____________________________________________
|  'BD'   |  R1  |  M3  |  B2  |      D2     |
|_________|______|______|______|_____________|
0         8      12     16     20           31
```

The first operand is compared with the second operand under control of a mask, and the result is indicated in the condition code.

The contents of the M_3 field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of general register R_1. The byte positions corresponding to ones in the mask are considered as a contiguous field and are compared with the second operand. The second operand is a contiguous field in storage, starting at the second-operand address and equal in length to the number of ones in the mask. The bytes in the general register corresponding to zeros in the mask do not participate in the operation.

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the fields is reached.

When the mask is not zero, exceptions associated with storage-operand access are recognized for no more than the number of bytes specified by the mask. Access exceptions may or may not be recognized for the portion of a storage operand to the right of the first unequal byte. When the mask is zero, access exceptions are recognized for one byte at the second-operand address.

Resulting Condition Code:

0	Operands equal, or mask bits all zeros
1	First operand low
2	First operand high
3	--

Program Exceptions:

Access (fetch, operand 2)

Programming Note

An example of the use of the COMPARE LOGICAL CHARACTERS UNDER MASK instruction is given in Appendix A.

COMPARE LOGICAL LONG

CLCL R_1,R_2 [RR]

```
 ______________________
|  '0F'   |  R1  |  R2 |
|_________|______|_____|
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 to be extended on the right with padding bytes.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by bits 8-31 of general registers R_1 and R_2, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 8-31 of general registers R_1 + 1 and R_2 + 1, respectively. Bit positions 0-7 of general register R_2 + 1 contain the padding byte. The contents of bit positions 0-7 of general registers R_1, R_2, and R_1 + 1 are ignored.

The contents of the registers just described are as follows:

```
R1
 _______________________________________
|//////// |  First-Operand Address      |
|_________|_____________________________|
0         8                            31

R1 + 1
 _______________________________________
|//////// |  First-Operand Length       |
|_________|_____________________________|
0         8                            31

R2
 _______________________________________
|//////// |  Second-Operand Address     |
|_________|_____________________________|
0         8                            31

R2 + 1
 _______________________________________
|   Pad   |  Second-Operand Length      |
|_________|_____________________________|
0         8                            31
```

The comparison proceeds left to right, byte by byte, and ends as soon as an inequality is found or the end of the longer operand is reached. If the operands are not of the same length, the shorter operand is considered to be extended on the right with the appropriate number of padding bytes.

If both operands are of zero length, the operands are considered to be equal.

The execution of the instruction is interruptible. When an interruption occurs, other than one that causes termination, the contents of general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes compared, and the contents of general registers R_1 and R_2 are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. The leftmost bits which are not part of the address in general registers R_1 and R_2 are set to zeros; the contents of bit positions 0-7 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged; and the condition code is unpredictable. If the operation is interrupted after the shorter operand has been exhausted, the length field pertaining to the shorter operand is zero, and its address is updated accordingly.

If the operation ends because of an inequality, the address fields in general registers R_1 and R_2 at completion identify the first unequal byte in each operand. The lengths in bit positions 8-31 of general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes that were equal, unless the inequality occurred with the padding byte, in which case the length field for the shorter operand is set to zero. The addresses in general registers R_1 and R_2 are incremented by the amounts by which the corresponding length fields were reduced.

If the two operands, including the padding byte, if necessary, are equal, both length fields are made zero at completion, and the addresses are incremented by the corresponding operand-length values.

At the completion of the operation, the leftmost bits which are not part of the address in general registers R_1 and R_2 are set to zeros, including the case when one or both of the initial length values are zero. The contents of bit positions 0-7 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged.

Access exceptions for the portion of a storage operand to the right of the first unequal byte may or may not be recognized. For operands longer than 2K bytes, access exceptions are not recognized more than 2K bytes beyond the byte being processed. Access exceptions are not indicated for locations more than 2K bytes beyond the first unequal byte.

When the length of an operand is zero, no access exceptions are recognized for that operand. Access exceptions are not recognized for an operand if the R field associated with that operand is odd.

Resulting Condition Code:

0 Operands equal, or both zero length
1 First operand low
2 First operand high
3 --

Program Exceptions:

Access (fetch, operands 1 and 2)
Specification

Programming Notes

1. An example of the use of the COMPARE LOGICAL LONG instruction is given in Appendix A.

2. When the R_1 and R_2 fields are the same, the operation proceeds in the same way as when two distinct pairs of registers having the same contents are specified, and, in the absence of dynamic modification of the operand area by another CPU or by a channel, condition code 0 is set. However, it is unpredictable whether access exceptions are recognized for the operand since the operation can be completed without storage being accessed.

3. Other programming notes concerning interruptible instructions are included in the section "Interruptible Instructions" in Chapter 5, "Program Execution."

4. Special precautions should be taken when COMPARE LOGICAL LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.

CONVERT TO BINARY

CVB $R_1,D_2(X_2,B_2)$ [RX]

'4F'	R_1	X_2	B_2	D_2

0 8 12 16 20 31

The second operand is changed from decimal to binary, and the result is placed at the first-operand location.

The second operand occupies eight bytes in storage and has the format of packed decimal data, as described in Chapter 8, "Decimal Instructions." It is checked for valid sign and digit codes, and a data exception is recognized when an invalid code is detected.

The result of the conversion is a 32-bit signed binary integer, which is placed in general register R_1. The maximum positive number that can be converted and still be contained in a 32-bit register is 2,147,483,647; the maximum negative number (the negative number with the greatest absolute value) that can be converted is -2,147,483,648. For any decimal number outside this range, the operation is completed by placing the 32 rightmost bits of the binary result in the register, and a fixed-point-divide exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)
Data
Fixed-point divide

Programming Notes

1. An example of the use of the CONVERT TO BINARY instruction is given in Appendix A.

2. When the second operand is negative, the result is in two's-complement notation.

3. The storage-operand references for CONVERT TO BINARY may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

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 first operand is changed from binary to decimal, and the result is stored at the second-operand location. The first operand is treated as a 32-bit signed binary integer.

The result occupies eight bytes in storage and is in the format for packed decimal data, as described in Chapter 8, "Decimal Instructions." The rightmost four bits of the result represent the sign. A positive sign is encoded as 1100; a negative sign is encoded as 1101.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

Programming Notes

1. An example of the use of the CONVERT TO DECIMAL instruction is given in Appendix A.

2. The number to be converted is a 32-bit signed binary integer obtained from a general register. Since 15 decimal digits are available for the result, and the decimal equivalent of 31 bits requires at most 10 decimal digits, an overflow cannot occur.

3. The storage-operand references for CONVERT TO DECIMAL may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

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 doubleword first operand (the dividend) is divided by the second operand (the divisor), and the remainder and the quotient are placed at the first-operand location.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The dividend is treated as a 64-bit signed binary integer. The divisor, the remainder, and the quotient are treated as 32-bit signed binary integers. The remainder is placed in general register

R_1, and the quotient is placed in general register $R_1 + 1$.

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 divisor is zero, or when the magnitudes of the dividend and divisor are such that the quotient cannot be expressed by a 32-bit signed binary integer, a fixed-point-divide exception is recognized. This includes the case of division of zero by zero.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2 of D only)
Fixed-point divide
Specification

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]

'57'	R_1	X_2	B_2	D_2	
0	8	12	16	20	31

XI $D_1(B_1),I_2$ [SI]

'97'	I_2	B_1	D_1	
0	8	16	20	31

XC $D_1(L,B_1),D_2(B_2)$ [SS]

'D7'	L	B_1	D_1	B_2	D_2	
0	8	16	20	32	36	47

The EXCLUSIVE OR of the first and second operands is placed at the first-operand location.

The connective EXCLUSIVE OR is applied to the operands bit by bit. A bit position in the result is set to one if the corresponding bit positions in the two operands are unlike; otherwise, the result bit is set to zero.

For EXCLUSIVE OR (XC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

For EXCLUSIVE OR (XI), the first operand is one byte in length, and only one byte is stored.

Resulting Condition Code:

0	Result 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 Notes

1. An example of the use of the EXCLUSIVE OR instruction is given in Appendix A.

2. EXCLUSIVE OR may be used to invert a bit, an operation particularly useful in testing and setting programmed binary bit switches.

3. A field EXCLUSIVE-ORed with itself becomes all zeros.

4. For EXCLUSIVE OR (XR), the sequence A EXCLUSIVE-OR B, B EXCLUSIVE-OR A, A EXCLUSIVE-OR B results in the exchange of the contents of A and B without the use of an additional general register.

5. Accesses to the first operand of EXCLUSIVE OR (XI) and EXCLUSIVE OR (XC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, EXCLUSIVE OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

EXECUTE

EX $R_1,D_2(X_2,B_2)$ [RX]

'44'	R_1	X_2	B_2	D_2	
0	8	12	16	20	31

The single instruction at the second-operand address is modified by the contents of general register R_1, and the resulting instruction, called the target instruction, is executed.

When the R_1 field is not zero, bits 8-15 of the instruction designated by the second-operand address are ORed with bits 24-31 of general register R_1. The ORing does not change either the contents of general register R_1 or the instruction in storage, and it is effective only for the interpretation of the instruction to be executed. When the R_1 field is zero, no ORing takes place.

The target instruction may be two, four, or six bytes in length. The execution and exception handling of the target instruction are exactly as if the target instruction were obtained in normal sequential operation, except for the instruction address and the instruction-length code.

The instruction address of the current PSW is increased by the length of EXECUTE. This updated address and the instruction-length code of EXECUTE are used, for example, as part of the link information when the target instruction is BRANCH AND LINK. When the target instruction is a successful branching instruction, the instruction address of the current PSW is replaced by the branch address specified by the target instruction.

When the target instruction is in turn EXECUTE, an execute exception is recognized.

The effective address of EXECUTE must be even; otherwise, a specification exception is recognized. When the target instruction is two or three halfwords in length but can be executed without fetching its second or third halfword, it is unpredictable whether access exceptions are recognized for the unused halfwords. Access exceptions are not recognized for the second-operand address when the address is odd.

The second-operand address of EXECUTE is an instruction address rather than a logical address; thus, when DAS is installed and the CPU is in the secondary-space mode, it is unpredictable whether the target instruction is fetched from the primary space or the secondary space. When DAS is not installed, an instruction address is the same as a logical address.

Condition Code: The code may be set by the target instruction.

Program Exceptions:

Access (fetch, target instruction)
Execute
Specification

Programming Notes

1. An example of the use of the EXECUTE instruction is given in Appendix A.

2. The ORing of eight bits from the general register with the designated instruction permits the indirect specification of the length, index, mask, immediate-data, register, or extended-op-code field.

3. The fetching of the target instruction is considered to be an instruction fetch for purposes of program-event recording and for purposes of reporting access exceptions.

4. An access or specification exception may be caused by EXECUTE or by the target instruction.

5. When an interruptible instruction is made the target of EXECUTE, the program normally should not designate any register updated by the interruptible instruction as the R_1, X_2, or B_2 register for EXECUTE. Otherwise, on resumption of execution after an interruption, or if the instruction is refetched without an interruption, the updated values of these registers will be used in the execution of EXECUTE. Similarly, the program should normally not let the destination field in storage of an interruptible instruction include the location of EXECUTE, since the new contents of the location may be interpreted when resuming execution.

6. EXECUTE should be executed in the secondary-space mode only if the virtual address of the target instruction translates to the same real address by means of both the primary segment table and secondary segment table. Otherwise, unpredictable results may occur.

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 general register R_1. 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 general register R_1 under control of a mask.

The contents of the M_3 field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of general register R_1. The byte positions corresponding to ones in the mask are filled, left to right, with bytes from successive storage locations beginning at the second-operand address. When the mask is not zero, the length of the second operand is equal to the number of ones in the mask. The bytes in the general register corresponding to zeros in the mask remain unchanged.

The resulting condition code is based on the mask and on the value of the bits inserted. When the mask is zero or when all inserted bits are zeros, the condition code is set to 0. When the inserted bits are not all zeros, the code is set according to the leftmost bit of the storage operand: if this bit is one, the code is set to 1; if this bit is zero, the code is set to 2.

When the mask is not zero, exceptions associated with storage-operand access are recognized only for the number of bytes specified by the mask. When the mask is zero, access exceptions are recognized for one byte at the second-operand address.

Resulting Condition Code:

0 All inserted bits zeros, or mask bits all zeros
1 Leftmost inserted bit one
2 Leftmost inserted bit zero, and not all inserted bits zeros
3 --

Program Exceptions:

Access (fetch, operand 2)

Programming Notes

1. Examples of the use of the INSERT CHARACTERS UNDER MASK instruction are given in Appendix A.

2. 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. Thus, the instruction may be used as a storage-to-register load-and-test operation.

3. INSERT CHARACTERS UNDER MASK with a mask of 1111 or 0001 performs a function similar to that of a LOAD (L) or INSERT CHARACTER (IC) instruction, respectively, with the exception of the condition-code setting. However, the performance of INSERT CHARACTERS UNDER MASK may be slower.

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 at the first-operand location.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2 of L only)

Programming Note

An example of the use of the LOAD instruction is given in Appendix A.

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 placed in bit positions 8-31 of general register R_1. Bits 0-7 of the register are set to zeros. The address computation follows the rules for address arithmetic.

No storage references for operands take place, and the address is not inspected for access exceptions.

Condition Code: The code remains unchanged.

Program Exceptions: None.

Programming Notes

1. An example of the use of the LOAD ADDRESS instruction is given in Appendix A.

2. LOAD ADDRESS may be used to increment the rightmost 24 bits of a general register, other than register 0, by the contents of the D_2 field of the instruction. The register to be incremented should be designated by R_1 and by either X_2 (with B_2 set to zero) or B_2 (with X_2 set to zero).

LOAD AND TEST

LTR R_1,R_2 [RR]

'12'	R_1	R_2

0 8 12 15

The second operand is placed unchanged at the first-operand location, and the sign and magnitude of the second operand, treated as a 32-bit signed binary integer, are indicated in the condition code.

Resulting Condition Code:

0 Result zero
1 Result less than zero
2 Result 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 at the first-operand location. The second operand and result are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

Fixed-point overflow

Programming Note

The operation complements all numbers. Zero and the maximum negative number remain unchanged. An overflow condition occurs when the maximum negative number is complemented.

LOAD HALFWORD

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 considered to be extended to a 32-bit signed binary integer and is placed at the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed binary integer. The second operand is extended to 32 bits by setting each of the 16 leftmost bit positions equal to the sign bit of the storage operand.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)

Programming Note

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

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 general register R_1 and ending with general register R_3 is loaded from storage beginning at the location designated by the second-operand address and continuing through as many locations as needed.

The general registers are loaded in the ascending order of their register numbers, starting with general register R_1 and continuing up to and including general register R_3, with general register 0 following general register 15.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)

Programming Note

All combinations of register numbers specified by R_1 and R_3 are valid. When the register numbers are equal, only four bytes are transmitted. When the number specified by R_3 is less than the number specified by R_1, the register numbers wrap around from 15 to 0.

LOAD NEGATIVE

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 at the first-operand location. The second operand and result are treated as 32-bit signed binary integers.

Resulting Condition Code:

0	Result zero
1	Result less than zero
2	--
3	--

Program Exceptions: None.

Programming Note

The operation complements positive numbers; negative numbers remain unchanged. The number zero remains unchanged.

LOAD 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 at the first-operand location. The second operand and the result are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

0	Result zero; no overflow
1	--
2	Result greater than zero; no overflow
3	Overflow

Program Exceptions:

Fixed-point overflow

Programming Note

The operation complements negative numbers; positive numbers and zero remain unchanged. An overflow condition occurs when the maximum negative number is complemented; the number remains unchanged.

MONITOR CALL

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.

The monitor-mask bits are in bit positions 16-31 of control register 8, which correspond to monitor classes 0-15, respectively.

Bit positions 12-15 in the I_2 field contain a binary number specifying one of 16 monitoring classes. When the monitor-mask bit corresponding to the class specified by the I_2 field is one, a monitor-event program interruption occurs. The contents of the I_2 field are stored at location 149, with zeros stored at location 148. Bit 9 of the program-interruption code is set to one.

The first-operand address is not used to address data; instead, the address specified by the B_1 and D_1 fields forms the monitor code, which is placed in the word at location 156. Address computation follows the rules of address arithmetic; bits 0-7 are set to zeros.

When the monitor-mask bit corresponding to the class specified by bits 12-15 of the instruction is zero, no interruption occurs, and the instruction is executed as a no-operation.

Bit positions 8-11 of the instruction must contain zeros; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

Monitor event
Specification

Programming Notes

1. MONITOR CALL provides the capability for passing control to a monitoring program when selected points are reached in the monitored program. This is accomplished by implanting MONITOR CALL instructions at the desired points in the monitored program. This function may be useful in performing various measurement functions; specifically, tracing information can be generated indicating which programs were executed, counting information can be generated indicating how often particular programs were used, and timing information can be generated indicating how long a particular program required for execution.

2. The monitor masks provide a means of disallowing all monitor-event program interruptions or allowing monitor-event program interruptions for all or selected classes.

3. The monitor code provides a means of associating descriptive information, in addition to the class number, with each MONITOR CALL. 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.

MOVE

MVI $D_1(B_1),I_2$ [SI]

'92'	I_2	B_1	D_1

0 8 16 20 31

MVC $D_1(L,B_1),D_2(B_2)$ [SS]

'D2'	L	B_1	D_1	B_2	D_2

0 8 16 20 32 36 47

The second operand is placed at the first-operand location.

For MOVE (MVC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand byte.

For MOVE (MVI), the first operand is one byte in length, and only one byte is stored.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2 of MVC; store, operand 1, MVI and MVC)

Programming Notes

1. Examples of the use of the MOVE instruction are given in Appendix A.

2. It is possible to propagate one byte through an entire field by having the first operand start one byte to the right of the second operand.

MOVE INVERSE

MVCIN $D_1(L,B_1),D_2(B_2)$ [SS]

'E8'	L	B_1	D_1	B_2	D_2	
0	8	16	20	32	36	47

The second operand is placed at the first-operand location with the left-to-right sequence of the bytes inverted.

The first-operand address designates the leftmost byte of the first operand. The second-operand address designates the rightmost byte of the second operand. Both operands have the same length.

The result is obtained as if the second operand were processed from right to left and the first operand from left to right. The second operand may wrap around from location 0 to location $2^{24} - 1$. The first operand may wrap around from location $2^{24} - 1$ to location 0.

When the operands overlap by more than one byte, the contents of the overlapped portion of the result field are unpredictable.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; store, operand 1)
Operation (if the move-inverse facility is not installed)

Programming Notes

1. An example of the use of the MOVE INVERSE instruction is given in Appendix A.

2. The contents of each byte moved remain unchanged.

3. MOVE INVERSE is the only SS-format instruction for which the second-operand address designates the rightmost, instead of the leftmost, byte of the second operand.

4. The storage-operand references for MOVE INVERSE may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

MOVE LONG

MVCL R_1,R_2 [RR]

'OE'	R_1	R_2	
0	8	12	15

The second operand is placed at the first-operand location, provided overlapping of operand locations would not affect the final contents of the first-operand location. The remaining rightmost byte positions, if any, of the first-operand location are filled with padding bytes.

The R_1 and R_2 fields each designate an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The location of the leftmost byte of the first operand and second operand is designated by bits 8-31 of general registers R_1 and R_2, respectively. The number of bytes in the first-operand and second-operand locations is specified by bits 8-31 of general registers $R_1 + 1$ and $R_2 + 1$, respectively. Bit positions 0-7 of register $R_2 + 1$ contain the padding byte. The contents of bit positions 0-7 of registers R_1, R_2, and $R_1 + 1$ are ignored.

The contents of the registers just described are as follows:

R_1

/////////	First-Operand Address
0	8 31

$R_1 + 1$

/////////	First-Operand Length
0	8 31

R_2

/////////	Second-Operand Address
0	8 31

$R_2 + 1$

Pad	Second-Operand Length
0	8 31

The movement starts at the left end of both fields and proceeds to the right. The operation is ended when the number of bytes specified by bit positions 8-31 of general register $R_1 + 1$ have been moved into the first-operand location. If the second operand is shorter than the first operand, the remaining rightmost bytes of the first-operand location are filled with the padding byte.

As part of the execution of the instruction, the values of the two length fields are compared for the setting of the condition code, and a check is made for destructive overlap of the operands. Operands are said to overlap destructively when the first-operand location is used as a source after data has been moved into it, assuming the inspection for overlap is performed by the use of logical operand addresses. When the operands overlap destructively, no movement takes place, and condition code 3 is set.

Operands do not overlap destructively, and movement is performed, if the leftmost byte of the first operand does not coincide with any of the second-operand bytes participating in the operation other than the leftmost byte of the second operand. When an operand wraps around from location 16,777,215 to location 0, operand bytes in locations up to and including 16,777,215 are considered to be to the left of bytes in locations from 0 up.

When the length specified by bit positions 8-31 of general register $R_1 + 1$ is zero, no movement takes place, and condition code 0 or 1 is set to indicate the relative values of the lengths.

The execution of the instruction is interruptible. When an interruption occurs other than one that causes termination, the contents of general registers $R_1 + 1$ and $R_2 + 1$ are decremented by the number of bytes moved, and the contents of general registers R_1 and R_2 are incremented by the same number, so that the instruction, when reexecuted, resumes at the point of interruption. The leftmost bits which are not part of the address in general registers R_1 and R_2 are set to zeros; the contents of bit positions 0-7 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged; and the condition code is unpredictable. If the operation is interrupted during padding, the length field in general register $R_2 + 1$ is 0, the address in general register R_2 is incremented by the original contents of general register $R_2 + 1$, and general registers R_1 and $R_1 + 1$ reflect the extent of the padding operation.

When the first-operand location includes the location of the instruction or of EXECUTE, the instruction may be refetched from storage and reinterpreted even in the absence of an interruption during execution. The exact point in the execution at which such a refetch occurs is unpredictable.

As observed by other CPUs and by channels, that portion of the first operand which is filled with the padding byte is not necessarily stored into in a left-to-right direction and may appear to be stored into more than once.

At the completion of the operation, the length in general register $R_1 + 1$ is decremented by the number of bytes stored at the first-operand location, and the address in general register R_1 is incremented by the same amount. The length in general register $R_2 + 1$ is decremented by the number of bytes moved out of the second-operand location, and the address in general register R_2 is incremented by the same amount. The leftmost bits which are not part of the address in general registers R_1 and R_2 are set to zeros, including the case when one or both of the original length values are zeros or when condition code 3 is set. The contents of bit positions 0-7 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged.

When condition code 3 is set, no exceptions associated with operand access are recognized. When the length of an operand is zero, no access exceptions for that operand are recognized. Similarly, when the second operand is longer than the first operand, access exceptions are not recognized for the part of the second-operand field that is in excess of the first-operand field. For operands longer than 2K bytes, access exceptions are not recognized for locations more than 2K bytes beyond the current location being processed. Access exceptions are not recognized for an operand if the R field associated with that operand is odd. Also, when the R_1 field is odd, PER storage-alteration events are not recognized, and no change bits are set.

Resulting Condition Code:

0	Operand lengths equal; no destructive overlap
1	First-operand length low; no destructive overlap
2	First-operand length high; no destructive overlap
3	No movement performed because of destructive overlap

Program Exceptions:

Access (fetch, operand 2; store, operand 1)
Specification

Programming Notes

1. An example of the use of the MOVE LONG instruction is given in Appendix A.

2. MOVE LONG may be used for clearing storage by setting the padding byte to zero and the second-operand length to zero. On most models, this is the fastest instruction for clearing storage areas in excess of 256 bytes. However, the stores associated with this clearing may be multiple-access stores and should not be used to clear an area if the possibility exists that another CPU or a channel will attempt to access and use the area as soon as it appears to be zero. For more details, see the section "Storage-Operand Consistency" in Chapter 5, "Program Execution."

3. The program should avoid specification of a length for either operand which would result in an addressing exception. Addressing (and also protection) exceptions may result in termination of the entire operation, not just the current unit of operation. The termination may be such that the contents of all result fields are unpredictable; in the case of MOVE LONG, this includes the condition code and the two even-odd general-register pairs, as well as the first-operand location in main storage. The following are situations that have actually occurred on one or more models:

   a. When a protection exception occurs on a 2K-byte block, or, when the storage-key 4K-byte-block facility is installed, on a 4K-byte block, of a first operand which is several blocks in length, stores to the protected block are suppressed. However, the move continues into the subsequent blocks of the first operand, which are not protected. Similarly, an addressing exception on a block does not necessarily suppress processing of subsequent blocks which are available.

   b. Some models may update the general registers only when an external, I/O, repressible machine-check, or restart interruption occurs, or when a program interruption occurs for which it is required to nullify or suppress a unit of operation. Thus, if, after a move into several blocks of the first operand, an addressing or protection exception occurs, the general registers may remain unchanged.

4. When the first-operand length is zero, the operation consists in setting the condition code and setting the leftmost bytes of general registers R_1 and R_2 to zero.

5. When the contents of the R_1 and R_2 fields are the same, the operation proceeds the same way as when two distinct pairs of registers having the same contents are designated. Condition code 0 is set.

6. The following is a detailed description of those cases in which movement takes place, that is, where destructive overlap does not exist. Depending on whether the second operand wraps around from location $2^{24} - 1$ to location 0, movement takes place in the following cases:

   a. When the second operand does not wrap around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, <u>or</u> if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.

   b. When the second operand wraps around, movement is performed if the leftmost byte of the first operand coincides with or is to the left of the leftmost byte of the second operand, <u>and</u> if the leftmost byte of the first operand is to the right of the rightmost second-operand byte participating in the operation.

   The rightmost second-operand byte is determined by using the smaller of the first-operand and second-operand lengths.

When the second-operand length is one or zero, destructive overlap cannot exist.

7. Special precautions should be taken if MOVE LONG is made the target of EXECUTE. See the programming note concerning interruptible instructions under EXECUTE.

8. Since the execution of MOVE LONG is interruptible, the instruction cannot be used for situations where the program must rely on uninterrupted execution of the instruction 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 or of EXECUTE because the new contents of the location may be interpreted for a resumption after an interruption, or the instruction may be refetched without an interruption.

9. Further programming notes concerning interruptible instructions are included in the section "Interruptible Instructions" in Chapter 5, "Program Execution."

MOVE NUMERICS

MVN $D_1(L,B_1),D_2(B_2)$ [SS]

'D1'	L	B_1	D_1	B_2	D_2
0	8	16	20	32	36 47

The rightmost four bits of each byte in the second operand are placed in the rightmost bit positions of the corresponding bytes in the first operand. The leftmost four bits of each byte in the first operand remain unchanged.

Each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes

1. An example of the use of the MOVE NUMERICS instruction is given in Appendix A.

2. MOVE NUMERICS moves the numeric portion of a decimal-data field that is in the zoned format. The zoned-decimal format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.

3. Accesses to the first operand of MOVE NUMERICS consist in fetching the rightmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

MOVE WITH OFFSET

MVO $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

'F1'	L_1	L_2	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The second operand is placed to the left of and adjacent to the rightmost four bits of the first operand.

The rightmost four bits of the first operand are attached as the rightmost bits to the second operand, the second operand bits are offset by four bit positions, and the result is placed at the first-operand location.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand is too short to contain all of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if the operands were processed one byte at a time, as if each result byte were stored immediately after fetching the necessary operand bytes, and as if the left digit of each second-operand byte were to remain available for the next result byte and need not be refetched.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes

1. An example of the use of the MOVE WITH OFFSET instruction is given in Appendix A.

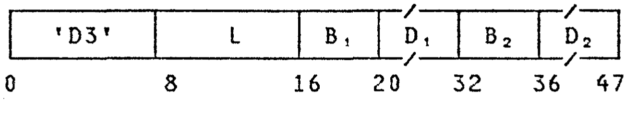
2. MOVE WITH OFFSET may be used to shift packed decimal data by an odd number of digit positions. The packed-decimal format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes. In many cases, however, SHIFT AND ROUND DECIMAL may be more convenient to use.

3. Access to the rightmost byte of the first operand of MOVE WITH OFFSET consists in fetching the rightmost four bits and subsequently storing the updated value of this byte. These fetch and store accesses to the rightmost byte of the first operand do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

4. The storage-operand references for MOVE WITH OFFSET may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

MOVE ZONES

MVZ $D_1(L,B_1),D_2(B_2)$ [SS]



The leftmost four bits of each byte in the second operand are placed in the leftmost four bit positions of the corresponding bytes in the first operand. The rightmost four bits of each byte in the first operand remain unchanged.

Each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte is fetched.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes

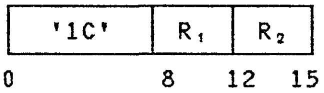
1. An example of the use of the MOVE ZONES instruction is given in Appendix A.

2. MOVE ZONES moves the zoned portion of a decimal field in the zoned format. The zoned format is described in Chapter 8, "Decimal Instructions." The operands are not checked for valid sign and digit codes.

3. Accesses to the first operand of MOVE ZONES consist in fetching the leftmost four bits of each byte in the first operand and subsequently storing the updated value of the byte. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for the OR (OI) instruction in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

MULTIPLY

MR R_1,R_2 [RR]



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 second word of the first operand (multiplicand) is multiplied by the second operand (multiplier), and the doubleword product is placed at the first-operand location.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

Both the multiplicand and multiplier are treated as 32-bit signed binary integers. The multiplicand is taken from general register $R_1 + 1$. The contents of general register R_1 are ignored. The product is a 64-bit signed binary integer, which replaces the contents of the even-odd pair of general registers designated by R_1. An overflow cannot occur.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2 of M only)
Specification

Programming Notes

1. An example of the use of the MULTIPLY instruction is given in Appendix A.

2. The significant part of the product usually occupies 62 bits or fewer. Only when two maximum negative numbers are multiplied are 63 significant product bits formed.

MULTIPLY HALFWORD

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 first operand (multiplicand) is multiplied by the second operand (multiplier), and the product is placed at the first-operand location. The second operand is two bytes in length and is considered to be a 16-bit signed binary integer.

The multiplicand is treated as a 32-bit signed binary integer and is replaced by the rightmost 32 bits of the signed-binary-integer product. The bits to the left of the 32 rightmost bits of the product are not tested for significance; no overflow indication is given.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand sign, except that a zero result is always positive.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)

Programming Notes

1. An example of the use of the MULTIPLY HALFWORD instruction is given in Appendix A.

2. The significant part of the product usually occupies 46 bits or fewer. Only when two maximum negative numbers are multiplied are 47 significant product bits formed. Since the rightmost 32 bits of the product are stored unchanged, ignoring all bits to the left, the sign bit of the result may differ from the true sign of the product in the case of overflow. For a negative product, the 32 bits placed in register R_1 are the rightmost part of the product in two's-complement notation.

OR

OR R_1,R_2 [RR]

'16'	R_1	R_2

0 8 12 15

O $R_1,D_2(X_2,B_2)$ [RX]

'56'	R_1	X_2	B_2	D_2

0 8 12 16 20 31

OI $D_1(B_1),I_2$ [SI]

'96'	I_2	B_1	D_1	
0	8	16	20	31

OC $D_1(L,B_1),D_2(B_2)$ [SS]

'D6'	L	B_1	D_1	B_2	D_2	
0	8	16	20	32	36	47

The OR of the first and second operands is placed at the first-operand location.

The connective OR is applied to the operands bit by bit. A bit position in the result is set to one if the corresponding bit position in one or both operands contains a one; otherwise, the result bit is set to zero.

For OR (OC), each operand is processed left to right. When the operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after fetching the necessary operand bytes.

For OR (OI), the first operand is only one byte in length, and only one byte is stored.

<u>Resulting</u> <u>Condition</u> <u>Code</u>:

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

<u>Program</u> <u>Exceptions</u>:

Access (fetch, operand 2, O and OC; fetch and store, operand 1, OI and OC)

<u>Programming</u> <u>Notes</u>

1. Examples of the use of the OR instruction are given in Appendix A.

2. OR may be used to set a bit to one.

3. Accesses to the first operand of OR (OI) and OR (OC) consist in fetching a first-operand byte from storage and subsequently storing the updated value. These fetch and store accesses to a particular byte do not necessarily occur one immediately after the other. Thus, OR cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

PACK

PACK $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

'F2'	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 zoned to packed, and the result is placed at the first-operand location. The zoned and packed formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as though it had the zoned format. The numeric bits of each byte are treated as a digit. The zone bits are ignored, except the zone bits in the rightmost byte, which are treated as a sign.

The sign and digits are moved unchanged to the first operand and are not checked for valid codes. The sign is placed in the rightmost four bit positions of the rightmost byte of the result field, and the digits are placed adjacent to the sign and to each other in the remainder of the result field.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if each result byte were stored immediately after fetching the necessary operand bytes. Two second-operand bytes are needed for each result byte, except for the rightmost byte of the result field, which requires only the rightmost second-operand byte.

<u>Condition</u> <u>Code</u>: The code remains unchanged.

<u>Program</u> <u>Exceptions</u>:

Access (fetch, operand 2; store, operand 1)

Programming Notes

1. An example of the use of the PACK instruction is given in Appendix A.

2. PACK may be used to interchange the two hexadecimal digits in one byte by specifying a zero in the L_1 and L_2 fields and the same address for both operands.

3. To remove the zone bits of all bytes of a field, including the rightmost byte, both operands must be extended on the right with a dummy byte, which subsequently is ignored in the result field.

4. The storage-operand references for PACK may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

SET PROGRAM MASK

SPM R_1 [RR]

'04'	R_1	////

0 8 12 15

The first operand is used to set the condition code and the program mask of the current PSW.

Bits 12-15 of the instruction are ignored.

Bits 2 and 3 of general register R_1 replace the condition code, and bits 4-7 replace the program mask. Bits 0, 1, and 8-31 of general register R_1 are ignored.

Condition Code: The code is set as specified by bits 2 and 3 of general register R_1.

Program Exceptions: None.

Programming Notes

1. Bits 2-7 of the general register may have been loaded from the PSW by BRANCH AND LINK.

2. SET PROGRAM MASK permits setting of the condition code and the mask bits in either the problem state or the supervisor state.

3. The program should take into consideration that the setting of the program mask can have a significant effect on subsequent execution of the program. Not only do the four mask bits control whether the corresponding interruptions occur, but the exponent-underflow and significance masks also determine the result which is obtained.

SHIFT LEFT DOUBLE

SLDA $R_1,D_2(B_2)$ [RS]

'8F'	R_1	////	B_2	D_2

0 8 12 16 20 31

The 63-bit numeric part of the signed first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

Bits 12-15 of the instruction are ignored.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 64-bit signed binary integer. The sign position of the even-numbered register remains unchanged. The leftmost bit position of the odd-numbered register contains a numeric bit, which participates in the shift in the same manner as the other numeric bits. Zeros are supplied to the vacated bit positions on the right.

If one or more bits unlike the sign bit are shifted out of bit position 1 of the even-numbered register, an overflow occurs, and condition code 3 is set. If the fixed-point-overflow mask bit is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

0	Result zero; no overflow
1	Result less than zero; no overflow
2	Result greater than zero; no overflow
3	Overflow

Program Exceptions:

Fixed-point overflow
Specification

Programming Notes

1. An example of the use of the SHIFT LEFT DOUBLE instruction is given in Appendix A.

2. The eight shift instructions provide the following three pairs of alternatives: left or right, single or double, and signed or logical. The signed shifts differ from the logical shifts in that, in the signed shifts, overflow is recognized, the condition code is set, and the leftmost bit participates as a sign.

3. A zero shift amount in the two signed double-shift operations provides a double-length sign and magnitude test.

4. The base register participating in the generation of the second-operand address permits indirect specification of the shift amount. A zero in the B_2 field indicates the absence of indirect shift specification.

SHIFT LEFT DOUBLE LOGICAL

SLDL $R_1,D_2(B_2)$ [RS]

'8D'	R_1	////	B_2	D_2
0	8	12	16	20 31

The 64-bit first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

Bits 12-15 of the instruction are ignored.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 0 of the even-numbered register are not inspected and are lost. Zeros are supplied to the vacated bit positions on the right.

Condition Code: The code remains unchanged.

Program Exceptions:

Specification

SHIFT LEFT SINGLE

SLA $R_1,D_2(B_2)$ [RS]

'8B'	R_1	////	B_2	D_2
0	8	12	16	20 31

The 31-bit numeric part of the signed first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

Bits 12-15 of the instruction are ignored.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 32-bit signed binary integer. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the left shift. Zeros are supplied to the vacated bit positions on the right.

If one or more bits unlike the sign bit are shifted out of bit position 1, an overflow occurs, and condition code 3 is set. If the fixed-point-overflow mask bit is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

Fixed-point overflow

Programming Notes

1. An example of the use of the SHIFT LEFT SINGLE instruction is given in Appendix A.

2. For numbers with a value greater than or equal to -2^{30} and less than 2^{30}, a left shift of one bit position is equivalent to multiplying the number by 2.

3. Shift amounts from 31 to 63 cause the entire numeric part to be shifted out of the register, leaving a result of the maximum negative number or zero, depending on whether or not the initial contents were negative.

SHIFT LEFT SINGLE LOGICAL

SLL $R_1,D_2(B_2)$ [RS]

'89'	R_1	////	B_2	D_2

0 8 12 16 20 31

The 32-bit first operand is shifted left the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

Bits 12-15 of the instruction are ignored.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 0 are not inspected and are lost. Zeros are supplied to the vacated bit positions on the right.

<u>Condition Code</u>: The code remains unchanged.

<u>Program Exceptions</u>: 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 63-bit numeric part of the signed first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

Bits 12-15 of the instruction are ignored.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 64-bit signed binary integer. The sign position of the even-numbered register remains unchanged. The leftmost bit position of the odd-numbered register contains a numeric bit, which participates in the shift in the same manner as the other numeric bits. Bits shifted out of bit position 31 of the odd-numbered register are not inspected and are lost. Bits equal to the sign are supplied to the vacated bit positions on the left.

<u>Resulting Condition Code</u>:

0	Result zero
1	Result less than zero
2	Result greater than zero
3	--

<u>Program Exceptions</u>:

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 64-bit first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

Bits 12-15 of the instruction are ignored.

The R_1 field designates an even-odd pair of general registers and must designate an even-numbered register; otherwise, a specification exception is recognized.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 64 bits of the first operand participate in the shift. Bits shifted out of bit position 31 of the odd-numbered register are not inspected and are lost. Zeros are supplied to the vacated bit positions on the left.

<u>Condition Code</u>: The code remains unchanged.

<u>Program Exceptions</u>:

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 31-bit numeric part of the signed first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

Bits 12-15 of the instruction are ignored.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

The first operand is treated as a 32-bit signed binary integer. The sign of the first operand remains unchanged. All 31 numeric bits of the operand participate in the right shift. Bits shifted out of bit position 31 are not inspected and are lost. Bits equal to the sign are supplied to the vacated bit positions on the left.

Resulting Condition Code:

0 Result zero
1 Result less than zero
2 Result greater than zero
3 --

Program Exceptions: None.

Programming Notes

1. A right shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the result is equivalent to dividing the number by 2. When an odd number is shifted right one position, the result is equivalent to dividing the next lower number by 2. For example, +5 shifted right by one bit position yields +2, whereas -5 yields -3.

2. Shift amounts from 31 to 63 cause the entire numeric part to be shifted out of the register, leaving a result of -1 or zero, depending on whether or not the initial contents were negative.

SHIFT RIGHT SINGLE LOGICAL

SRL $R_1,D_2(B_2)$ [RS]

'88'	R_1	////	B_2	D_2
0	8	12	16	20 31

The 32-bit first operand is shifted right the number of bits specified by the second-operand address, and the result is placed at the first-operand location.

Bits 12-15 of the instruction are ignored.

The second-operand address is not used to address data; its rightmost six bits indicate the number of bit positions to be shifted. The remainder of the address is ignored.

All 32 bits of the first operand participate in the shift. Bits shifted out of bit position 31 are not inspected and are lost. Zeros are supplied to the vacated bit positions on the left.

Condition Code: The code remains unchanged.

Program Exceptions: None.

STORE

ST $R_1,D_2(X_2,B_2)$ [RX]

'50'	R_1	X_2	B_2	D_2
0	8	12	16	20 31

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]

'42'	R_1	X_2	B_2	D_2
0	8	12	16	20 31

Bits 24-31 of general register R_1 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]

'BE'	R_1	M_3	B_2	D_2

0 8 12 16 20 31

Bytes selected from general register R_1 under control of a mask are placed at contiguous byte locations beginning at the second-operand address.

The contents of the M_3 field are used as a mask. These four bits, left to right, correspond one for one with the four bytes, left to right, of general register R_1. The bytes corresponding to ones in the mask are placed in the same order at successive and contiguous storage locations beginning at the second-operand address. When the mask is not zero, the length of the second operand is equal to the number of ones in the mask. The contents of the general register remain unchanged.

When the mask is not zero, exceptions associated with storage-operand accesses are recognized only for the number of bytes specified by the mask.

When the mask is zero, the single byte designated by the second-operand address remains unchanged; however, on some models, the value may be fetched and subsequently stored back unchanged at the same storage location. This update appears to be an interlocked-update reference as observed by other CPUs.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

Programming Notes

1. An example of the use of the STORE CHARACTERS UNDER MASK instruction is given in Appendix A.

2. STORE CHARACTERS UNDER MASK with a mask of 0111 may be used to store a three-byte address, for example, in modifying the address in a CCW.

3. STORE CHARACTERS UNDER MASK with a mask of 1111, 0011, or 0001 performs the same function as STORE, STORE HALFWORD, or STORE CHARACTER, respectively. However, on most models, the performance of STORE CHARACTERS UNDER MASK is slower.

4. Using STORE CHARACTERS UNDER MASK with a zero mask should be avoided since this instruction, depending on the model, may perform a fetch and store of the single byte designated by the second-operand address. This reference is not interlocked against accesses by channels. In addition, it may cause any of the following to occur for the byte designated by the second-operand address: a PER storage-alteration event may be recognized; access exceptions may be recognized; and, provided no access exceptions exist, the change bit may be set to one.

STORE CLOCK

STCK $D_2(B_2)$ [S]

'B205'	B_2	D_2

0 16 20 31

The current value of the TOD 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.

Zeros are stored for the rightmost 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 storage.

Resulting Condition Code:

0 Clock in set state
1 Clock in not-set state
2 Clock in error state

3 Clock in stopped state or not-operational state

Program Exceptions:

Access (store, operand 2)

Programming Notes

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

2. Condition code 0 normally indicates that the clock has been set by the control program. Accordingly, the value may be used in elapsed-time measurements and as a valid time-of-day and calendar indication. Condition code 1 indicates that the clock value is the elapsed time since the power for the clock was turned on. In this case, the value may be used in elapsed-time measurements but is not a valid time-of-day indication. Condition codes 2 and 3 mean that the value provided by STORE CLOCK cannot be used for time measurement or indication.

3. Condition code 3 indicates that the clock is in either the stopped state or the not-operational state. These two states can normally be distinguished because an all-zero value is stored when the clock is in the not-operational state.

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

Bits 16-31 of general register R_1 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 contents of the set of general registers starting with general register R_1 and ending with general register R_3 are placed in the storage area beginning at the location designated by the second-operand address and continuing through as many locations as needed.

The general registers are stored in the ascending order of register numbers, starting with general register R_1 and continuing up to and including general register R_3, with general register 0 following general register 15.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)

Programming Note

An example of the use of the STORE MULTIPLE instruction is given in Appendix A.

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 at the first-operand location. The operands and the difference are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

Access (fetch, operand 2 of S only)
Fixed-point overflow

Programming Notes

1. When, in the RR format, R_1 and R_2 designate the same register, subtracting is equivalent to clearing the register.

2. Subtracting a maximum negative number from another maximum negative number gives a zero result and no overflow.

SUBTRACT HALFWORD

SH $R_1,D_2(X_2,B_2)$ [RX]

'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 at the first-operand location. The second operand is two bytes in length and is treated as a 16-bit signed binary integer. The first operand and the difference are treated as 32-bit signed binary integers.

When there is an overflow, the result is obtained by allowing any carry into the sign-bit position and ignoring any carry out of the sign-bit position, and condition code 3 is set. If the fixed-point-overflow mask is one, a program interruption for fixed-point overflow occurs.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

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 at the first-operand location. The operands and the difference are treated as 32-bit unsigned binary integers.

Resulting Condition Code:

0 --
1 Result not zero; no carry
2 Result zero; carry
3 Result not zero; carry

Program Exceptions:

Access (fetch, operand 2 of SL only)

Programming Notes

1. Logical subtraction is performed by adding the one's complement of the second operand and a value of one to the first operand. The use of the one's complement and the value of one instead of the two's complement of the second operand results in a carry when the second operand is zero.

2. SUBTRACT LOGICAL differs from SUBTRACT only in the meaning of the condition code and in the absence of the interruption for overflow.

3. A zero difference is always accompanied by a carry out of bit position 0.

4. The condition-code setting for SUBTRACT LOGICAL can also be interpreted as indicating the presence and absence of a borrow, as follows:

1 Result not zero; borrow
2 Result zero; no borrow
3 Result not zero; no borrow

SUPERVISOR CALL

SVC I [RR]

'0A'	I	
0	8	15

The instruction causes a supervisor-call interruption, with the I field of the instruction providing the rightmost byte of the interruption code.

Bits 8-15 of the instruction, with eight zeros appended on the left, are placed in the supervisor-call interruption code that is stored in the course of the interruption. See "Supervisor-Call Interruption" in Chapter 6, "Interruptions."

A serialization and checkpoint-synchronization function is performed.

<u>Condition Code</u>: The code remains unchanged and is saved as part of the old PSW. A new condition code is loaded as part of the supervisor-call interruption.

<u>Program Exceptions</u>: None.

TEST AND SET

TS $D_2(B_2)$ [S]

'93'	/////////	B_2	D_2	
0	8	16	20	31

The leftmost bit (bit position 0) of the byte located at the second-operand address is used to set the condition code, and then the byte is set to all ones.

Bits 8-15 of the instruction are ignored.

The byte in storage is set to all ones as it is fetched for the testing of bit position 0. This update appears to be an interlocked-update reference as observed by other CPUs.

A serialization function is performed before the byte is fetched and again after the storing of all ones.

<u>Resulting Condition Code</u>:

0 Leftmost bit zero
1 Leftmost bit one
2 --
3 --

<u>Program Exceptions</u>:

Access (fetch and store, operand 2)

<u>Programming Notes</u>

1. TEST AND SET may be used for controlled sharing of a common storage area by programs operating on different CPUs. This instruction is provided primarily for compatibility with programs written for System/360. The instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP provide functions which are more suitable for sharing among programs on a single CPU or for programs that may be interrupted. See the description of these instructions and the associated programming notes for details.

2. TEST AND SET does not interlock against storage accesses by channels. Therefore, the instruction should not be used to update a location into which a channel program may store, since the channel-program data may be lost.

TEST UNDER MASK

TM $D_1(B_1),I_2$ [SI]

'91'	I_2	B_1	D_1	
0	8	16	20	31

A mask is used to select bits of the first operand, and the result is indicated in the condition code.

The byte of immediate data, I_2, is used as an eight-bit mask. The bits of the mask are made to correspond one for one with the bits of the byte in storage designated by the first-operand address.

A mask bit of one indicates that the storage bit is to be tested. When the mask bit is zero, the storage bit is ignored. When all storage bits thus selected are zero, condition code 0 is set. Condition code 0 is also set when the mask is all zeros. When the selected bits are all ones, condition code 3 is set; otherwise, condition code 1 is set.

Access exceptions associated with the storage operand are recognized for one byte even when the mask is all zeros.

<u>Resulting Condition Code</u>:

0 Selected bits all zeros; or mask bits all zeros

1 Selected bits mixed zeros and ones
2 --
3 Selected bits all ones

Program Exceptions:

Access (fetch, operand 1)

Programming Note

An example of the use of the TEST UNDER MASK instruction is given in Appendix A.

TRANSLATE

TR $D_1(L,B_1),D_2(B_2)$ [SS]

```
┌─────────┬─────────┬─────┬─────/┬─────┬─────/┐
│  'DC'   │    L    │ B₁  │  D₁  │ B₂  │  D₂  │
└─────────┴─────────┴─────┴─────/┴─────┴─────/┘
0         8         16    20     32    36    47
```

The bytes of the first operand are used as eight-bit arguments to reference a list designated by the second-operand address. Each function byte selected from the list replaces the corresponding argument in the first operand.

The L field specifies the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding left to right. Each argument byte is added to the initial second-operand address. The addition is performed following the rules for address arithmetic, with the argument byte treated as an eight-bit unsigned binary integer and extended with zeros on the left. The sum is used as the address of the function byte, which then replaces the original argument byte.

The operation proceeds until the first-operand field is exhausted. The list is not altered unless an overlap occurs.

When the operands overlap, the result is obtained as if each result byte were stored immediately after fetching the corresponding function byte.

Access exceptions are recognized only for those bytes in the second operand which are actually required.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)

Programming Notes

1. An example of the use of the TRANSLATE instruction is given in Appendix A.

2. TRANSLATE may be used to convert data from one code to another code.

3. The instruction may also be used to rearrange data. This may be accomplished by placing a pattern in the destination area, by designating the pattern as the first operand of TRANSLATE, and by designating the data that is to be rearranged as the second operand. Each byte of the pattern contains an eight-bit number specifying the byte destined for this position. Thus, when the instruction is executed, the pattern selects the bytes of the second operand in the desired order.

4. Because each eight-bit argument byte is added to the initial second-operand address to obtain the address of a function byte, the list may contain 256 bytes. In cases where it is known that not all eight-bit argument values will occur, it is possible to reduce the size of the list.

5. Significant performance degradation is possible when, with DAT on, the second-operand address of TRANSLATE designates a location that is less than 256 bytes to the left of a 2K-byte boundary. This is because the machine may perform a trial execution of the instruction to determine if the second operand actually crosses the boundary.

6. The fetch and subsequent store accesses to a particular byte in the first-operand field do not necessarily occur one immediately after the other. Thus, this instruction cannot be safely used to update a location in storage if the possibility exists that another CPU or a channel may also be updating the location. An example of this effect is shown for OR (OI) in the section "Multiprogramming and Multiprocessing Examples" in Appendix A.

7. The storage-operand references of TRANSLATE may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

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 bytes of the first operand are used as eight-bit arguments to select function bytes from a list designated by the second-operand address. The first nonzero function byte is inserted in general register 2, and the related argument address in general register 1.

The L field specifies the length of only the first operand.

The bytes of the first operand are selected one by one for translation, proceeding from left to right. The first operand remains unchanged in storage. Calculation of the address of the function byte is performed as in the TRANSLATE instruction. The function byte retrieved from the list is inspected for a value of zero.

When the function byte is zero, the operation proceeds with the next byte of the first operand. When the first-operand field is exhausted before a nonzero function byte is encountered, the operation is completed by setting condition code 0. The contents of general registers 1 and 2 remain unchanged.

When the function byte is nonzero, the operation is completed by inserting the function byte in general register 2 and the related argument address in general register 1. This address points to the argument byte last translated. The function byte replaces bits 24-31 of general register 2. The address replaces bits 8-31 of general register 1. Bits 0-7 of general register 1 and bits 0-23 of general register 2 remain unchanged.

When the function byte is nonzero, either condition code 1 or 2 is set, depending on whether the argument byte is the rightmost byte of the first operand. Condition code 1 is set if one or more argument bytes remain to be translated. Condition code 2 is set if no more argument bytes remain.

Access exceptions are recognized only for those bytes in the second operand which are actually required. Access exceptions are not recognized for those bytes in the first operand which are to the right of the first byte for which a nonzero function byte is obtained.

<u>Resulting Condition Code</u>:

0 All function bytes zero
1 Nonzero function byte; first-operand field not exhausted
2 Nonzero function byte; first-operand field exhausted
3 --

<u>Program Exceptions</u>:

Access (fetch, operands 1 and 2)

<u>Programming Notes</u>

1. An example of the use of the TRANSLATE AND TEST instruction is given in Appendix A.

2. TRANSLATE AND TEST may be used to scan the first operand for characters with special meaning. The second operand, or list, is set up with all-zero function bytes for those characters to be skipped over and with nonzero function bytes for the characters to be detected.

UNPACK

UNPK $D_1(L_1,B_1),D_2(L_2,B_2)$ [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 at the first-operand location. The packed and zoned formats are described in Chapter 8, "Decimal Instructions."

The second operand is treated as though it had the packed format. Its digits and sign are placed unchanged in the first-operand location, using the zoned format. Zone bits with coding of 1111 are supplied for all bytes except the rightmost byte, the zone of which receives the sign of the second operand. The sign and digits are not checked for valid codes.

The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first-operand field is too short to contain all digits of the second operand, the remaining leftmost portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated.

When the operands overlap, the result is obtained as if the operands were processed one byte at a time and as if the first result byte were stored immediately after fetching the first operand byte. The entire rightmost second-operand byte is used in forming the first result byte. For the remainder of the field, information for two result bytes is obtained from a single second-operand byte, and execution proceeds as if the leftmost four bits of the byte were to remain available for the next result byte and need not be refetched. Thus, the result is as if two result bytes were to be stored immediately after fetching a single operand byte.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; store, operand 1)

Programming Notes

1. An example of the use of the UNPACK instruction is given in Appendix A.

2. A field that is to be unpacked can be destroyed by improper overlapping. To save storage space for unpacking by overlapping the operands, the rightmost byte of the first operand must be to the right of the rightmost byte of the second operand by the number of bytes in the second operand minus 2. If only one or two bytes are to be unpacked, the rightmost bytes of the two operands may coincide.

3. The storage-operand references of UNPACK may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

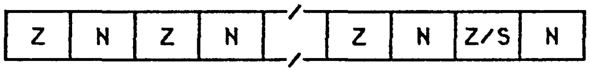
CHAPTER 8. DECIMAL INSTRUCTIONS

The decimal instructions of this chapter perform arithmetic and editing operations on decimal data. Additional operations on decimal data are provided by several of the instructions in Chapter 7, "General Instructions." Decimal operands always reside in storage, and all decimal instructions use the SS instruction format. Decimal operands occupy storage fields that can start on any byte boundary.

DECIMAL-NUMBER FORMATS

Decimal numbers may be represented in either the zoned or packed format. Both decimal-number formats are of variable length; the instructions used to operate on decimal data each specify the length of their operands and results. Each byte of either format consists of a pair of four-bit codes; the four-bit codes include decimal-digit codes, sign codes, and a zone code.

ZONED FORMAT

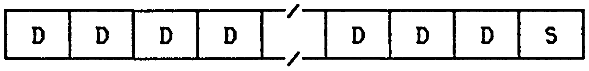


In the zoned format, the rightmost four bits of a byte are called the numeric bits (N) and normally consist of a code representing a decimal digit. The leftmost four bits of a byte are called the zone bits (Z), except for the rightmost byte of a decimal operand, where these bits may be treated either as a zone or as a sign (S).

Decimal digits in the zoned format may be part of a larger character set, which includes also alphabetic and special characters. The zoned format is, therefore, suitable for input, editing, and output of numeric data in human-readable form. There are no decimal-arithmetic instructions which operate directly on decimal numbers in the zoned format; such numbers must first be converted to the packed format.

The editing instructions produce a result of up to 256 bytes; each byte may be a decimal digit in the zoned format, a message byte, or a fill byte.

PACKED 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. Decimal arithmetic is performed with operands in the packed format and generates results in the packed format.

The packed-format operands and results of decimal-arithmetic instructions may be up to 16 bytes (31 digits and sign), except that the maximum length of a multiplier or divisor is eight bytes (15 digits and sign). In division, the sum of the lengths of the quotient and remainder may be from two to 16 bytes. The editing instructions can fetch as many as 256 decimal digits from one or more decimal numbers of variable length, each in the packed format.

DECIMAL CODES

The decimal digits 0-9 have the binary encoding 0000-1001.

The preferred sign codes are 1100 for plus and 1101 for minus. These are the sign codes generated for the results of the decimal-arithmetic instructions and the CONVERT TO DECIMAL instruction.

Alternate sign codes are also recognized as valid in the sign position: 1010, 1110, and 1111 are alternate codes for plus, and 1011 is an alternate code for minus. Alternate sign codes are accepted for any decimal source operand, but are not generated in the completed result of a decimal-arithmetic instruction or CONVERT TO DECIMAL. This is true even when an operand remains otherwise unchanged, such as when adding zero to a number. An alternate sign code is, however, left unchanged by MOVE NUMERICS, MOVE WITH OFFSET, MOVE ZONES, PACK, and UNPACK.

When an invalid sign or digit code is detected, a data exception is recognized. For the decimal-arithmetic instructions and CONVERT TO BINARY, the action taken for a data exception depends on whether a sign code is invalid. When a sign code is invalid, the operation is suppressed regardless of whether any other condition causing a data exception exists. When an invalid digit code is detected but no sign code is invalid, the operation is terminated.

For the editing instructions EDIT and EDIT AND MARK, an invalid sign code is not recognized. The operation is terminated for a data exception due to an invalid digit code. No validity checking is performed by MOVE NUMERICS, MOVE WITH OFFSET, MOVE ZONES, PACK, and UNPACK.

The zone code 1111 is generated in the left four bit positions of each byte representing a zone and a decimal digit in zoned-format results. Zoned-format results are produced by EDIT, EDIT AND MARK, and UNPACK. For EDIT and EDIT AND MARK, each result byte representing a zoned-format decimal digit contains the zone code 1111 in the left four bit positions and the decimal-digit code in the right four bit positions. For UNPACK, zone bits with a coding of 1111 are supplied for all bytes except the rightmost byte, the zone of which receives the sign.

The meaning of the decimal codes is summarized in the figure "Summary of Digit and Sign Codes."

Programming Note

Since 1111 is both the zone code and an alternate code for plus, unsigned (positive) decimal numbers may be represented in the zoned format with 1111 zone codes in all byte positions. The result of the PACK instruction converting such a number to the packed format may be used directly as an operand for decimal instructions.

	Recognized As	
Code	Digit	Sign
0000	0	Invalid
0001	1	Invalid
0010	2	Invalid
0011	3	Invalid
0100	4	Invalid
0101	5	Invalid
0110	6	Invalid
0111	7	Invalid
1000	8	Invalid
1001	9	Invalid
1010	Invalid	Plus
1011	Invalid	Minus
1100	Invalid	Plus (preferred)
1101	Invalid	Minus (preferred)
1110	Invalid	Plus
1111	Invalid	Plus (zone)

Summary of Digit and Sign Codes

DECIMAL OPERATIONS

The decimal instructions in this chapter consist of two classes, the decimal-arithmetic instructions and the editing instructions.

DECIMAL-ARITHMETIC INSTRUCTIONS

The decimal-arithmetic instructions perform addition, subtraction, multiplication, division, comparison, and shifting.

Operands of the decimal-arithmetic instructions are in the packed format and are treated as signed decimal integers. A decimal integer is represented in true form as an absolute value with a separate plus or minus sign. It contains an odd number of decimal digits, from one to 31, and the sign; this corresponds to an operand length of one to 16 bytes.

A decimal zero normally has a plus sign, but multiplication, division, and overflow may produce a zero value with a minus sign. Such a negative zero is a valid operand and is treated as equal to a positive zero by COMPARE DECIMAL.

The lengths of the two operands specified in the instruction need not be the same. If necessary, the shorter operand is considered to be extended with zeros on the left. Results, however, cannot exceed the first-operand length as specified in the instruction.

When a carry or leftmost nonzero digits of the result are lost because the first-operand field is too short, the result is obtained by ignoring the overflow digits, condition code 3 is set, and, if the decimal-overflow mask bit is one, a program interruption for decimal overflow occurs. The operand lengths alone are not an indication of overflow; nonzero digits must have been lost during the operation.

The operands of decimal-arithmetic instructions should not overlap at all or should have coincident rightmost bytes. In ZERO AND ADD, the operands may also overlap in such a manner that the rightmost byte of the first operand (which becomes the result) is to the right of the rightmost byte of the second operand. For these cases of proper overlap, the result is obtained as if operands were processed right to left. Because the codes for digits and signs are verified during the performance of the arithmetic, improperly overlapping operands are recognized as data exceptions.

Programming Note

A packed decimal number in storage may be designated as both the first and second operand of ADD DECIMAL, COMPARE DECIMAL, DIVIDE DECIMAL, MULTIPLY DECIMAL, SUBTRACT DECIMAL, or ZERO AND ADD. Thus, a decimal number may be added to itself, compared with itself, and so forth; SUBTRACT DECIMAL may be used to set a decimal field in storage to zero, and, for MULTIPLY DECIMAL, a decimal number may be squared in place.

EDITING INSTRUCTIONS

The editing instructions are EDIT and EDIT AND MARK. For these instructions, only the first operand (the pattern) has an explicitly specified length. The second operand (the source) is considered to have as many digits as necessary for the completion of the operation.

Overlapping operands for the editing instructions yield unpredictable results.

EXECUTION OF DECIMAL INSTRUCTIONS

During the execution of a decimal instruction, all bytes of the operands are not necessarily accessed concurrently, and the fetch and store accesses to a single location do not necessarily occur one immediately after the other. Furthermore, for decimal instructions, data in source fields may be accessed more than once, and intermediate values may be placed in the result field that may differ from the original operand and final result values. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.") Thus, in a multiprocessing configuration, an instruction such as ADD DECIMAL cannot be safely used to update a shared storage location when the possibility exists that another CPU may also be updating that location.

OTHER INSTRUCTIONS FOR DECIMAL OPERANDS

In addition to the decimal instructions in this chapter, MOVE NUMERICS and MOVE ZONES are provided for operating on data of lengths up to 256 bytes in the zoned format. Two instructions are provided for converting data between the zoned and packed formats: PACK transforms zoned data of lengths up to 16 bytes into packed data, and UNPACK performs the reverse transformation. MOVE WITH OFFSET can operate on packed data of lengths up to 16 bytes. Two instructions are provided for conversion between the packed-decimal and signed-binary-integer formats. CONVERT TO BINARY converts packed decimal to binary, and CONVERT TO DECIMAL converts binary to packed decimal; the length of the packed decimal operand of these instructions is eight bytes (15 digits and sign). These seven instructions are not considered to be decimal instructions and are described in Chapter 7, "General Instructions." The editing instructions in this chapter may also be used to change data from the packed to the zoned format.

INSTRUCTIONS

The decimal instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of Decimal Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For ADD DECIMAL, for example, AP is the mnemonic and $D_1(L_1,B_1),D_2(L_2,B_2)$ the operand designation.

Name	Mnemonic	Characteristics										Op Code
ADD DECIMAL	AP	SS	C	A		D	DF				ST	FA
COMPARE DECIMAL	CP	SS	C	A		D						F9
DIVIDE DECIMAL	DP	SS		A	SP	D		DK			ST	FD
EDIT	ED	SS	C	A		D					ST	DE
EDIT AND MARK	EDMK	SS	C	A		D			G1	R	ST	DF
MULTIPLY DECIMAL	MP	SS		A	SP	D					ST	FC
SHIFT AND ROUND DECIMAL	SRP	SS	C	A		D	DF				ST	F0
SUBTRACT DECIMAL	SP	SS	C	A		D	DF				ST	FB
ZERO AND ADD	ZAP	SS	C	A		D	DF				ST	F8

Explanation:

A	Access exceptions for logical addresses.
C	Condition code is set.
D	Data exception.
DF	Decimal-overflow exception.
DK	Decimal-divide exception.
G1	Instruction execution includes the implied use of general register 1.
R	PER general-register-alteration event.
SP	Specification exception.
SS	SS instruction format.
ST	PER storage-alteration event.

Summary of Decimal Instructions

ADD DECIMAL

AP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

'FA'	L_1	L_2	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The second operand is added to the first operand, and the resulting sum is placed at the first-operand location. The operands and result are in the packed format.

Addition is algebraic, taking into account the signs and all digits of both operands. All sign and digit codes are checked for validity.

If the first operand is too short to contain all leftmost nonzero digits of the sum, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs.

The sign of the sum is determined by the rules of algebra. In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, a zero result is given either a positive or negative sign, as determined by what the sign of the correct sum would have been.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal overflow

Programming Note

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

COMPARE DECIMAL

CP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

'F9'	L_1	L_2	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The first operand is compared with the second operand, and the result is indicated in the condition code. The operands are in the packed format.

Comparison is algebraic and follows the procedure for decimal subtraction, except that both operands remain unchanged. When the difference is zero, the operands are equal. When a nonzero difference is positive or negative, the first operand is high or low, respectively.

Overflow cannot occur because the difference is discarded.

All sign and digit codes are checked for validity.

Resulting Condition Code:

0 Operands equal
1 First operand low
2 First operand high
3 --

Program Exceptions:

Access (fetch, operands 1 and 2)
Data

Programming Notes

1. An example of the use of the COMPARE DECIMAL instruction is given in Appendix A.
2. The preferred and alternate sign codes for a particular sign are treated as equivalent for comparison purposes.
3. A negative zero and a positive zero compare equal.

DIVIDE DECIMAL

DP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

'FD'	L_1	L_2	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The first operand (the dividend) is divided by the second operand (the divisor). The resulting quotient and remainder are placed at the first-operand location. The operands and results are in the packed format.

The quotient is placed leftmost in the first-operand location. The number of bytes in the quotient field is equal to the difference between the dividend and divisor lengths ($L_1 - L_2$). The remainder is placed rightmost in the first-operand location and has a length equal to the divisor length. Together, the quotient and remainder fields occupy the entire first operand; therefore, the address of the quotient is the address of the first operand.

The divisor length cannot exceed 15 digits and sign (L_2 not greater than seven) and must be less than the dividend length (L_2 less than L_1); otherwise, a specification exception is recognized.

The dividend, divisor, quotient, and remainder are each signed decimal integers in the packed format and are right-aligned in their fields. All sign and digit codes of the dividend and divisor are checked for validity.

The sign of the quotient is determined by the rules of algebra from the dividend and divisor signs. The sign of the remainder has the same value as the dividend sign. These rules hold even when the quotient or remainder is zero.

Overflow cannot occur. If the divisor is zero or the quotient is too large to be represented by the number of digits specified, a decimal-divide exception is recognized. This includes the case of division of zero by zero. The decimal-divide exception is indicated only if the sign codes of both the dividend and divisor are valid, and only if the digit or digits used in establishing the exception are valid.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal divide
Specification

Programming Notes

1. An example of the use of the DIVIDE DECIMAL instruction is given in Appendix A.

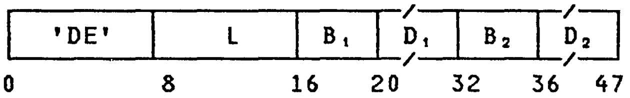
2. The dividend cannot exceed 31 digits and sign. Since the remainder cannot be shorter than one digit and sign, the quotient cannot exceed 29 digits and sign.

3. The condition for a decimal-divide exception can be determined by a trial comparison. The leftmost digit of the divisor is aligned one digit to the right of the leftmost dividend digit. When the divisor, so aligned, is less than or equal to the dividend, ignoring signs, a divide exception is indicated.

4. If a data exception does not exist, a decimal-divide exception occurs when the leftmost dividend digit is not zero.

EDIT

ED $D_1(L,B_1),D_2(B_2)$ [SS]



The second operand (the source), which normally contains one or more decimal numbers in the packed format, is changed to the zoned format and modified under the control of the first operand (the pattern). The edited result replaces the first operand.

The length field specifies the length of the first operand, which may contain bytes of any value.

The length of the source is determined by the operation according to the contents of the pattern. The source normally consists of one or more decimal numbers, each in the packed format. The leftmost four bits of each source byte must specify a decimal-digit code (0000-1001); a sign code (1010-1111) is recognized as a data exception. The rightmost four bits may specify either a sign code or a decimal-digit code. Access and data exceptions are recognized only for those bytes in the second operand which are actually required.

The result is obtained as if both operands were processed left to right one byte at a time. Overlapping pattern and source fields give unpredictable results.

During the editing process, each byte of the pattern is affected in one of three ways:

1. It is left unchanged.

2. It is replaced by a source digit expanded to the zoned format.

3. It is replaced by the first byte in the pattern, called the fill byte.

Which of the three actions takes place is determined by one or more of the following: the type of the pattern byte, the state of the significance indicator, and whether the source digit examined is zero.

Pattern Bytes: There are four types of pattern bytes: digit selector, significance starter, field separator, and message byte. Their coding is as follows:

Name	Code
Digit selector	0010 0000
Significance starter	0010 0001
Field separator	0010 0010
Message byte	Any other

The detection of either a digit selector or a significance starter in the pattern causes an examination to be made of the significance indicator and of a source digit. As a result, either the expanded source digit or the fill byte, as appropriate, is selected to replace the pattern byte. Additionally, encountering a digit selector or a significance starter may cause the significance indicator to be changed.

The field separator identifies individual fields in a multiple-field editing operation. It is always replaced in the result by the fill byte, and the significance indicator is always off after the field separator is encountered.

Message bytes in the pattern are either replaced by the fill byte or remain unchanged in the result, depending on the state of the significance indicator. They may thus be used for padding, punctuation, or text in the significant portion of a field or for the insertion of sign-dependent symbols.

Fill Byte: The first byte of the pattern is used as the fill byte. The fill byte can have any code and may concurrently specify a control function. If this byte is a digit selector or significance starter, the indicated editing action is taken after the code has been assigned to the fill byte.

Source Digits: Each time a digit selector or significance starter is encountered in the pattern, a new source digit is examined for placement in the pattern field. Either the source digit is disregarded, or it is expanded to the zoned format, by appending the zone code 1111 on the left, and stored in place of the pattern byte.

Execution is as if the source digits were selected one byte at a time and as if a source byte were 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 byte that calls for a digit examination. When the leftmost four bits contain an invalid digit code, a data exception is recognized, and the operation is terminated.

At the time the left digit of a source byte is examined, the rightmost four bits are checked for the existence of a sign code. When a sign code is encountered in the rightmost four bit positions, these bits are not treated as a decimal-digit code, and a new source byte is fetched from storage when the next pattern byte calls for a source-digit examination.

When the pattern contains no digit selector or significance starter, no source bytes are fetched and examined.

Significance Indicator: The significance indicator is turned on or off to indicate the significance or nonsignificance, respectively, of subsequent source digits or message bytes. Significant source digits replace their corresponding digit selectors or significance starters in the result. Significant message bytes remain unchanged in the result.

The significance indicator, by its on or off state, indicates also the negative or positive value, respectively, of a completed source field and is used as one factor in the setting of the condition code.

The significance indicator is set to off at the start of the editing operation, after a field separator is encountered, or after a source byte is examined that has a plus code in the rightmost four bit positions.

The significance indicator is set to on when a significance starter is encountered whose source digit is a valid decimal digit, or when a digit selector is encountered whose source digit is a nonzero decimal digit, provided that in both instances the source byte does not have a plus code in the rightmost four bit positions.

In all other situations, the significance indicator is not changed. A minus sign code has no effect on the significance indicator.

Result Bytes: The result of an editing operation replaces and is equal in length to the pattern. It is composed of pattern bytes, fill bytes, and zoned source digits.

If the pattern byte is a message byte and the significance indicator is on, the message byte remains unchanged in the result. If the pattern byte is a field separator or if the significance indicator is off when a message byte is encountered in the pattern, the fill byte replaces the pattern byte in the result.

If the digit selector or significance starter is encountered in the pattern with the significance indicator off and the source digit zero, the source digit is considered nonsignificant, and the fill byte replaces the pattern byte. If the digit selector or significance starter is encountered with either the significance indicator on or with a nonzero decimal source digit, the source digit is considered significant, is changed to the zoned format, and replaces the pattern byte in the result.

Condition Code: The sign and magnitude of the last field edited are used to set the condition code. The term "last field" refers to those source digits, if any, in the second operand selected by digit selectors or significance starters after the last field separator; if the pattern contains no field separator, there is only one field, which is considered to be the last field. If no such source digits are selected, the last field is considered to be of zero length.

Condition code 0 is set when the last field edited is zero or of zero length.

Condition code 1 is set when the last field edited is nonzero and the significance indicator is on. (This indicates a result less than zero if the last source byte examined contained a sign code in the rightmost four bits.)

Condition code 2 is set when the last field edited is nonzero and the significance indicator is off. (This indicates a result greater than zero if the last source byte examined contained a sign code in the rightmost four bits.)

The figure "Summary of Editing Functions" summarizes the functions of the EDIT and EDIT AND MARK operations. The leftmost four columns list all the significant combinations of the four conditions that can be encountered in the execution of an editing operation. The rightmost two columns list the action taken for each case -- the type of byte placed in the result field and the new setting of the significance indicator.

Resulting Condition Code:

0	Last field zero or zero length
1	Last field less than zero
2	Last field greater than zero
3	--

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Notes

1. Examples of the use of the EDIT instruction are given in Appendix A.

2. Editing includes sign and punctuation control, and the suppression and protection of leading zeros by replacing them with blanks or asterisks. It also facilitates programmed blanking of all-zero fields. Several fields may be edited in one operation, and numeric information may be combined with text.

3. In most cases, the source is shorter than the pattern because each four-bit source digit produces an eight-bit byte in the result.

4. The total number of digit selectors and significance starters in the pattern always equals the number of source digits edited.

5. If the fill byte is a blank, if no significance starter exists in the pattern, and if the source digit examined for each digit selector is zero, the editing operation blanks the result field.

6. The resulting condition code indicates whether or not the last field is all zeros and, if nonzero, reflects the state of the significance indicator. The significance indicator reflects the sign of the source field only if the last source byte examined contains a sign code in the rightmost four bits. For multiple-field editing operations, the condition code reflects the sign and value only of the field following the last field separator.

7. Significant performance degradation is possible when, with DAT on, the second-operand address of EDIT designates a location that is less than the length of the first operand to the left of a 2K-byte boundary. This is because the machine may perform a trial execution of the instruction to determine if the second operand actually crosses the boundary. The second operand of EDIT, while normally shorter than the first operand, can in the extreme case have the same length as the first.

Conditions				Results	
Pattern Byte	Previous State of Significance Indicator	Source Digit	Right Four Source Bits Are Plus Code	Result Byte	State of Significance Indicator at End of Digit Examination
Digit selector	Off	0	*	Fill byte	Off
		1-9	No	Source digit#	On
		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 byte	On
		0	Yes	Fill byte	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 byte	Off
Message byte	Off	**	**	Fill byte	Off
	On	**	**	Message byte	On

Explanation:

- * No effect on result byte or on new state of significance indicator.
- ** Not applicable because source is not examined.
- # For EDIT AND MARK only, the address of the rightmost such result byte is placed in general register 1.

Summary of Editing Functions

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

The second operand (the source), which normally contains one or more decimal numbers in the packed format, is changed to the zoned format and modified under the control of the first operand (the pattern). The address of the first significant result byte is inserted in general register 1. The edited result replaces the pattern.

EDIT AND MARK is identical to EDIT, except for the additional function of inserting the address of the result byte in bit positions 8-31 of general register 1 if the result byte is a zoned source digit and the significance indicator was off before the examination. Bits 0-7 of the register are not changed. If no result byte meets the criteria, general register 1 remains unchanged; if more than one result byte meets the criteria, the address of the rightmost such result byte is inserted.

See the figure "Summary of Editing Functions" under EDIT for a summary of the EDIT and EDIT AND MARK operations.

Resulting Condition Code:

0 Last field zero or zero length
1 Last field less than zero
2 Last field greater than zero
3 --

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data

Programming Notes

1. Examples of the use of the EDIT AND MARK instruction are given in Appendix A.

2. EDIT AND MARK facilitates the programming of floating currency-symbol insertion. Using appropriate source and pattern data, the address inserted in general register 1 is one greater than the address where a floating currency-sign would be inserted. BRANCH ON COUNT (BCTR), with zero in the R_2

field, may be used to reduce the inserted address by one.

3. No address is inserted in general register 1 when the significance indicator is turned on as a result of encountering a significance starter with the corresponding source digit zero. To ensure that general register 1 contains a proper address when this occurs, the address of the pattern byte that immediately follows the appropriate significance starter could be placed in the register beforehand.

4. When multiple fields are edited with one execution of the EDIT AND MARK instruction, the address, if any, inserted in general register 1 applies to the rightmost field edited for which the criteria were met.

5. See also the programming note under EDIT regarding performance degradation due to a possible trial execution.

MULTIPLY DECIMAL

MP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

```
+--------+----+----+----+--/--+----+--/--+
|  'FC'  | L1 | L2 | B1 |  D1 | B2 |  D2 |
+--------+----+----+----+--/--+----+--/--+
0        8    12   16   20    32   36    47
```

The product of the first operand (the multiplicand) and the second operand (the multiplier) is placed at the first-operand location. The operands and result are in the packed format.

The multiplier length cannot exceed 15 digits and sign (L_2 not greater than seven) and must be less than the multiplicand length (L_2 less than L_1); otherwise, a specification exception is recognized.

The multiplicand must have at least as many bytes of leftmost zeros as the number of bytes in the multiplier; otherwise, a data exception is recognized. This restriction ensures that no product overflow occurs.

The multiplicand, multiplier, and product are each signed decimal integers in the packed format and are right-aligned in their fields. All sign and digit codes of the multiplicand and multiplier are checked for validity.

The sign of the product is determined by the rules of algebra from the multiplier and multiplicand signs, even if one or both operands are zeros.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data
Specification

Programming Notes

1. An example of the use of the MULTIPLY DECIMAL instruction is given in Appendix A.

2. The product cannot exceed 31 digits and sign. The leftmost digit of the product is always zero.

SHIFT AND ROUND DECIMAL

SRP $D_1(L_1,B_1),D_2(B_2),I_3$ [SS]

```
+--------+----+----+----+--/--+----+--/--+
|  'F0'  | L1 | I3 | B1 |  D1 | B2 |  D2 |
+--------+----+----+----+--/--+----+--/--+
0        8    12   16   20    32   36    47
```

The first operand is shifted in the direction and for the number of decimal-digit positions specified by the second-operand address, and, when shifting to the right is specified, the absolute value of the first operand is rounded by the rounding digit, I_3. The first operand and the result are in the packed format.

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. The result replaces the first operand. Nothing is stored outside of the specified first-operand location.

The second-operand address, specified by the B_2 and D_2 fields, is not used to address data; bits 26-31 of that address are the shift value, and the leftmost bits of the address are ignored.

The shift value is a six-bit signed binary integer, indicating the direction and the number of decimal-digit positions to be shifted. Positive shift values specify shifting to the left. Negative shift values, which are represented in two's complement notation, specify shifting to the right. The following are examples of the interpretation of shift values:

Shift Value	Amount and Direction
011111	31 digits to the left
000001	One digit to the left
000000	No shift
111111	One digit to the right
100000	32 digits to the right

For a right shift, the I_3 field, bits 12-15 of the instruction, are used as a decimal rounding digit. The first operand, which is treated as positive by ignoring the sign, is rounded by decimally adding the rounding digit to the leftmost of the digits to be shifted out and by propagating the carry, if any, to the left. The result of this addition is then shifted right. Except for validity checking and the participation in rounding, the digits shifted out of the rightmost decimal-digit position are ignored and are lost.

If one or more nonzero digits are shifted out during a left shift, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs. Overflow cannot occur for a right shift, with or without rounding, or when no shifting is specified.

In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, the sign of the result is the same as the original sign but with the preferred sign code.

A data exception is recognized when the first operand does not have valid sign and digit codes or when the rounding digit is not a valid digit code. The validity of the first-operand codes is checked even when no shift is specified, and the validity of the rounding digit is checked even when no addition for rounding takes place.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

Access (fetch and store, operand 1)
Data
Decimal overflow

Programming Notes

1. Examples of the use of the SHIFT AND ROUND instruction are given in Appendix A.

2. 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 number even with rounding.

3. For right shifts, the rounding digit 5 provides conventional rounding of the result. The rounding digit 0 specifies truncation without rounding.

4. When the B_2 field is zero, the six-bit shift value is obtained directly from bits 42-47 of the instruction.

SUBTRACT DECIMAL

SP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

'FB'	L_1	L_2	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The second operand is subtracted from the first operand, and the resulting difference is placed at the first-operand location. The operands and result are in the packed format.

SUBTRACT DECIMAL is executed the same as ADD DECIMAL, except that the second operand is considered to have a sign opposite to the sign in storage. The second operand in storage remains unchanged.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

Access (fetch, operand 2; fetch and store, operand 1)
Data
Decimal overflow

ZERO AND ADD

ZAP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS]

'F8'	L_1	L_2	B_1	D_1	B_2	D_2

0 8 12 16 20 32 36 47

The second operand is placed at the first-operand location. The operation is equivalent to an addition to zero. The operand and result are in the packed format.

Only the second operand is checked for valid sign and digit codes. Extra zeros are supplied on the left for the shorter operand if needed.

If the first operand is too short to contain all leftmost nonzero digits of the second operand, decimal overflow occurs. The operation is completed. The result is obtained by ignoring the overflow digits, and condition code 3 is set. If the decimal-overflow mask is one, a program interruption for decimal overflow occurs.

In the absence of overflow, the sign of a zero result is made positive. If overflow occurs, a zero result is given the sign of the second operand but with the preferred sign code.

The two operands may overlap, provided the rightmost byte of the first operand is coincident with or to the right of the rightmost byte of the second operand. In this case the result is obtained as if the operands were processed right to left.

Resulting Condition Code:

0 Result zero; no overflow
1 Result less than zero; no overflow
2 Result greater than zero; no overflow
3 Overflow

Program Exceptions:

Access (fetch, operand 2; store, operand 1)
Data
Decimal overflow

Programming Note

An example of the use of the ZERO AND ADD instruction is given in Appendix A.

CHAPTER 9. FLOATING-POINT INSTRUCTIONS

Floating-point instructions are used to perform calculations on operands with a wide range of magnitude and to yield results scaled to preserve precision.

The floating-point instructions provide for loading, rounding, adding, subtracting, comparing, multiplying, dividing, and storing, as well as controlling the sign of short, long, and extended operands. Short operands generally permit faster processing and require less storage than long or extended operands. On the other hand, long and extended operands permit greater precision in computation. Four floating-point registers are provided. Instructions may perform either register-to-register or storage-and-register operations.

Most of the instructions generate normalized results, which preserve the highest precision in the operation. For addition and subtraction, instructions are also provided that generate unnormalized results. Either normalized or unnormalized numbers may be used as operands for any floating-point operation.

The rounding and extended-operand instructions are part of the extended-precision floating-point facility. The other floating-point instructions and the floating-point registers are part of the floating-point facility.

FLOATING-POINT NUMBER REPRESENTATION

A floating-point number consists of a signed hexadecimal fraction and an unsigned seven-bit binary integer called the characteristic. The characteristic represents a signed exponent and is obtained by adding 64 to the exponent value (excess-64 notation). The range of the characteristic is 0 to 127, which corresponds to an exponent range of -64 to +63. The value of a floating-point number is the product of its fraction and the number 16 raised to the power of the exponent which is represented by its characteristic.

The fraction of a floating-point number is treated as a hexadecimal number because it is considered to be multiplied by a number which is a power of 16. The name, fraction, indicates that the radix point is assumed to be immediately to the left of the leftmost fraction digit. The fraction is represented by its absolute value and a separate sign bit. The entire number is positive or negative, depending on whether the sign bit of the fraction is zero or one, respectively.

When a floating-point operation would cause the result exponent to exceed 63, the characteristic wraps around from 127 to 0, and an exponent-overflow condition exists. The result characteristic is then too small by 128. When an operation would cause the exponent to be less than -64, the characteristic wraps around from 0 to 127, and an exponent-underflow condition exists. The result characteristic is then too large by 128, except that a zero characteristic is produced when a true zero is forced.

A true zero is a floating-point number with a zero characteristic, zero fraction, and plus sign. A true zero may arise as the normal result of an arithmetic operation because of the particular magnitude of the operands. The result is forced to be a 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 the HALVE instruction, one or both operands of the MULTIPLY instruction, or the dividend in the DIVIDE instruction has a zero fraction.

When a program interruption for exponent underflow occurs, a true zero is not forced; instead, the fraction and sign remain correct, and the characteristic is too large by 128. When a program interruption for significance occurs, the fraction remains zero, the sign is positive, and the characteristic remains correct.

The sign of a sum, difference, product, or quotient with a zero fraction is positive. The sign of a zero fraction resulting from other operations is established from the operand sign, the same as for nonzero fractions.

NORMALIZATION

A quantity can be represented with the greatest precision by a floating-point number of a given fraction length when that number is normalized. A normalized floating-point number has a nonzero leftmost hexadecimal fraction digit. If one or more leftmost fraction digits are zeros, the number is said to be unnormalized.

Unnormalized numbers are normalized by shifting the fraction left, one digit at a time, until the leftmost hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A number with a zero fraction cannot be normalized; its characteristic either remains unchanged, or it is made zero when the result is forced to be a true zero.

Addition and subtraction with extended operands, as well as the MULTIPLY, DIVIDE, and HALVE operations, are performed only with normalization. Addition and subtraction with short or long operands may be specified as either normalized or unnormalized. For all other operations, the result is produced without normalization.

With unnormalized operations, leftmost zeros in the result fraction are not eliminated. The result may or may not be in normalized form, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. The operands for multiplication and division are normalized before the arithmetic process. For other normalized operations, normalization takes place when the intermediate arithmetic result is changed to the final result.

When the intermediate result of addition, subtraction, or rounding causes the fraction to overflow, the fraction is shifted right by one hexadecimal-digit position and the value one is supplied to the vacated leftmost digit position. The fraction is then truncated to the final result length, while the characteristic is increased by one. This adjustment is made for both normalized and unnormalized operations.

Programming Note

Up to three leftmost bits of the fraction of a normalized number may be zeros, since the nonzero test applies to the entire leftmost hexadecimal digit.

FLOATING-POINT-DATA FORMAT

Floating-point numbers have a 32-bit (short) format, a 64-bit (long) format, or a 128-bit (extended) format. Numbers in the short and long formats may be designated as operands both in storage and in the floating-point registers, whereas operands having the extended format can be designated only in the floating-point registers.

The floating-point registers contain 64 bits each and are numbered 0, 2, 4, and 6. A short or long floating-point number requires a single floating-point register. An extended floating-point number requires a pair of these registers: either registers 0 and 2 or registers 4 and 6; the two register pairs are designated as 0 or 4, respectively. When the R_1 or R_2 field of a floating-point instruction designates any register number other than 0, 2, 4, or 6 for the short or long format, or any register number other than 0 or 4 for the extended format, a program interruption for specification exception occurs.

Short Floating-Point Number

S	Characteristic	6-Digit Fraction	
0	1	8	31

Long Floating-Point Number

S	Characteristic	14-Digit Fraction	
0	1	8	63

Extended Floating-Point Number

High-Order Part

S	High-Order Characteristic	Leftmost 14 Digits of 28-Digit Fraction	
0	1	8	63

Low-Order Part

S	Low-Order Characteristic	Rightmost 14 Digits of 28-Digit Fraction	
64		72	127

In all formats, the first bit (bit 0) is the sign bit (S). The next seven bits are the characteristic. In the short and long formats, the remaining bits constitute the fraction, which consists of six or 14 hexadecimal digits, respectively.

A short floating-point number occupies only the leftmost 32 bit positions of a floating-point register. The rightmost 32 bit positions of the register are ignored when used as an operand in the short format and remain unchanged when a short result is placed in the register.

An extended floating-point number has a 28-digit fraction and consists of two long floating-point numbers which are called the high-order and low-order parts. The high-order part may be any long floating-point number. The fraction of the high-order part contains the leftmost 14 hexadecimal digits of the 28-digit fraction. The characteristic and sign of the high-order part are the characteristic and sign of the extended floating-point number. If the high-order part is normalized, the extended number is considered normalized. The fraction of the low-order part contains the rightmost 14 digits of the 28-digit fraction. The sign and characteristic of the low-order part of an extended operand are ignored.

When a result in the extended format is placed in a register pair, the sign of the low-order part is made the same as that of the high-order part, and, unless the result is a true zero, the low-order characteristic is made 14 less than the high-order characteristic. When the subtraction of 14 would cause the low-order characteristic to become less than zero, the characteristic is made 128 greater than its correct value. Exponent underflow is indicated only when the high-order characteristic underflows.

When an extended result is made a true zero, both the high-order and low-order parts are made a true zero.

The range' covered by the magnitude (M) of a normalized floating-point number depends on the format.

In the short format:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \leq M \leq (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \leq M \leq (1 - 16^{-28}) \times 16^{63}$$

In all formats, approximately:

$$5.4 \times 10^{-79} \leq M \leq 7.2 \times 10^{75}$$

Although the final result of a floating-point operation has six hexadecimal fraction digits in the short format, 14 fraction digits in the long format, and 28 fraction digits in the extended format, intermediate results have one additional hexadecimal digit on the right. This digit is called the guard digit. The guard digit may increase the precision of the final result because it participates in addition, subtraction, and comparison operations and in the left shift that occurs during normalization.

The entire set of floating-point operations is available for both short and long operands. The instructions generate a result that has the same format as the operands, except that for MULTIPLY, a long product is produced from a short multiplier and multiplicand. Floating-point operations in the extended format are available only for normalized addition, subtraction, and multiplication. MULTIPLY can also generate an extended product from a long multiplier and multiplicand. LOAD ROUNDED provides for rounding from extended to long format or from long to short format.

Programming Notes

1. A long floating-point number can be converted 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 the LOAD ROUNDED instruction.

2. 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 applies also when the result is a true zero. When the high-order characteristic is less than 14 but the number is not a true zero, the low-order part, when considered as a long floating-point number, does not express the correct characteristic value.

3. The entire fraction of an extended result participates in normalization. The low-order part alone may or may not appear to be a normalized long floating-point number, depending on whether the 15th digit of the normalized 28-digit fraction is nonzero or zero.

INSTRUCTIONS

The floating-point instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of Floating-Point Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

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

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For a register-to-register operation using LOAD (short), for example, LER is the mnemonic and R_1,R_2 the operand designation.

Name	Mne- monic	Characteristics										Op Code
ADD NORMALIZED (extended)	AXR	RR	C	XP		SP	EU	EO		LS		36
ADD NORMALIZED (long)	ADR	RR	C	FP		SP	EU	EO		LS		2A
ADD NORMALIZED (long)	AD	RX	C	FP	A	SP	EU	EO		LS		6A
ADD NORMALIZED (short)	AER	RR	C	FP		SP	EU	EO		LS		3A
ADD NORMALIZED (short)	AE	RX	C	FP	A	SP	EU	EO		LS		7A
ADD UNNORMALIZED (long)	AWR	RR	C	FP		SP		EO		LS		2E
ADD UNNORMALIZED (long)	AW	RX	C	FP	A	SP		EO		LS		6E
ADD UNNORMALIZED (short)	AUR	RR	C	FP		SP		EO		LS		3E
ADD UNNORMALIZED (short)	AU	RX	C	FP	A	SP		EO		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	EU	EO	FK			2D
DIVIDE (long)	DD	RX		FP	A	SP	EU	EO	FK			6D
DIVIDE (short)	DER	RR		FP		SP	EU	EO	FK			3D
DIVIDE (short)	DE	RX		FP	A	SP	EU	EO	FK			7D
HALVE (long)	HDR	RR		FP		SP	EU					24
HALVE (short)	HER	RR		FP		SP	EU					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 (ext. to long)	LRDR	RR		XP		SP		EO				25
LOAD ROUNDED (long to short)	LRER	RR		XP		SP		EO				35
MULTIPLY (extended)	MXR	RR		XP		SP	EU	EO				26
MULTIPLY (long)	MDR	RR		FP		SP	EU	EO				2C
MULTIPLY (long)	MD	RX		FP	A	SP	EU	EO				6C
MULTIPLY (long to extended)	MXDR	RR		XP		SP	EU	EO				27
MULTIPLY (long to extended)	MXD	RX		XP	A	SP	EU	EO				67
MULTIPLY (short to long)	MER	RR		FP		SP	EU	EO				3C
MULTIPLY (short to long)	ME	RX		FP	A	SP	EU	EO				7C
STORE (long)	STD	RX		FP	A	SP					ST	60
STORE (short)	STE	RX		FP	A	SP					ST	70
SUBTRACT NORMALIZED (ext.)	SXR	RR	C	XP		SP	EU	EO		LS		37
SUBTRACT NORMALIZED (long)	SDR	RR	C	FP		SP	EU	EO		LS		2B
SUBTRACT NORMALIZED (long)	SD	RX	C	FP	A	SP	EU	EO		LS		6B
SUBTRACT NORMALIZED (short)	SER	RR	C	FP		SP	EU	EO		LS		3B
SUBTRACT NORMALIZED (short)	SE	RX	C	FP	A	SP	EU	EO		LS		7B
SUBTRACT UNNORMALIZED (long)	SWR	RR	C	FP		SP		EO		LS		2F
SUBTRACT UNNORMALIZED (long)	SW	RX	C	FP	A	SP		EO		LS		6F
SUBTRACT UNNORMALIZED (short)	SUR	RR	C	FP		SP		EO		LS		3F
SUBTRACT UNNORMALIZED (short)	SU	RX	C	FP	A	SP		EO		LS		7F

Summary of Floating-Point Instructions (Part 1 of 2)

Explanation:

A	Access exceptions for logical addresses.
C	Condition code is set.
EO	Exponent-overflow exception.
EU	Exponent-underflow exception.
FK	Floating-point-divide exception.
FP	Floating-point facility.
LS	Significance exception.
RR	RR instruction format.
RX	RX instruction format.
SP	Specification exception.
ST	PER storage-alteration event.
XP	Extended-precision floating-point facility.

Summary of Floating-Point Instructions (Part 2 of 2)

ADD NORMALIZED

AER R_1,R_2 [RR, Short Operands]

```
┌─────────┬────┬────┐
│  '3A'   │ R1 │ R2 │
└─────────┴────┴────┘
0         8    12   15
```

AE $R_1,D_2(X_2,B_2)$ [RX, Short Operands]

```
┌─────────┬────┬────┬────┬────────────┐
│  '7A'   │ R1 │ X2 │ B2 │     D2     │
└─────────┴────┴────┴────┴────────────┘
0         8    12   16   20           31
```

ADR R_1,R_2 [RR, Long Operands]

```
┌─────────┬────┬────┐
│  '2A'   │ R1 │ R2 │
└─────────┴────┴────┘
0         8    12   15
```

AD $R_1,D_2(X_2,B_2)$ [RX, Long Operands]

```
┌─────────┬────┬────┬────┬────────────┐
│  '6A'   │ R1 │ X2 │ B2 │     D2     │
└─────────┴────┴────┴────┴────────────┘
0         8    12   16   20           31
```

AXR R_1,R_2 [RR, Extended Operands]

```
┌─────────┬────┬────┐
│  '36'   │ R1 │ R2 │
└─────────┴────┴────┘
0         8    12   15
```

The second operand is added to the first operand, and the normalized sum is placed at the first-operand location.

Addition of two floating-point numbers consists in characteristic comparison, fraction alignment, and signed fraction addition. The characteristics of the two operands are compared, and the fraction accompanying the smaller characteristic is aligned with the other fraction by a right shift, with its characteristic increased by one for each hexadecimal digit of shift until the two characteristics agree.

When a fraction is shifted right during alignment, the leftmost hexadecimal digit shifted out is retained as a guard digit. The fraction that is not shifted is considered to be extended with a zero in the guard-digit position. When no alignment shift occurs, both operands are considered to be extended with zeros in the guard-digit position. The fractions with signs are then added algebraically to form a signed intermediate sum.

The intermediate-sum fraction consists of seven (short format), 15 (long format), or 29 (extended format) hexadecimal digits, including the guard digit, and a possible carry. If a carry is present, the sum is shifted right one digit position so that the carry becomes the leftmost digit of the fraction, and the characteristic is increased by one.

If the addition produces no carry, the intermediate-sum fraction is shifted left as necessary to eliminate any leading hexadecimal zero digits resulting from the addition, provided the fraction is not zero. Zeros are supplied to the vacated rightmost digits, and the characteristic is reduced by the number of hexadecimal digits of shift. The fraction thus normalized is then truncated on the right to six (short format), 14 (long format), or 28 (extended format) hexadecimal digits. In the extended format, a characteristic is generated for the low-order part, which is 14 less than the high-order characteristic.

The sign of the sum is determined by the rules of algebra, unless all digits of the intermediate-sum fraction are zero, in which case the sign is made plus.

An exponent-overflow exception is recognized when a carry from the leftmost position of the intermediate-sum fraction would cause the characteristic of the normalized sum to exceed 127. The operation is completed by making the result characteristic 128 less than the correct value, and a program interruption for exponent overflow takes place. The result sign and fraction remain correct, and, for AXR, the characteristic of the low-order part remains correct.

An exponent-underflow exception is recognized when the characteristic of the normalized sum would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the result characteristic 128 greater than the correct value. The result sign and fraction remain correct, and a program interruption for exponent underflow 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, no exponent underflow is recognized when the characteristic of the low-order part would be less than zero but the characteristic of the high-order part is zero or greater.

The result fraction is zero when the intermediate-sum fraction, including the guard digit, is zero. With a zero result fraction, the action depends on the setting of the significance mask bit. If the significance mask bit is one, no normalization occurs, the intermediate and final result characteristics are the same, 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 zero
1 Result less than zero
2 Result greater than zero
3 --

Program Exceptions:

Access (fetch, operand 2 of AE and AD only)
Exponent overflow
Exponent underflow
Operation (if the floating-point facility is not installed, or, for AXR, if the extended-precision floating-point facility is not installed)
Significance
Specification

Programming Notes

1. An example of the use of the ADD NORMALIZED instruction is given in Appendix A.

2. Interchanging the two operands in a floating-point addition does not affect the value of the sum.

3. The ADD NORMALIZED instruction normalizes the sum but not the operands. Thus, if one or both operands are unnormalized, precision may be lost during fraction alignment.

ADD UNNORMALIZED

AUR R_1,R_2 [RR, Short Operands]

'3E'	R_1	R_2
0	8	12 15

AU $R_1,D_2(X_2,B_2)$ [RX, Short Operands]

'7E'	R_1	X_2	B_2	D_2
0	8	12	16	20 31

AWR R_1,R_2 [RR, Long Operands]

'2E'	R_1	R_2
0	8	12 15

AW $R_1,D_2(X_2,B_2)$ [RX, Long Operands]

'6E'	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 unnormalized sum is placed at the first-operand location.

The execution of ADD UNNORMALIZED is identical to that of ADD NORMALIZED, except that:

1. When no carry is present after the addition, the intermediate-sum fraction is truncated to the proper result-fraction length without a left shift to eliminate leading hexadecimal zeros and without the corresponding reduction of the characteristic.

2. Exponent underflow cannot occur.

3. The guard digit does not participate in the recognition of a zero result fraction. A zero result fraction is recognized when the fraction (that is, the intermediate-sum fraction, excluding the guard digit) is zero.

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 zero
1 Result less than zero
2 Result greater than zero
3 --

Program Exceptions:

Access (fetch, operand 2 of AU and AW only)
Exponent overflow
Operation (if the floating-point facility is not installed)
Significance
Specification

Programming Notes

1. An example of the use of the ADD UNNORMALIZED instruction is given in Appendix A.

2. Except when the result is made a true zero, the characteristic of the result of ADD UNNORMALIZED is equal to the greater of the two operand characteristics, increased by one if the fraction addition produced a carry, or set to zero if exponent overflow occurred.

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.

The comparison is algebraic and follows the procedure for normalized floating-point subtraction, except that the difference is discarded after setting the condition code and both operands remain unchanged. When the difference, including the guard digit, is zero, the operands are equal. When a nonzero difference is positive or negative, the first operand is high or low, respectively.

An 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 equal
1 First operand low
2 First operand high
3 --

Program Exceptions:

Access (fetch, operand 2 of CE and CD only)
Operation (if the floating-point facility is not installed)
Specification

Programming Notes

1. Examples of the use of the COMPARE instruction are given in Appendix A.

2. An exponent inequality alone is not sufficient to determine the inequality of two operands with the same sign, because the fractions may have different numbers of leading hexadecimal zeros.

3. Numbers with zero fractions compare equal even when they differ in sign or characteristic.

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 the normalized quotient is placed at the first-operand location. No remainder is preserved.

Floating-point division consists in characteristic subtraction and fraction division. The operands are first normalized to eliminate leading hexadecimal zeros. The difference between the dividend and divisor characteristics of the normalized operands, plus 64, is used as the characteristic of an intermediate quotient.

All dividend and divisor fraction digits participate in forming the fraction of the intermediate quotient. The intermediate-quotient fraction can have no leading hexadecimal zeros, but a right shift of one digit position may be necessary with an increase of the characteristic by one. The fraction is then truncated to the proper result-fraction length.

An exponent-overflow exception is recognized when the characteristic of the final quotient would exceed 127 and the fraction is not zero. The operation is completed by making the characteristic 128 less than the correct value. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent overflow occurs.

An exponent-underflow exception exists when the characteristic of the final quotient would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the quotient a true zero.

Exponent underflow does not occur when an operand characteristic becomes less than zero during normalization of the operands or when the intermediate-quotient characteristic is less than zero, as long as the final quotient can be represented with the correct characteristic.

When the divisor fraction is zero, a floating-point-divide exception is recognized. This includes the case of division of zero by zero.

When the dividend fraction is zero, but the divisor fraction is nonzero, the quotient is made a true zero. No exponent overflow or exponent underflow occurs.

The sign of the quotient is determined by the rules of algebra, except that the sign is always plus when the quotient is made a true zero.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2 of DD and DE only)
Exponent overflow
Exponent underflow
Floating-point divide
Operation (if the floating-point facility is not installed)
Specification

Programming Note

Examples of the use of the DIVIDE instruction are given in Appendix A.

HALVE

HER R_1,R_2 [RR, Short Operands]

'34'	R_1	R_2
0	8	12 15

HDR R_1,R_2 [RR, Long Operands]

'24'	R_1	R_2
0	8	12 15

The second operand is divided by 2, and the normalized quotient is placed at the first-operand location.

The fraction of the second operand is shifted right one bit position, placing the contents of the rightmost bit position in the leftmost bit position of the guard digit, and a zero is supplied to the leftmost bit position of the fraction. The intermediate result, including the guard digit, is then normalized, and the final result is truncated to the proper length.

An exponent-underflow exception exists when the characteristic of the final result would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, a program interruption does not take place; instead, the operation is completed by making the result a true zero.

When the fraction of the second operand is zero, the result is made a true zero, and no exponent underflow occurs.

The sign of the result is the same as that of the second operand, except that the sign is always plus when the quotient is made a true zero.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

<u>Condition Code</u>: The code remains unchanged.

<u>Program Exceptions</u>:

Exponent underflow
Operation (if the floating-point facility is not installed)
Specification

<u>Programming Notes</u>

1. An example of the use of the HALVE instruction is given in Appendix A.

2. With short and long operands, the halve operation is identical to a divide operation with the number 2 as divisor. Similarly, the result of HDR is identical to that of MD or MDR with one-half as a multiplier. No multiply operation corresponds to HER, since no multiply operation produces short results.

3. The result of HALVE is zero only when the second-operand fraction is zero, or when exponent underflow occurs with the exponent-underflow mask set to zero. A fraction with zeros in every bit position, except for a one in the rightmost bit position, does not become zero after the right shift. This is because the one bit is preserved in the guard-digit position and, when the result is not made a true zero because of exponent underflow, becomes the leftmost bit after normalization of the result.

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 at 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:

Access (fetch, operand 2 of LE and LD only)
Operation (if the floating-point facility is not installed)
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 at 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 zero
1	Result less than zero
2	Result greater than zero
3	--

Program Exceptions:

Operation (if the floating-point facility is not installed)
Specification

Programming Note

When the same register is designated as the first-operand and second-operand location, the operation is equivalent to a test without data movement.

LOAD COMPLEMENT

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 at the first-operand location with the sign bit inverted.

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 zero
1	Result less than zero
2	Result greater than zero
3	--

Program Exceptions:

Operation (if the floating-point facility 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 at 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 zero
1 Result less than zero
2 --
3 --

Program Exceptions:

Operation (if the floating-point facility 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 at 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 zero
1 --
2 Result greater than zero
3 --

Program Exceptions:

Operation (if the floating-point facility 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 shorter format, and the result is placed at 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 any carry to the left. 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 leftmost hexadecimal digit position of the fraction, the fraction is shifted right one digit position so that the carry becomes the leftmost digit of the fraction, and the characteristic is increased by one.

The intermediate fraction is then truncated to the proper result-fraction length.

The sign of the result is the same as the sign of the second operand. There is no normalization to eliminate leading zeros.

An exponent-overflow exception exists when shifting the fraction right would cause 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:

Exponent overflow
Operation (if the extended-precision floating-point facility is not installed)
Specification

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 at the first-operand location.

Multiplication of two floating-point numbers consists in exponent addition and fraction multiplication. The operands are first normalized to eliminate leading hexadecimal zeros. The sum of the characteristics of the normalized operands, less 64, is used as the characteristic of the intermediate product.

The fraction of the intermediate product is the exact product of the normalized operand fractions. When the intermediate-product fraction has one leading hexadecimal zero digit, the fraction is shifted left one digit position, bringing the contents of the guard-digit position into the rightmost position of the result fraction, and the intermediate-product characteristic is reduced by one. The fraction is then truncated to the proper result-fraction length.

For MER and ME, the multiplier and multiplicand fractions have six hexadecimal digits; the product fraction has the full 14 digits of the long format, with the two rightmost fraction digits always zeros. For MDR and MD, the multiplier and multiplicand fractions have 14 digits, and the final 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 final product fraction contains 28 digits and is an exact product of the operand fractions. For MXR, the multiplier and multiplicand fractions have 28 digits, and the final product fraction is truncated to 28 digits.

An exponent-overflow exception is recognized when the characteristic of the final product would exceed 127 and the fraction 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 would also exceed 127, it, too, is decreased by 128. The result is normalized, and the sign and fraction remain correct. A program interruption for exponent overflow occurs.

Exponent overflow is not recognized when the intermediate-product characteristic is initially 128 but is brought back within range by normalization.

An exponent-underflow exception exists when the characteristic of the final product would be less than zero and the fraction is not zero. If the exponent-underflow mask bit is one, the operation is completed by making the characteristic 128 greater than the correct value, and a program interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is zero, 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 would be less than zero but the high-order characteristic is equal to or greater than zero.

Exponent underflow does not occur when the characteristic of an operand becomes less than zero during normalization of the operands, as long as the final product can be represented with the correct characteristic.

When either or both operand fractions are zero, the result is made a true zero, and no exponent overflow or exponent underflow occurs.

The sign of the product is determined by the rules of algebra, except that the sign is always zero when the result is made a true zero.

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.

<u>Condition</u> <u>Code</u>: The code remains unchanged.

<u>Program</u> <u>Exceptions</u>:

Access (fetch, operand 2 of ME, MD, and MXD only)
Exponent overflow
Exponent underflow
Operation (if the floating-point facility is not installed, or, for MXDR, MXD, and MXR, if the extended-precision floating-point facility is not installed)
Specification

<u>Programming</u> <u>Notes</u>

1. An example of the use of the MULTIPLY instruction is given in Appendix A.

2. 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]

'70'	R_1	X_2	B_2	D_2
0	8	12	16	20 31

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.

<u>Condition</u> <u>Code</u>: The code remains unchanged.

<u>Program</u> <u>Exceptions</u>:

Access (store, operand 2)
Operation (if the floating-point facility is not installed)
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'  │ R1 │ R2 │
└────────┴────┴────┘
0        8    12  15
```

The second operand is subtracted from the first operand, and the normalized difference is placed at the first-operand location.

The execution of SUBTRACT NORMALIZED is identical to that of ADD NORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

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.

<u>Resulting</u> <u>Condition</u> <u>Code</u>:

0	Result fraction zero
1	Result less than zero
2	Result greater than zero
3	--

<u>Program</u> <u>Exceptions</u>:

Access (fetch, operand 2 of SE and SD only)
Exponent overflow
Exponent underflow
Operation (if the floating-point facility is not installed, or, for SXR, if the extended-precision floating-point facility is not installed)
Significance
Specification

SUBTRACT UNNORMALIZED

SUR R_1,R_2 [RR, Short Operands]

```
┌────────┬────┬────┐
│  '3F'  │ R1 │ R2 │
└────────┴────┴────┘
0        8    12  15
```

SU $R_1,D_2(X_2,B_2)$ [RX, Short Operands]

```
┌────────┬────┬────┬────┬────────────┐
│  '7F'  │ R1 │ X2 │ B2 │     D2     │
└────────┴────┴────┴────┴────────────┘
0        8    12   16   20          31
```

SWR R_1,R_2 [RR, Long Operands]

```
┌────────┬────┬────┐
│  '2F'  │ R1 │ R2 │
└────────┴────┴────┘
0        8    12  15
```

SW $R_1,D_2(X_2,B_2)$ [RX, Long Operands]

```
┌────────┬────┬────┬────┬────────────┐
│  '6F'  │ R1 │ X2 │ B2 │     D2     │
└────────┴────┴────┴────┴────────────┘
0        8    12   16   20          31
```

The second operand is subtracted from the first operand, and the unnormalized difference is placed at the first-operand location.

The execution of SUBTRACT UNNORMALIZED is identical to that of ADD UNNORMALIZED, except that the second operand participates in the operation with its sign bit inverted.

The R_1 and R_2 fields must designate register 0, 2, 4, or 6; otherwise, a specification exception is recognized.

<u>Resulting</u> <u>Condition</u> <u>Code</u>:

0	Result fraction zero
1	Result less than zero
2	Result greater than zero
3	--

<u>Program</u> <u>Exceptions</u>:

Access (fetch, operand 2 of SU and SW only)
Exponent overflow
Operation (if the floating-point facility is not installed)
Significance
Specification

CHAPTER 10. CONTROL INSTRUCTIONS

This chapter includes all privileged and semiprivileged instructions described in this publication, except the input/output instructions, which are described in Chapter 13, "Input/Output Operations."

Privileged instructions may be executed only when the CPU is in the supervisor state. An attempt to execute an installed privileged instruction in the problem state generates a privileged-operation exception.

The semiprivileged instructions are those instructions that can be executed in the problem state when certain authority requirements are met. An attempt to execute an installed semiprivileged instruction in the problem state when the authority requirements are not met generates a privileged-operation exception or some other program-interruption condition depending on the particular requirement which is violated. Those requirements which cause a privileged-operation exception to be generated in the problem state are not enforced when execution is attempted in the supervisor state.

The control instructions and their mnemonics, formats, and operation codes are listed in the figure "Summary of Control Instructions." The figure also indicates when the condition code is set and the exceptional conditions in operand designations, data, or results that cause a program interruption.

For those control instructions which have special rules regarding the handling of exceptional situations, a section called "Special Conditions" is included. This section indicates the type of ending (suppression, nullification, or completion) only for those exceptions for which the ending may vary.

<u>Note</u>: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. For LOAD PSW, for example, LPSW is the mnemonic and $D_2(B_2)$ the operand designation.

Name	Mnemonic	Characteristics													Op Code
CONNECT CHANNEL SET	CONCS	S	C	CS	P					$					B200
DIAGNOSE			DM		P	DM									83
DISCONNECT CHANNEL SET	DISCS	S	C	CS	P					$					B201
EXTRACT PRIMARY ASN	EPAR	RRE		DU	Q			SO					R		B226
EXTRACT SECONDARY ASN	ESAR	RRE		DU	Q			SO					R		B227
INSERT ADDRESS SPACE CONTROL	IAC	RRE	C	DU	Q			SO					R		B224
INSERT PSW KEY	IPK	S		PK	Q						G2		R		B20B
INSERT STORAGE KEY	ISK	RR			P	A^1	SP	SO					R		09
INSERT STORAGE KEY EXTENDED	ISKE	RRE		EK	P	A^1							R		B229
INSERT VIRTUAL STORAGE KEY	IVSK	RRE		DU	Q	A^1		SO					R		B223
INVALIDATE PAGE TABLE ENTRY	IPTE	RRE		EF	P	A^1				$					B221
LOAD ADDRESS SPACE PARAMETERS	LASP	SSE	C	DU	P	AS	SP	SO							E500
LOAD CONTROL	LCTL	RS			P	A	SP								B7
LOAD PSW	LPSW	S	L		P	A	SP			¢					82
LOAD REAL ADDRESS	LRA	RX	C	TR	P	A^1							R		B1
MOVE TO PRIMARY	MVCP	SS	C	DU	Q	A		SO		¢				ST	DA
MOVE TO SECONDARY	MVCS	SS	C	DU	Q	A		SO		¢				ST	DB
MOVE WITH KEY	MVCK	SS	C	DU	Q	A								ST	D9
PROGRAM CALL	PC	S		DU	Q	AT		Z^1	T	¢	GM	B	R	ST	B218
PROGRAM TRANSFER	PT	RRE		DU	Q	AT	SP	Z^2	T	¢		B		ST	B228
PURGE TLB	PTLB	S		TR	P					$					B20D
READ DIRECT	RDD	SI		DC	P	A^1				$				SD	85
RESET REFERENCE BIT	RRB	S	C	TR	P	A^1		SO							B213
RESET REFERENCE BIT EXTENDED	RRBE	RRE	C	EK	P	A^1									B22A
SET ADDRESS SPACE CONTROL	SAC	S		DU			SP	SO		¢					B219
SET CLOCK	SCK	S	C		P	A	SP								B204
SET CLOCK COMPARATOR	SCKC	S		CK	P	A	SP								B206
SET CPU TIMER	SPT	S		CK	P	A	SP								B208
SET PREFIX	SPX	S		MP	P	A	SP			$					B210
SET PSW KEY FROM ADDRESS	SPKA	S		PK	Q										B20A
SET SECONDARY ASN	SSAR	RRE		DU		AT		Z^3	T	¢				ST	B225
SET STORAGE KEY	SSK	RR			P	A^1	SP	SO		¢					08
SET STORAGE KEY EXTENDED	SSKE	RRE		EK	P	A^1				¢					B22B
SET SYSTEM MASK	SSM	S			P	A	SP	SO							80
SIGNAL PROCESSOR	SIGP	RS	C	MP	P					$			R		AE
STORE CLOCK COMPARATOR	STCKC	S		CK	P	A	SP							ST	B207
STORE CONTROL	STCTL	RS			P	A	SP							ST	B6
STORE CPU ADDRESS	STAP	S		MP	P	A	SP							ST	B212
STORE CPU ID	STIDP	S			P	A	SP							ST	B202
STORE CPU TIMER	STPT	S		CK	P	A	SP							ST	B209
STORE PREFIX	STPX	S		MP	P	A	SP							ST	B211
STORE THEN AND SYSTEM MASK	STNSM	SI		TR	P	A								ST	AC
STORE THEN OR SYSTEM MASK	STOSM	SI		TR	P	A	SP							ST	AD
TEST BLOCK	TB	RRE	C	TB	P	A^1		II		$	G0		R		B22C
TEST PROTECTION	TPROT	SSE	C	EF	P	A^1									E501
WRITE DIRECT	WRD	SI		DC	P	A^1				$					84

Summary of Control Instructions (Part 1 of 2)

Explanation:

¢	Causes serialization and checkpoint synchronization.
$	Causes serialization.
A	Access exceptions for logical addresses.
A^1	Access exceptions; not all access exceptions may occur; see instruction description for details.
AS	Access exceptions and ASN-translation-specification exception; see instruction description for details.
AT	ASN-translation exceptions (which include addressing, ASN-translation specification, AFX translation, and ASX translation).
B	PER branch event.
C	Condition code is set.
CK	CPU-timer and clock-comparator facility.
CS	Channel-set-switching facility.
DC	Direct-control facility.
DM	Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code.
DU	Dual-address-space facility.
EF	Extended facility.
EK	Storage-key-instruction-extension facility.
G0	Instruction execution includes the implied use of general register 0.
G2	Instruction execution includes the implied use of general register 2.
GM	Instruction execution includes the implied use of general registers 3, 4, and 14.
II	Interruptible instruction.
L	New condition code is loaded.
MP	Multiprocessing facility.
P	Privileged-operation exception.
PK	PSW-key-handling facility.
Q	Privileged-operation exception for semiprivileged instructions.
R	PER general-register-alteration event.
RR	RR instruction format.
RRE	RRE instruction format.
RS	RS instruction format.
RX	RX instruction format.
S	S instruction format.
SD	PER storage-alteration event, which can be caused by READ DIRECT only when INVALIDATE PAGE TABLE ENTRY is not installed.
SI	SI instruction format.
SO	Special-operation exception.
SP	Specification exception.
SS	SS instruction format.
SSE	SSE instruction format.
ST	PER storage-alteration event.
T	Trace exceptions (which include access and specification).
TB	Test-block facility.
TR	Translation facility.
Z^1	Additional exceptions and events for PROGRAM CALL (which include addressing, EX-translation, LX-translation, PC-translation-specification, and special-operation exceptions and space-switch event).
Z^2	Additional exceptions and events for PROGRAM TRANSFER (which include addressing, primary-authority, and special-operation exceptions and space-switch event).
Z^3	Additional exceptions for SET SECONDARY ASN (which include addressing, secondary authority, and special operation).

Summary of Control Instructions (Part 2 of 2)

CONNECT CHANNEL SET

CONCS $D_2(B_2)$ [S]

'B200'	B_2	D_2	
0	16	20	31

The channel set currently connected to this CPU is disconnected, and the addressed channel set, if currently disconnected, is connected to this CPU.

The second-operand address, specified by the B_2 and D_2 fields, is not used to address data; bits 16-31 form the 16-bit channel-set address. Bits 8-15 of the second-operand address are ignored.

When the channel set currently connected to this CPU is not the channel set addressed by the instruction, the currently connected channel set is immediately disconnected from this CPU, regardless of whether the channel set addressed by the instruction is operational or can be connected to this CPU.

If the addressed channel set is currently connected to this CPU, no channel-set connection is changed, and condition code 0 is set. If the addressed channel set is operational and currently disconnected, it is connected to this CPU, and condition code 0 is set.

When the addressed channel set is connected to another CPU, it is not connected to this CPU, and condition code 1 is set.

When the addressed channel set is not operational, no connection is performed, and condition code 3 is set.

A serialization function is performed.

If a channel in the channel set which is connected by means of this instruction has an I/O interruption pending, and if the CPU is enabled for I/O interruptions, the interruption is recognized at the completion of this instruction.

Resulting Condition Code:

0 Connection completed
1 Connection not performed; channel set connected to another CPU
2 --
3 Not operational

Program Exceptions:

Operation (if the channel-set-switching facility is not installed)
Privileged operation

Programming Note

The switching of channel sets and the associated states of a channel set are described in the section "Channel-Set Switching" in Chapter 4, "Control."

DIAGNOSE

```
┌────────┬───────────────────────┐
│  '83'  │                       │
└────────┴───────────────────────┘
0        8                      31
```

The CPU performs built-in diagnostic functions, or other model-dependent functions. The purpose of the diagnostic functions is to verify proper functioning of equipment and to locate faulty components. Other model-dependent functions may include disabling of failing buffers, reconfiguration of CPUs, storage, channel sets, 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 may cause the test indicator 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

1. Since the instruction is not intended for problem-state-program or control-program use, DIAGNOSE has no mnemonic.

2. DIAGNOSE, unlike other instructions, does not follow the rule that programming errors are distinguished from equipment errors. Improper use of DIAGNOSE may result in false machine-check indications or may cause actual machine malfunctions to be ignored. It may also alter other aspects of system operation, including instruction execution and channel-program operation, to an extent that the operation does not comply with that specified in this publication. As a result of the improper use of DIAGNOSE, the system may be left in such a condition that the power-on reset or initial-microprogram-loading (IML) function must be performed. Since the function performed by DIAGNOSE may differ from model to model and between versions of a model, the program should avoid issuing DIAGNOSE unless the program recognizes both the model number and version code stored by STORE CPU ID.

DISCONNECT CHANNEL SET

DISCS $D_2(B_2)$ [S]

'B201'	B_2	D_2
0	16	20 31

The addressed channel set is disconnected from the CPU to which it is currently connected. If the channel set is not connected, no operation is performed.

The second-operand address, specified by the B_2 and D_2 fields, is not used to address data; bits 16-31 form the 16-bit channel-set address. Bits 8-15 of the second-operand address are ignored.

When the addressed channel set is operational but not connected to any CPU, no disconnection operation is performed, and condition code 0 is set.

When the addressed channel set is connected either to the CPU issuing the DISCONNECT CHANNEL SET instruction or to a CPU that is in the stopped or check-stop state, the disconnection operation is performed, and condition code 0 is set.

When the addressed channel set is connected to another CPU which is in the operating state, which is being reset, or for which a SIGNAL PROCESSOR reset order or IML order is pending, no disconnection operation is performed, and condition code 1 is set.

When the addressed channel set is connected to another CPU which is in the operator-intervening state, it depends on the model if condition code 0 or 1 is set. The action taken in this case is consistent with the condition code indicated.

When the addressed channel set is not operational, no disconnection operation is performed, and condition code 3 is set.

A serialization function is performed.

If a channel in a channel set which is disconnected by this instruction has an I/O interruption pending, the interruption condition remains pending in the channel while the channel set is in the disconnected state.

Resulting Condition Code:

0 Disconnection completed
1 Disconnection not performed; channel set connected to another CPU not in proper state
2 --
3 Not operational

Program Exceptions:

Operation (if the channel-set-switching facility is not installed)
Privileged operation

Programming Note

The switching of channel sets and the associated states of a channel set are described in the section "Channel-Set Switching" in Chapter 4, "Control."

EXTRACT PRIMARY ASN

EPAR R_1 [RRE]

'B226'	////////	R_1	////
0	16	24	28 31

The 16-bit PASN, bits 16-31 of control register 4, is placed in bit positions 16-31 of general register R_1. Bits 0-15 of the general register are set to zeros.

Bits 16-23 and 28-31 of the instruction are ignored.

Special Conditions

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: EXTRACT PRIMARY ASN."

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the dual-address-space facility is not installed)
Privileged operation (extraction-authority control is zero in the problem state)
Special operation

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B.1 Operation exception if the dual-address-space facility is not installed.

7.B.2 Special-operation exception due to DAT being off.

8. Privileged-operation exception due to extraction-authority control, bit 4 of control register 0, being zero.

Priority of Execution: EXTRACT PRIMARY ASN

EXTRACT SECONDARY ASN

ESAR R_1 [RRE]

'B227'	////////	R_1	////

0 16 24 28 31

The 16-bit SASN, bits 16-31 of control register 3, is placed in bit positions 16-31 of general register R_1. Bits 0-15 of the general register are set to zeros.

Bits 16-23 and 28-31 of the instruction are ignored.

Special Conditions

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: EXTRACT SECONDARY ASN."

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the dual-address-space facility is not installed)
Privileged operation (extraction-authority control is zero in the problem state)
Special operation

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B.1 Operation exception if the dual-address-space facility is not installed.

7.B.2 Special-operation exception due to DAT being off.

8. Privileged-operation exception due to extraction-authority control, bit 4 of control register 0, being zero.

Priority of Execution: EXTRACT SECONDARY ASN

INSERT ADDRESS SPACE CONTROL

IAC R_1 [RRE]

'B224'	////////	R_1	////

0 16 24 28 31

The address-space-control bit, bit 16 of the current PSW, is placed in bit position 23 of general register R_1. Bits 16-22 of the register are set to zeros, and bits 0-15 and 24-31 of the register remain unchanged. The address-space-control bit is also used to set the condition code.

Bits 16-23 and 28-31 of the instruction are ignored.

Special Conditions

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: INSERT ADDRESS SPACE CONTROL."

<u>Resulting Condition Code</u>:

0 PSW bit 16 zero
1 PSW bit 16 one
2 --
3 --

<u>Program Exceptions</u>:

Operation (if the dual-address-space facility is not installed)
Privileged operation (extraction-authority control is zero in the problem state)
Special operation

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B.1 Operation exception if the dual-address-space facility is not installed.

7.B.2 Special-operation exception due to DAT being off.

8. Privileged-operation exception due to extraction-authority control, bit 4 of control register 0, being zero.

Priority of Execution: INSERT ADDRESS SPACE CONTROL

<u>Programming Notes</u>

1. Bits 16-22 of general register R_1 are reserved for expansion for use with possible future facilities. The program should not depend on these bits being set to zeros. Similarly, condition codes 2 and 3 may be set as a result of future facilities.

2. INSERT ADDRESS SPACE CONTROL and SET ADDRESS SPACE CONTROL are defined to operate on the third byte of a general register so that the address-space-control bit can be saved in the same general register as the PSW key, which is placed in the fourth byte of general register 2 by INSERT PSW KEY.

INSERT PSW KEY

IPK [S]

'B20B'	/////////////////
0	16 31

The four-bit PSW-key, bits 8-11 of the current PSW, is inserted in bit positions 24-27 of general register 2, and bits 28-31 of that register are set to zeros. Bits 0-23 of general register 2 remain unchanged.

Bits 16-31 of the instruction are ignored.

<u>Special Conditions</u>

In the problem state, when DAS is installed, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. When DAS is not installed, execution of the instruction in the problem state results in a privileged-operation exception regardless of the extraction-authority control. In the supervisor state, the extraction-authority-control bit is not examined.

<u>Condition Code</u>: The code remains unchanged.

<u>Program Exceptions</u>:

Operation (if the PSW-key-handling facility is not installed)
Privileged operation (executed in the problem state, and either the dual-address-space facility is not installed or the extraction-authority control is zero)

INSERT STORAGE KEY

ISK R_1,R_2 [RR]

'09'	R_1	R_2
0	8	12 15

The storage key for the 2K-byte block that is addressed by the contents of general register R_2 is inserted in general register R_1.

Bits 8-20 of general register R_2 designate a 2K-byte block in real 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.

When the storage-key 4K-byte-block facility is not installed, all blocks are double-key 4K-byte blocks, and the operation proceeds normally.

When the storage-key 4K-byte-block facility is installed, all blocks are single-key 4K-byte blocks, and the action taken depends on the setting of the storage-key-exception-control bit, bit 7 of control register 0. If the bit is zero, a special-operation exception is recognized. If the bit is one, the operation is performed on the single key for the 4K-byte block.

The address designating the storage block, being a real address, is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The execution of the instruction depends on whether the PSW specifies the EC or BC mode. In the EC mode, the seven-bit storage key is inserted in bit positions 24-30 of general register R_1, and bit 31 is set to zero. In the BC mode, bits 0-4 of the storage key are placed in bit positions 24-28 of that register, and bits 29-31 of the register are set to zeros. In both modes, the contents of bit positions 0-23 of the register remain unchanged.

Special Conditions

Bits 28-31 of general register R_2 must be zeros; otherwise, a specification exception is recognized.

When the storage-key 4K-byte-block facility is installed and the storage-key-exception-control bit (bit 7 of control register 0) is zero, a special-operation exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (address specified by general register R_2)
- Privileged operation
- Special operation
- Specification

INSERT STORAGE KEY EXTENDED

ISKE R_1,R_2 [RRE]

'B229'	////////	R_1	R_2
0	16	24	28 31

The storage key for the block that is addressed by the contents of general register R_2 is inserted in general register R_1.

Bits 16-23 of the instruction are ignored.

The contents of general register R_2 are treated as a 31-bit real address of a 4K-byte block in storage. Bits 1-19 of the register designate the 4K-byte block, and bits 0 and 20-31 of the register are ignored.

The address designating the storage block, being a real address, is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

When the storage-key 4K-byte-block facility is not installed, all blocks are double-key 4K-byte blocks. The key for the first 2K-byte block within the 4K-byte block designated by the instruction is called the low-order key. The key for the second 2K-byte block is called the high-order key. The contents of the low-order key are inserted, but with the resultant change bit being the OR of the change bits from the low-order and high-order keys. Similarly, the resultant reference bit is the OR of the reference bits from the low-order and high-order keys. The contents of the storage keys are not changed.

When the storage-key 4K-byte-block facility is installed, all blocks are single-key 4K-byte blocks, and the single key is inserted in the register.

The seven-bit storage key is inserted in bit positions 24-30 of general register R_1, and bit 31 is set to zero. The contents of bit positions 0-23 of the register remain unchanged. The operation is not dependent on whether the PSW specifies the EC or BC mode.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (address specified by general register R_2)
- Operation (if the storage-key-instruction-extension facility is not installed)
- Privileged operation

Programming Note

The results of the execution of INSERT STORAGE KEY EXTENDED for a double-key 4K-byte block may have intermediate values for the reference and change bits if there is a concurrent storage-key operation being executed on either key for the same double-key block by another CPU in the configuration.

INSERT VIRTUAL STORAGE KEY

IVSK R_1,R_2 [RRE]

'B223'	////////	R_1	R_2
0	16	24	28 31

The storage key for the location designated by the virtual address in general register R_2 is inserted in general register R_1.

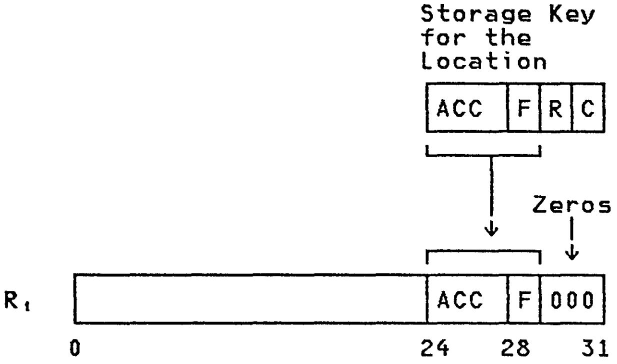
Bits 16-23 of the instruction are ignored.

Bits 8-31 of general register R_2 are used as a virtual address. Bits 0-7 of the register are ignored.

The address is a virtual address and is subject to the address-space-control bit, bit 16 of the current PSW. In the primary-space mode, the address is treated as a primary virtual address; in the secondary-space mode, the address is treated as a secondary virtual address. The reference to the storage key is not subject to a protection exception.

Bits 0-4 of the storage key, which are the access-control bits and the fetch-protection bit, are placed in bit positions 24-28 of general register R_1, with bits 29-31 set to zeros. The contents of bit positions 0-23 of the register remain unchanged. The change and reference bits in the storage key are not inspected. The change bit is not affected by the operation. The reference bit, depending on the model, may or may not be set to one as a result of the operation.

The following diagram shows the storage key and the register positions just described.



Special Conditions

The instruction must be executed with DAT on; otherwise, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

In the problem state, the extraction-authority control, bit 4 of control register 0, must be one; otherwise, a privileged-operation exception is recognized. In the supervisor state, the extraction-authority-control bit is not examined.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: INSERT VIRTUAL STORAGE KEY."

Condition Code: The code remains unchanged.

Program Exceptions:

Access (except for protection, address specified by general register R_2)
Operation (if the dual-address-space facility is not installed)
Privileged operation (extraction-authority control is zero in the problem state)
Special operation

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B.1	Operation exception if the dual-address-space facility is not installed.
7.B.2	Special-operation exception due to DAT being off.
8.	Privileged-operation exception due to extraction-authority control, bit 4 of control register 0, being zero.
9.	Access exceptions (except for protection) for address specified by general register R_2.

Priority of Execution: INSERT VIRTUAL STORAGE KEY

Programming Note

Since all bytes in a 2K-byte block are associated with the same page and the same storage key, bits 21-31 of general register R_2 effectively are ignored. When 4K-byte pages are used, the storage-key 4K-byte-block facility is installed, and all blocks are single-key 4K-byte blocks, then bits 20-31 of general register R_2 essentially are ignored.

INVALIDATE PAGE TABLE ENTRY

IPTE R_1,R_2 [RRE]

'B221'	////////	R_1	R_2
0	16	24	28 31

The designated page-table entry is invalidated, and the translation-lookaside buffers (TLBs) in all CPUs in the configuration are cleared of the associated entries.

Bits 16-23 of the instruction are ignored.

The contents of general register R_1 have the format of a segment-table entry with only the page-table origin used. The contents of general register R_2 have the format of a virtual address with only the page index used. The contents of fields that are not part of the page-table origin or page index are ignored. The translation format, contained in bit positions 8-12 of control register 0, specifies the mode for translation.

The contents of the general registers just described are as follows:

R_1

////////	Page-Table Origin	///
0	8	29 31

R_2 (for 64K-byte segments and 4K-byte pages)

////////////////	PX	/////////////
0	16	20 31

R_2 (for 64K-byte segments and 2K-byte pages)

////////////////	PX	////////////
0	16	21 31

R_2 (for 1M-byte segments and 4K-byte pages)

/////////////	PX	/////////////
0	12	20 31

R_2 (for 1M-byte segments and 2K-byte pages)

/////////////	PX	////////////
0	12	21 31

The page-table origin and the page index designate a page-table entry, following the dynamic-address-translation rules for page-table lookup. The address formed from these two components is a real address. The page-invalid bit of this page-table entry is set to one. During this procedure, no page-table-length check is made, and the page-table entry is not inspected for availability or format errors. Additionally, the page-frame real address (including the extended-addressing bits, when applicable) contained in the entry is not checked for an addressing exception.

The entire page-table entry is fetched concurrently from storage. Subsequently the byte containing the page-invalid bit is stored. The fetch access to the page-table entry is subject to key-controlled protection, and the store access is subject to key-controlled protection and low-address protection.

A serialization function is performed before the operation begins and again after the operation is completed. As is the case for all serialization operations, this serialization applies only to this CPU; other CPUs are not necessarily serialized.

If it is successful in setting the page-invalid bit to one, this CPU clears selected entries from its TLB and signals all CPUs in the configuration to clear selected entries from their TLBs. Each TLB is cleared of at least those entries that have been formed using all of the following:

- The translation format specified in bit positions 8-12 of control register 0 of the CPU executing the INVALIDATE PAGE TABLE ENTRY instruction
- The page-table origin designated by the first operand
- The page index designated by the second operand
- The page-frame real address (including the extended-addressing bits, when applicable) contained in the designated page-table entry

The execution of INVALIDATE PAGE TABLE ENTRY is not completed on the CPU which executes it until (1) all entries corresponding to the specified parameters have been cleared from the TLB on this CPU and (2) all other CPUs in the configuration have completed any storage accesses, including the updating of the change and reference bits, by using TLB entries corresponding to the specified parameters.

Special Conditions

When bit positions 8-12 of control register 0 contain an invalid code, a translation-specification exception is recognized. The exception is recognized regardless of whether DAT is on or off.

The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

Addressing (page-table entry)
Operation (if the extended facility is not installed)
Privileged operation
Protection (fetch and store, page-table entry, key-controlled protection, and low-address protection)
Translation specification (bits 8-12 in control register 0 only)

Programming Notes

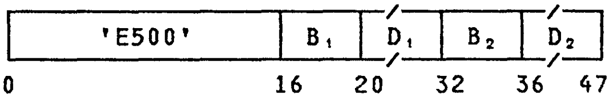
1. The selective clearing of entries may be implemented in different ways, depending on the model, and, in general, more entries may be cleared than the minimum number required. Some models may clear all entries which contain the designated page-frame real address. Others may clear all entries which contain the designated page index, and some implementations may clear precisely the minimum number of entries required. Therefore, in order for a program to operate on all models, the program should not take advantage of any properties obtained by a less selective clearing on a particular model.

2. The clearing of TLB entries may make use of the page-frame real address in the page-table entry. Therefore, if the page-table entry, when in the attached state, ever contained a page-frame real address that is different from the current value, copies of the previous values may remain in the TLB.

3. INVALIDATE PAGE TABLE ENTRY cannot be safely used to update a shared location in main storage if the possibility exists that another CPU or a channel may also be updating the location.

LOAD ADDRESS SPACE PARAMETERS

LASP $D_1(B_1),D_2(B_2)$ [SSE]



The contents of the doubleword at the first-operand location contain values to be loaded into control registers 3 and 4, including a secondary ASN and a primary ASN. Execution of the instruction consists in performing four major steps: PASN translation, SASN translation, SASN authorization, and control-register loading. Each of these steps may or may not be performed, depending on the outcome of certain tests and on the setting of bits 29-31 of the second-operand address. These steps, when successful, obtain additional values, which are loaded into control registers 1, 5, and 7. When the steps are not successful, no control registers are changed, and the reason is indicated in the condition code.

The doubleword first operand contains a PSW-key mask (PKM), a secondary ASN (SASN), an authorization index (AX), and a primary ASN (PASN). The primary ASN is translated by means of the ASN-translation tables to obtain a PSTD, LTD, and, optionally, an AX. The secondary ASN is translated by means of the ASN-translation tables to obtain an SSTD, and, optionally, an authority check is made to ensure that the new AX is authorized to establish the new SASN.

The doubleword at the first-operand location has the following format:

PKM-d	SASN-d	AX-d	PASN-d

```
0           16          32          48          63
```

The "d" stands for designated doubleword and is used to distinguish these fields from other fields with similar names which are referred to in the definition. The current contents of the corresponding fields in the control registers are referred to as PKM-old, SASN-old, etc. The updated contents of the control registers are referred to as PKM-new, SASN-new, etc.

The second-operand address is not used to address data; instead, the rightmost three bits are used to control portions of the operation. The remainder of the second-operand address is ignored. Bits 29-31 of the second-operand address are used as follows:

Bit	Function Specified in Second Operand	
	When Bit Is Zero	When Bit Is One
29	ASN translation performed only when new ASN and old ASN are different.	ASN translation performed.*
30	Use AX associated with PASN.	Use AX from first operand.
31	SASN authorization performed.*	SASN authorization not performed.

* SASN translation and SASN authorization are performed only when SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d, the SSTD is loaded from the PSTD, and no authorization is performed.

The operation of LOAD ADDRESS SPACE PARAMETERS is depicted in the figure "Execution of LOAD ADDRESS SPACE PARAMETERS."

PASN Translation

In the PASN translation process, the PASN-d is translated by means of the ASN first table and the ASN second table. The STD and LTD fields and, optionally, the AX field, obtained from the ASN-second-table entry are subsequently used to update the corresponding control registers.

When bit 29 of the second-operand address is one, PASN translation is always performed. When bit 29 is zero, PASN translation is performed only when PASN-d is not equal to PASN-old. When bit 29 is zero and PASN-d is equal to PASN-old, the PSTD-old and LTD-old are left unchanged in the control registers and become the PSTD-new and LTD-new, respectively. In this case, if bit 30 is zero, then the AX-old is left unchanged in the control register and becomes the AX-new.

The PASN translation follows the normal rules for ASN translation, except that the invalid bits, bit 0 in the ASN-first-table entry and bit 0 in the ASN-second-table entry, when ones, do not result in an ASN-translation exception, and the space-switch-event-control bit in the ASN-second-table entry, when one, does not result in a space-switch event. When either of the invalid bits is one, condition code 1 is set. When the ASN-second-table entry is valid and either the current space-switch-event-control bit in control register 1 is one or the space-switch-event-control bit in the ASN-second-table entry is one, condition code 3 is set. When condition code 1 or 3 is set, the control registers remain unchanged.

The contents of the AX, STD, and LTD fields in the ASN-second-table entry which is accessed as a result of the PASN translation are referred to as AX-p, STD-p, and LTD-p, respectively.

SASN Translation

In the SASN-translation process, the SASN-d is translated by means of the ASN first table and the ASN second table. The STD field obtained from the ASN-second-table entry is subsequently used to update the secondary-segment-table designation (SSTD) in control register 7. The ATO and ATL fields obtained are used in the SASN authorization, if it occurs.

SASN translation is performed only when SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d, the SSTD-new is set to the same value as PSTD-new. When SASN-d is equal to SASN-old, bit 29 (force ASN translation) is zero, and bit 31 (skip SASN authorization) is one, then SASN translation is not performed, and SSTD-old becomes SSTD-new.

The SASN translation follows the normal rules for ASN translation, except that the invalid bits, bit 0 in the ASN-first-table entry and bit 0 in the ASN-second-table entry, when ones, do not result in an ASN-translation exception. When either or both of the invalid bits are ones, condition code 2 is set, and the control registers remain unchanged.

The contents of the STD, ATO, and ATL fields in the ASN-second-table entry which is accessed as a result of the SASN translation are referred to as STD-s, ATO-s, and ATL-s, respectively.

SASN Authorization

SASN authorization is performed when bit 31 of the second-operand address is zero and SASN-d is not equal to PASN-d. When SASN-d is equal to PASN-d or when bit 31 of the second-operand address is one, SASN authorization is not performed.

SASN authorization is performed by using ATO-s, ATL-s, and the intended value for AX-new. When bit 30 of the second-operand address is zero and PASN translation was performed, the intended value for AX-new is AX-p. When bit 30 of that address is zero and PASN translation was not performed, the AX is not changed, and AX-new is the same as AX-old. When bit 30 of that address is one, the intended value for AX-new is AX-d. SASN authorization follows the rules for secondary authorization as described in the section "ASN-Authorization Process" in Chapter 3, "Storage." If the SASN is not authorized (that is, the authority-table length is exceeded, or the selected bit is zero), condition code 2 is set, and none of the control registers are updated.

Control-Register Loading

When the PASN-translation, SASN-translation, and SASN-authorization functions, if called for in the operation, are performed without encountering any exceptions, the operation is completed by replacing the contents of control registers 1, 3, 4, 5, and 7 with the new values, and condition code 0 is set. The control registers are loaded as follows:

The PSW-key-mask and SASN fields in control register 3 are replaced by the PKM-d and SASN-d fields from the first-operand location.

The PASN, bits 16-31 of control register 4, is replaced by the PASN-d field from the first-operand location.

The authorization index, bits 0-15 of control register 4, is replaced as follows:

- When bit 30 of the second-operand address is one, from AX-d.

- When bit 30 of the second-operand address is zero and PASN translation is performed, from AX-p.

- When bit 30 of the second-operand address is zero and PASN translation is not performed, the authorization index is not changed.

The primary-segment-table designation in control register 1 and the linkage-table designation in control register 5 are replaced as follows:

- When PASN translation is performed, the primary-segment-table designation in control register 1 and the linkage-table designation in control register 5 are replaced from the STD-p and LTD-p fields, respectively, which are obtained during PASN translation.

- When PASN translation is not performed, the primary-segment-table-designation and linkage-table-designation fields remain unchanged.

The contents of the secondary-segment-table designation in control register 7 are replaced as follows:

- When SASN-d equals PASN-d, by the new contents of control register 1, the primary-segment-table designation.

- When SASN translation is performed, by the contents of the STD-s.

When SASN-d does not equal PASN-d and SASN translation is not performed, the secondary-segment-table designation remains unchanged.

Other Condition-Code Settings

When PASN translation is called for and cannot be completed because bit 0 is one in either the ASN-first-table or the ASN-second-table entries, condition code 1 is set, and the control registers are not changed.

When (1) PASN translation is called for and completed and (2) either the current space-switch-event-control bit, bit 31 of control register 1 is one or the space-switch-event-control bit in the ASN-second-table entry is one, condition code 3 is set, and the control registers are not changed.

When SASN translation is called for and the translation cannot be completed because bit 0 is one in either the ASN-first-table or ASN-second-table entries, or because SASN authorization is called for and the SASN is not authorized, condition code 2 is set, and the control registers are not changed.

Special Conditions

The instruction can be executed only when the ASN-translation control, bit 12 of control register 14, is one. If the ASN-translation-control bit is zero, a special-operation exception is recognized.

The first operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

The figures "Summary of Actions: LOAD ADDRESS SPACE PARAMETERS" and "Priority of Execution: LOAD ADDRESS SPACE PARAMETERS" summarize the functions of the instruction and the priority of recognition of exceptions and condition codes.

Resulting Condition Code:

0 Translation and authorization complete; parameters loaded
1 Primary ASN not available; parameters not loaded
2 Secondary ASN not available or not authorized; parameters not loaded
3 Space-switch event specified; parameters not loaded

Program Exceptions:

Access (fetch, operand 1)
Addressing (ASN-first-table entry, ASN-second-table entry, authority-table entry)
ASN-translation specification
Operation (if the dual-address-space facility is not installed)
Privileged operation
Special operation
Specification

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second and third instruction halfwords.

7.B.1 Operation exception if the dual-address-space facility is not installed.

7.B.2 Privileged-operation exception.

7.B.3 Special-operation exception due to the ASN-translation control, bit 12 of control register 14, being zero.

8. Specification exception.

9. Access exceptions for the first operand.

10. Execution of PASN translation (when performed).

10.1 Addressing exception for access to ASN-first-table entry.

10.2 Condition code 1 due to I bit (bit 0) in ASN-first-table entry being one.

10.3 ASN-translation-specification exception due to invalid ones (bits 1-7, 28-31) in ASN-first-table entry.

10.4 Addressing exception for access to ASN-second-table entry.

10.5 Condition code 1 due to I bit (bit 0) in ASN-second-table entry being one.

10.6 ASN-translation-specification exception due to invalid ones (bits 1-7, 30, 31, 60-63, 97-103) in ASN-second-table entry.

10.7 Condition code 3 due to either the old or new space-switch-event-control bit being one.

11. Execution of SASN translation (when performed).

11.1 Addressing exception for access to ASN-first-table entry.

11.2 Condition code 2 due to I bit (bit 0) in ASN-first-table entry being one.

11.3 ASN-translation-specification exception due to invalid ones (bits 1-7, 28-31) in ASN-first-table entry.

11.4 Addressing exception for access to ASN-second-table entry.

11.5 Condition code 2 due to I bit (bit 0) in ASN-second-table entry being one.

11.6 ASN-translation-specification exception due to invalid ones (bits 1-7, 30, 31, 60-63, 97-103) in ASN-second-table entry.

12. Execution of secondary authorization (when performed).

12.1 Condition code 2 due to authority-table entry being outside table.

12.2 Addressing exception for access to authority-table entry.

12.3 Condition code 2 due to S bit in authority-table entry being zero.

Priority of Execution: LOAD ADDRESS SPACE PARAMETERS

PASN-d Equals PASN-old	Second-Operand-Address Bits*		PASN Translation Performed	Result Field					
	29	30		PSTD-new	AX-new	LTD-new	PKM-new	SASN-new	PASN-new
Yes	0	0	No	PSTD-old	AX-old	LTD-old	PKM-d	SASN-d	PASN-d
Yes	0	1	No	PSTD-old	AX-d	LTD-old	PKM-d	SASN-d	PASN-d
Yes	1	0	Yes	STD-p	AX-p	LTD-p	PKM-d	SASN-d	PASN-d
Yes	1	1	Yes	STD-p	AX-d	LTD-p	PKM-d	SASN-d	PASN-d
No	-	0	Yes	STD-p	AX-p	LTD-p	PKM-d	SASN-d	PASN-d
No	-	1	Yes	STD-p	AX-d	LTD-p	PKM-d	SASN-d	PASN-d

Summary of Actions: LOAD ADDRESS SPACE PARAMETERS (Part 1 of 2)

SASN-d Equals PASN-d	SASN-d Equals SASN-old	Second-Operand-Address Bits*		SASN Translation Performed	SASN Authorization Performed#	Result Field SSTD-new
		29	31			
Yes	-	-	-	No	No	PSTD-new
No	Yes	0	1	No	No	SSTD-old
No	Yes	1	1	Yes	No	STD-s
No	Yes	-	0	Yes	Yes	STD-s
No	No	-	1	Yes	No	STD-s
No	No	-	0	Yes	Yes	STD-s

Explanation:

- Action in this case is the same regardless of the outcome of this comparison or of the setting of this bit.

* Second-operand-address bits:
29 Force ASN translation.
30 Use AX from first operand.
31 Skip secondary authority test.

SASN authorization is performed using ATO-s, ATL-s, and AX-new.

Summary of Actions: LOAD ADDRESS SPACE PARAMETERS (Part 2 of 2)

Programming Notes

1. Bits 29 and 31 in the second-operand address are intended primarily to provide improved performance for those cases where the associated action is unnecessary.

   When bit 29 is set to zero, the action of the instruction is based on the assumption that the current values for PSTD-old, LTD-old, and AX-old are consistent with PASN-old and that SSTD-old is consistent with SASN-old. When this is not the case, bit 29 should be set to one.

   Bit 31, when one, eliminates the SASN-authorization test. The program may be able to determine in certain cases that the SASN is authorized, either because of prior use or because the AX being loaded is authorized to access all address spaces.

2. The SASN-translation and SASN-authorization steps are not performed when SASN-d is equal to PASN-d. This is consistent with the action in SET SECONDARY ASN to current primary (SSAR-cp), which does not perform the translation or ASN authorization.

3. See the figure "Summary of Abbreviations for LOAD ADDRESS SPACE PARAMETERS" for a listing of abbreviations used in this instruction description.

Control-Register Number.Bit	Abbreviation for	
	Previous Contents	Subsequent Contents
1.0-31	PSTD-old	PSTD-new
3.0-15	PKM-old	PKM-new
3.16-31	SASN-old	SASN-new
4.0-15	AX-old	AX-new
4.16-31	PASN-old	PASN-new
5.0-31	LTD-old	LTD-new
7.0-31	SSTD-old	SSTD-new

First-Operand Bit Positions	Abbreviation
0-15	PKM-d
16-31	SASN-d
32-47	AX-d
48-63	PASN-d

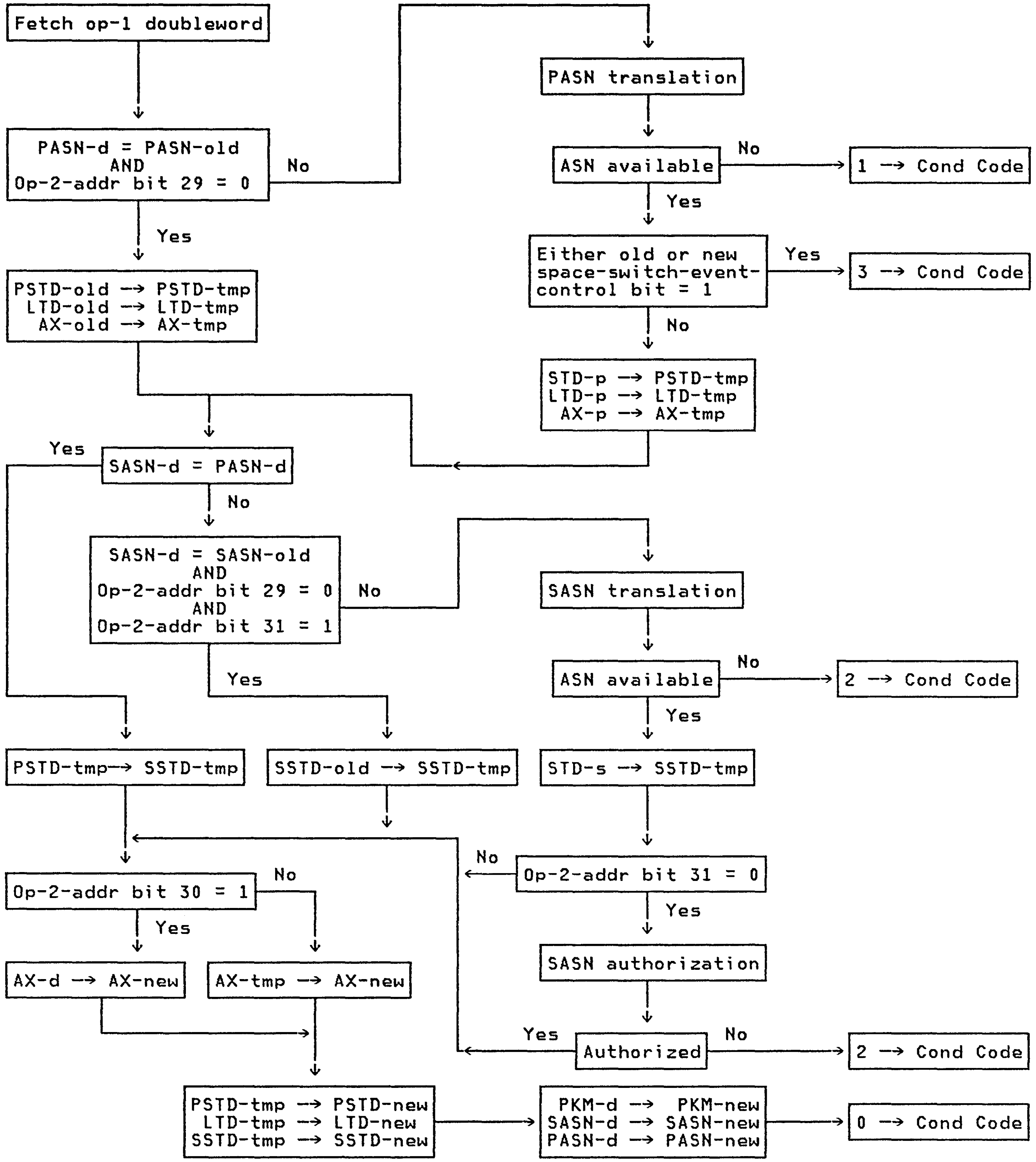
Field in ASN-Second-Table Entry	Abbreviation Used for the Field When Accessed as Part of	
	PASN Translation	SASN Translation
8-29	-	ATO-s
32-47	AX-p	-
48-59	-	ATL-s
64-95	STD-p	STD-s
96-127	LTD-p	-[1]

Explanation:

- The field is not used in this case.

[1] Although the field is not used, bits 97-103 are tested for zeros.

Summary of Abbreviations for LOAD ADDRESS SPACE PARAMETERS



Execution of LOAD ADDRESS SPACE PARAMETERS

LOAD CONTROL

LCTL R_1,R_3,D_2(B_2) [RS]

'B7'	R_1	R_3	B_2	D_2

0 8 12 16 20 31

The set of control registers starting with control register R_1 and ending with control register R_3 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 register numbers, starting with control register R_1 and continuing up to and including control register R_3, with control register 0 following control register 15. The second operand remains unchanged.

Special Conditions

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)
Privileged operation
Specification

Programming Notes

1. To ensure that existing programs operate correctly if and when new facilities using additional control-register positions are defined, only zeros should be loaded in unassigned control-register positions.

2. Loading of control registers on some models may require a significant amount of time. This is particularly true for changes in significant parameters.

   For example, the TLB may be cleared of entries as a result of changing the translation parameters in control register 0 or as a result of changing or enabling the program-event-recording parameters in control registers 9-11. Where possible, the program should avoid unnecessary loading of control registers. In loading control registers 9-11, most models attempt to optimize for the case when the bits of control register 9 are zeros.

   As another example, the translation format, bits 8-12 of control register 0, is initialized to all zeros by initial CPU reset. An all-zero value is an invalid translation format, and, on some models, results in purging the TLB even though DAT may be off. Thus, the program should avoid loading invalid values for this field.

LOAD PSW

LPSW D_2(B_2) [S]

'82'	/////////	B_2	D_2

0 8 16 20 31

The current PSW is replaced by the contents of the doubleword at the location designated by the second-operand address.

Bits 8-15 of the instruction are ignored.

If the new PSW specifies the BC mode, information in bit positions 16-33 of the new PSW is not retained as the PSW is loaded. When the PSW is subsequently stored, these bit positions contain the new interruption code and the instruction-length code.

A serialization and checkpoint-synchronization function is performed before or after the operand is fetched and again after the operation is completed.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The value which is to be loaded by the instruction is not checked for validity before it is loaded. However, immediately after loading, a specification exception is recognized and a program interruption occurs when any of the following is true for the newly loaded PSW:

- The EC mode is specified (PSW bit 12 is one) in a CPU that does not have the translation facility installed.

- Bit position 16 of an EC-mode PSW is one, and DAS is not installed.

- A one is introduced into an unassigned bit position of an EC-mode PSW (that is, any of bit positions 0, 2-4, 17, or 24-39).

In these cases, the operation is completed, and the resulting instruction-length code is zero.

The test for a specification exception after the PSW is loaded is described in the section "Early Exception Recognition" in Chapter 6, Interruptions." It may be considered as occurring early in the process of preparing to execute the subsequent instruction.

The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code is set as specified in the new PSW loaded.

Program Exceptions:

Access (fetch, operand 2)
Privileged operation
Specification

LOAD REAL ADDRESS

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 virtual address is placed in general register R_1.

The virtual address specified by the X_2, B_2, and D_2 fields is translated by means of the dynamic-address-translation facility, regardless of whether the current PSW specifies EC or BC mode, and regardless of whether DAT is on or off.

When DAS is not installed, the translation is performed by using the current contents of control registers 0 and 1. When DAS is installed, the translation is performed by using the current translation format in control register 0 and the segment-table designation in either control register 1 or 7. Control register 1 is used if the current PSW specifies BC mode or specifies EC mode with bit 16 set to zero. Control register 7 is used if the current PSW specifies EC mode with bit 16 set to one.

The translation is performed without the use of the translation-lookaside buffer (TLB). Sufficient zeros are appended on the left of the resultant real address to produce a 32-bit result, which is then placed in general register R_1. The translated address is not inspected for boundary alignment or for addressing or protection exceptions.

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 general register R_1. 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 general register R_1. When either the segment-table entry or the page-table entry is outside the table, condition code 3 is set, and general register R_1 contains the real address of the entry that would have been fetched if the length violation had not occurred. In all these cases, sufficient zeros are appended on the left of the resultant real address to produce a 32-bit result, and the 32-bit result is placed in the register.

Special Conditions

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 the I bit with a value of zero and has a format error.

The operation is suppressed on all addressing exceptions.

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 exceeded

Program Exceptions:

Addressing (segment-table entry or page-table entry)
Operation (if the translation facility is not installed)
Privileged operation
Translation specification

Programming Note

Caution must be exercised in the use of LOAD REAL ADDRESS in a multiprocessing configuration. Since INVALIDATE PAGE TABLE ENTRY may set the I bit in storage to one before causing the corresponding entries in TLBs of other CPUs to be cleared, the simultaneous execution of LOAD REAL ADDRESS on this CPU and INVALIDATE PAGE TABLE ENTRY on another CPU may produce inconsistent results. Because LOAD REAL ADDRESS accesses the tables in storage, the page-table entry may appear to be invalid (condition code 2) even though the corresponding TLB entry has not yet been cleared, and the TLB entry may remain in the TLB until the completion of INVALIDATE PAGE TABLE ENTRY on the other CPU. There is no guaranteed limit to the number of instructions which may occur between the completion of LOAD REAL ADDRESS and the TLB being cleared of the entry.

MOVE TO PRIMARY

MVCP $D_1(R_1,B_1),D_2(B_2),R_3$ [SS]

'DA'	R_1	R_3	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

MOVE TO SECONDARY

MVCS $D_1(R_1,B_1),D_2(B_2),R_3$ [SS]

'DB'	R_1	R_3	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The first operand is replaced by the second operand. One operand is in the primary address space, and the other is in the secondary address space. The accesses to the operand in the primary space are performed by using the PSW key; the accesses to the operand in the secondary space are performed by using the key specified by the third operand.

The addresses of the first and second operands are virtual, one operand address being translated by means of the primary segment-table designation and the other by means of the secondary segment-table designation. Operand-address translation is performed by ignoring the state of the address-space-control bit in the current PSW.

For MOVE TO PRIMARY, movement is to the primary space from the secondary space. The first-operand address is translated by using the primary segment table, and the second-operand address is translated by using the secondary segment table.

For MOVE TO SECONDARY, movement is to the secondary space from the primary space. The first-operand address is translated by using the secondary segment table, and the second-operand address is translated by using the primary segment table.

Bit positions 24-27 of general register R_3 are used as the secondary-space access key. Bit positions 0-23 and 28-31 of the register are ignored.

The contents of general register R_1 are a 32-bit unsigned value called the true length.

The contents of the general registers just described are as follows:

	True Length	
R_1	0	31

	//////////////////////////	Key	////
R_3	0	24	28 31

The first and second operands are the same length, called the effective length. The effective length is equal to the true length, or 256, whichever is less. Access exceptions for the first and second operands are recognized only for that portion of the operand within the effective length. When the effective length is zero, no access exceptions are recognized for the first and second operands, and no movement takes place.

Each storage operand is processed left to right. The storage-operand-consistency rules are the same as for MOVE (MVC), except that when the operands overlap in real storage, the use of the common real-storage locations is not necessarily recognized.

As part of the execution of the instruction, the value of the true length is used to set the condition code. If the true length is 256 or less, including zero, the true length and effective length are equal, and condition code 0 is set. If the true length is greater than 256, the effective length is 256, and condition code 3 is set.

For both MOVE TO PRIMARY and MOVE TO SECONDARY, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Special Conditions

Since the secondary space is accessed, the operation is performed only when the secondary-space control, bit 5 of control register 0, is one and DAT is on. When either the secondary-space control is zero or DAT is off, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

In the problem state, the operation is performed only if the secondary-space access key is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the secondary-space access key is valid.

The priority of the recognition of exceptions and condition codes is shown in the figure "Priority of Execution: MOVE TO PRIMARY and MOVE TO SECONDARY."

Resulting Condition Code:

0	True length less than or equal to 256
1	--
2	--
3	True length greater than 256

Program Exceptions:

Access (fetch, primary virtual address, operand 2, MVCS; fetch, secondary virtual address, operand 2, MVCP; store, secondary virtual address, operand 1, MVCS; store, primary virtual address, operand 1, MVCP)
Operation (if the dual-address-space facility is not installed)
Privileged operation (selected PSW-key-mask bit is zero in the problem state)
Special operation

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second and third instruction halfwords.
7.B.1	Operation exception if the dual-address-space facility is not installed.
7.B.2	Special-operation exception due to the secondary-space control, bit 5 of control register 0, being zero or to DAT being off.
8.	Privileged-operation exception due to selected PSW-key-mask bit being zero in the problem state.
9.	Completion due to length zero.
10.	Access exceptions for operands.

Priority of Execution: MOVE TO PRIMARY and MOVE TO SECONDARY

Programming Notes

1. MOVE TO PRIMARY and MOVE TO SECONDARY can be used in a loop to move a variable number of bytes of any length. See the programming note under MOVE WITH KEY.

2. MOVE TO PRIMARY and MOVE TO SECONDARY should be used only when movement is between different address spaces. The performance of these instructions on most models may be significantly slower than MOVE WITH KEY, MOVE (MVC), or MOVE LONG. In addition, the definition of overlapping operands for MOVE TO PRIMARY and MOVE TO SECONDARY is not compatible with the more precise definitions for MOVE (MVC), MOVE WITH KEY, or MOVE LONG.

MOVE WITH KEY

MVCK $D_1(R_1,B_1),D_2(B_2),R_3$ [SS]

'D9'	R_1	R_3	B_1	D_1	B_2	D_2
0	8	12	16	20	32	36 47

The first operand is replaced by the second operand. The fetch accesses to the second-operand location are performed by using the key specified in the third operand, and the store accesses to the first-operand location are performed by using the PSW key.

Bit positions 24-27 of general register R_3 are used as the source access key. Bit positions 0-23 and 28-31 of the register are ignored.

The contents of general register R_1 are a 32-bit unsigned value called the true length.

The contents of the general registers just described are as follows:

R_1	True Length
	0 31

R_3	/////////////////////////	Key	////
	0	24	28 31

The first and second operands are the same length, called the effective length. The effective length is equal to the true length, or 256, whichever is less. Access exceptions for the first and second operands are recognized only for that portion of the operand within the effective length. When the effective length is zero, no access exceptions are recognized for the first and second operands, and no movement takes place.

Each storage operand is processed left to right. When the storage operands overlap, the result is obtained as if the operands were processed one byte at a time and each result byte were stored immediately after the necessary operand byte was fetched. The storage-operand-consistency rules are the same as for the MOVE (MVC) instruction.

As part of the execution of the instruction, the value of the true length is used to set the condition code. If the true length is 256 or less, including zero, the true length and effective length are equal, and condition code 0 is set. If the true length is greater than 256, the effective length is 256, and condition code 3 is set.

Special Conditions

In the problem state, the operation is performed only if the source access key is valid, that is, if the corresponding PSW-key-mask bit in control register 3 is one. Otherwise, a privileged-operation exception is recognized. In the supervisor state, any value for the source access key is valid.

The priority of the recognition of exceptions and condition codes is shown in the figure "Priority of Execution: MOVE WITH KEY Instruction."

Resulting Condition Code:

0 True length less than or equal to 256
1 --
2 --
3 True length greater than 256

Program Exceptions:

Access (fetch, operand 2; store, operand 1)
Privileged operation (selected PSW-key-mask bit is zero in the problem state)
Operation (if the dual-address-space facility is not installed)

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second and third instruction halfwords.
7.B	Operation exception if the dual-address-space facility is not installed.
8.	Privileged-operation exception due to selected PSW-key-mask bit being zero in the problem state.
9.	Completion due to length zero.
10.	Access exceptions for operands.

Priority of Execution: MOVE WITH KEY

Programming Notes

1. MOVE WITH KEY can be used in a loop to move a variable number of bytes of any length, as follows:

```
        LA      RW,256
LOOP    MVCK    D1(R1,B1),D2(B2),R3
        BC      8,END
        AR      B1,RW
        AR      B2,RW
        SR      R1,RW
        B       LOOP
END
```

2. The performance of MOVE WITH KEY on most models may be significantly slower than that of the MOVE (MVC) and MOVE LONG instructions. Therefore, MOVE WITH KEY should not be used if the key of the source and the target are the same.

PROGRAM CALL

PC $D_2(B_2)$ [S]

```
┌──────────────────┬─────┬─────────────┐
│      'B218'      │ B2  │     D2      │
└──────────────────┴─────┴─────────────┘
0                  16    20           31
```

A two-level lookup is performed to locate an entry-table entry (ETE). The ETE contains an authorization-key mask; an ASN; an entry parameter, which is loaded into general register 4; and information to update the PSW-key mask in control register 3 and to replace the problem-state bit and instruction address in the PSW. The original contents of the control-register and the PSW fields are saved in general registers 3 and 14.

The ETE also causes a space-switching operation to occur if it specifies a nonzero ASN. When the ETE specifies a zero ASN, the operation is called PROGRAM CALL to current primary (PC-cp); when the ETE specifies a nonzero ASN, the operation is called PROGRAM CALL with space switching (PC-ss). When space switching is specified, the new PASN is loaded into control register 4 from the ETE and is used in a two-level lookup to locate an ASN-second-table entry (ASTE). From this ASTE, a new PSTD, AX, and LTD are loaded into control registers 1, 4, and 5, respectively. Whether or not space switching is specified, the previous PASN and PSTD are placed in the SASN and SSTD, respectively, and the previous PASN is saved in general register 3.

PROGRAM CALL PC-Number Translation

The second-operand address is not used to address data; instead, the rightmost 20 bits of the address are used as a PC number and have the following format:

Second-Operand Address

```
             ┌────────PC Number────────┐
┌────────────┬─────────────┬───────────┐
│////////////│     LX      │    EX     │
└────────────┴─────────────┴───────────┘
0            12            24         31
```

Linkage Index (LX): Bits 12-23 of the second-operand address are the linkage index and are used to select an entry from the linkage table designated by the linkage-table designation in control register 5.

Entry Index (EX): Bits 24-31 of the second-operand address are the entry index and are used to select an entry from the entry table designated by the linkage-table entry.

Bits 0-11 of the second-operand address are ignored.

The linkage-table and entry-table lookup process is depicted in part 1 of the figure "Execution of PROGRAM CALL." The detailed definition for this table-lookup process is in the section "PC-Number Translation" in Chapter 5, "Program Execution." The entry-table entry has the following format:

```
┌──────────┬──────────┬──────┬──────────┬─┐
│   AKM    │   ASN    │ 0-0  │    IA    │P│
└──────────┴──────────┴──────┴──────────┴─┘
0          16         32     40          63

┌────────────────────┬──────────┬─────────┐
│        PARM        │   EKM    │/////////│
└────────────────────┴──────────┴─────────┘
64                   96         112     127
```

LTE bits 1-7 and ETE bits 32-39 must be zeros; otherwise, a PC-translation-specification exception is recognized.

After the entry-table entry has been fetched, if the current PSW specifies the problem state, the current PSW-key mask in control register 3 is tested against the AKM field in the entry-table entry to determine whether the program is authorized to access this entry. The AKM and PSW-key mask are ANDed, and if the result is zero, a privileged-operation exception is recognized. When PROGRAM CALL is executed in the supervisor state, the AKM field is ignored.

If the result of the AND of the AKM and the PSW-key mask is not zero, or if the CPU is in the supervisor state, the execution of the instruction continues.

The PSW-key mask, bits 0-15 of control register 3, is placed in bit positions 0-15 of general register 3, and the current PASN, bits 16-31 of control register 4, is placed in bit positions 16-31 of general register 3.

The current PSTD, bits 0-31 of control register 1, is placed in control register 7 to become the current SSTD.

The current PASN, bits 16-31 of control register 4, is placed in bit positions 16-31 of control register 3 to become the current SASN.

Bits 40-62 of the current PSW (the updated instruction address) are placed in bit positions 8-30 of general register 14. Bit 15 of the PSW (the problem-state bit) is placed in bit position 31 of general register 14. Bits 0-7 of general register 14 are set to zeros.

Bits 40-62 of the ETE, with a rightmost zero appended, are placed in PSW bit positions 40-63 (the instruction address). Bit 63 of the ETE is placed in PSW bit position 15 (the problem-state bit).

Bits 64-95 of the ETE (the entry parameter) are loaded into general register 4.

Bits 96-111 of the ETE (the EKM) are ORed with the PSW-key mask, bits 0-15 of control register 3, and the result replaces the PSW-key mask in control register 3.

PROGRAM CALL to Current Primary (PC-cp)

If bits 16-31 of the ETE (the ASN) are zeros, a PROGRAM CALL to current primary (PC-cp) is specified, and the operation is completed after performing those actions as described above.

The PC-cp operation is depicted in parts 1 and 2 of the figure "Execution of PROGRAM CALL."

PROGRAM CALL with Space Switching (PC-ss)

If the ASN in the ETE is nonzero, a PROGRAM CALL with space switching (PC-ss) instruction is specified, and the ASN is translated by means of a two-level table lookup.

The PC-ss operation is depicted in parts 1, 2 and 3 of the figure "Execution of PROGRAM CALL." The PC-ss operation is completed as follows:

Bits 16-25 of the ETE are used as a 10-bit AFX to index into the ASN first table, and bits 26-31 are used as a six-bit ASX to index into the ASN second table specified by the AFX. The ASN table-lookup process is described in the section "ASN Translation" in Chapter 3, "Storage." The exceptions associated with ASN translation are collectively called ASN-translation exceptions. These exceptions and their priority are described in Chapter 6, "Interruptions."

Bits 16-31 of the entry-table entry are placed in bit positions 16-31 of control register 4 as the new PASN.

Bits 64-95 of the ASN-second-table entry (the STD) are loaded into control register 1 as the new PSTD.

Bits 32-47 of the ASN-second-table entry (the AX) are loaded into bit positions 0-15 of control register 4 as the new authorization index.

Bits 96-127 of the ASN-second-table entry (the LTD) are loaded into control register 5 as the new linkage-table designation.

For both the PC-cp and PC-ss operations, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Special Conditions

The instruction can be executed only when the CPU is in primary-space mode and the subsystem-linkage control, bit 0 of control register 5, is one. If the CPU is in real mode or secondary-space mode, or if the subsystem-linkage control is zero, a special-operation exception is recognized. In addition, the PC-ss instruction can be executed only when the ASN-translation control, bit 12 of control register 14, is one. If PC-ss is attempted with the ASN-translation control zero, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

When, for PC-ss, the space-switch-event-control bit, bit 31 of control register 1, is one either before or after the execution of the instruction, a space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PC-ss operation if a PER event is reported.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: PROGRAM CALL."

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (linkage-table entry or entry-table entry)
- ASN translation (PC-ss only)
- EX translation
- LX translation
- Operation (if the dual-address-space facility is not installed)
- PC-translation specification
- Privileged operation (AND of AKM and PSW-key mask is zero in the problem state)
- Space-switch event (PC-ss only)
- Special operation
- Trace

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B.1 Operation exception if the dual-address-space facility is not installed.

7.B.2 Special-operation exception due to DAT being off, the CPU being in secondary-space mode, or the subsystem-linkage-control bit in control register 5 being zero.

8.A Trace exceptions.

8.B.1 LX-translation exception due to linkage-table entry being outside table.

8.B.2 Addressing exception for access to linkage-table entry.

8.B.3 LX-translation exception due to I bit (bit 0) in linkage-table entry being one.

8.B.4 PC-translation-specification exception due to invalid ones (bits 1-7) in linkage-table entry.

8.B.5 EX-translation exception due to entry-table entry being outside table.

8.B.6 Addressing exception for access to entry-table entry.

8.B.7 PC-translation-specification exception due to invalid ones (bits 32-39) in entry-table entry.

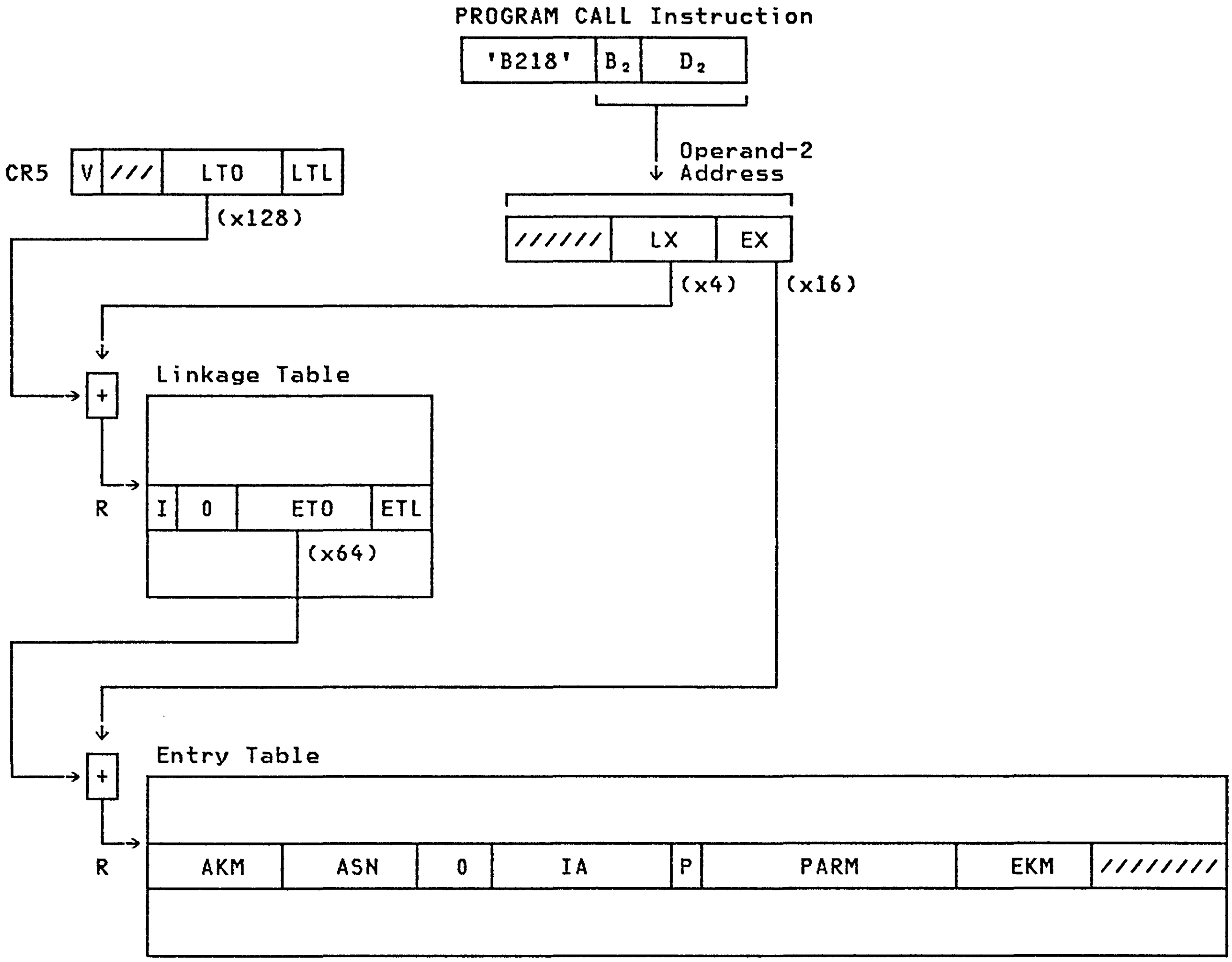
8.B.8 Privileged-operation exception due to a zero result from ANDing PSW-key mask and AKM in the problem state.

8.B.9 Special-operation exception due to the ASN-translation control, bit 12 of control register 14, being zero (PC-ss only).

8.B.10 ASN-translation exceptions (PC-ss only).

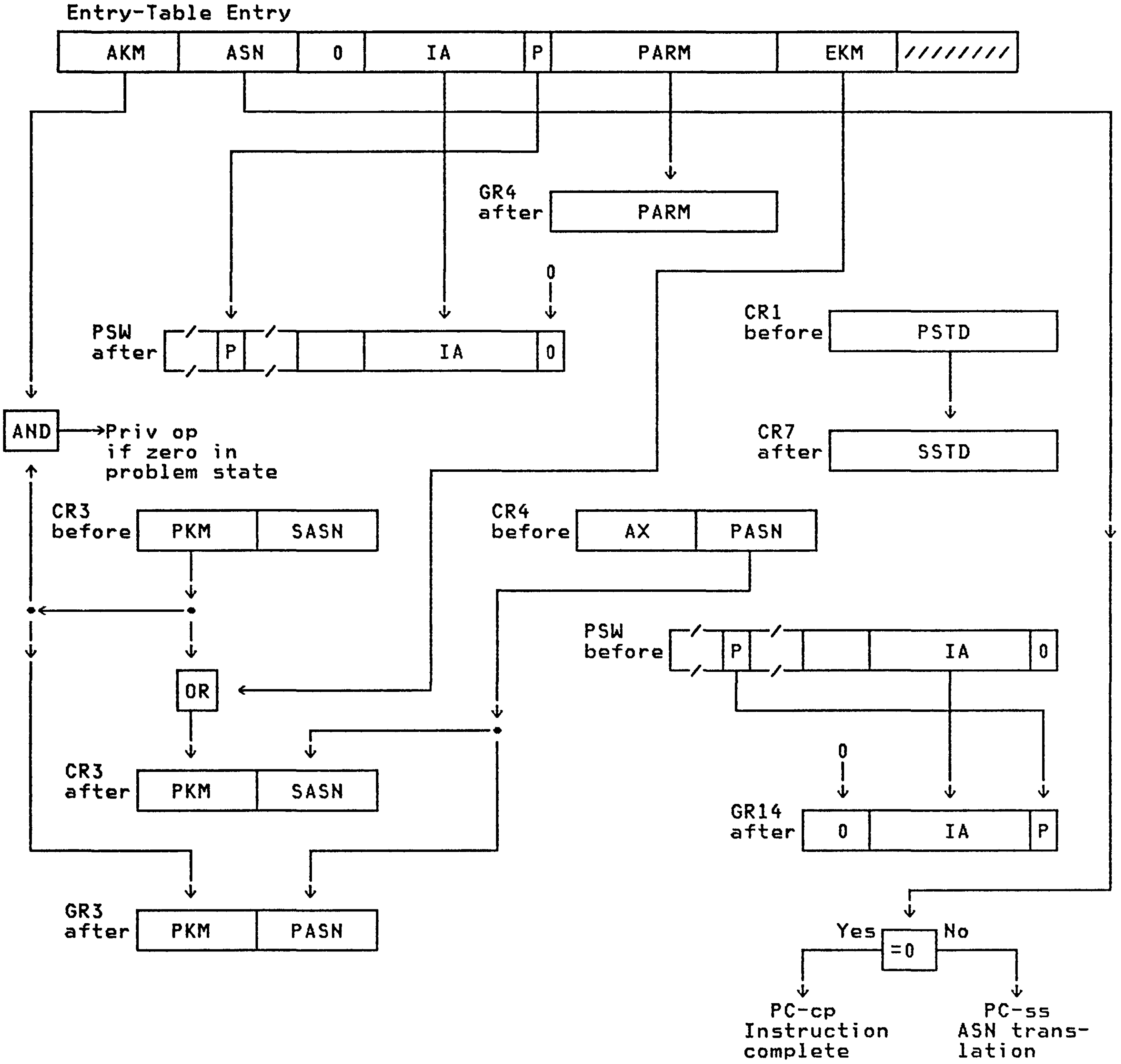
9. Space-switch event (PC-ss only).

Priority of Execution: PROGRAM CALL

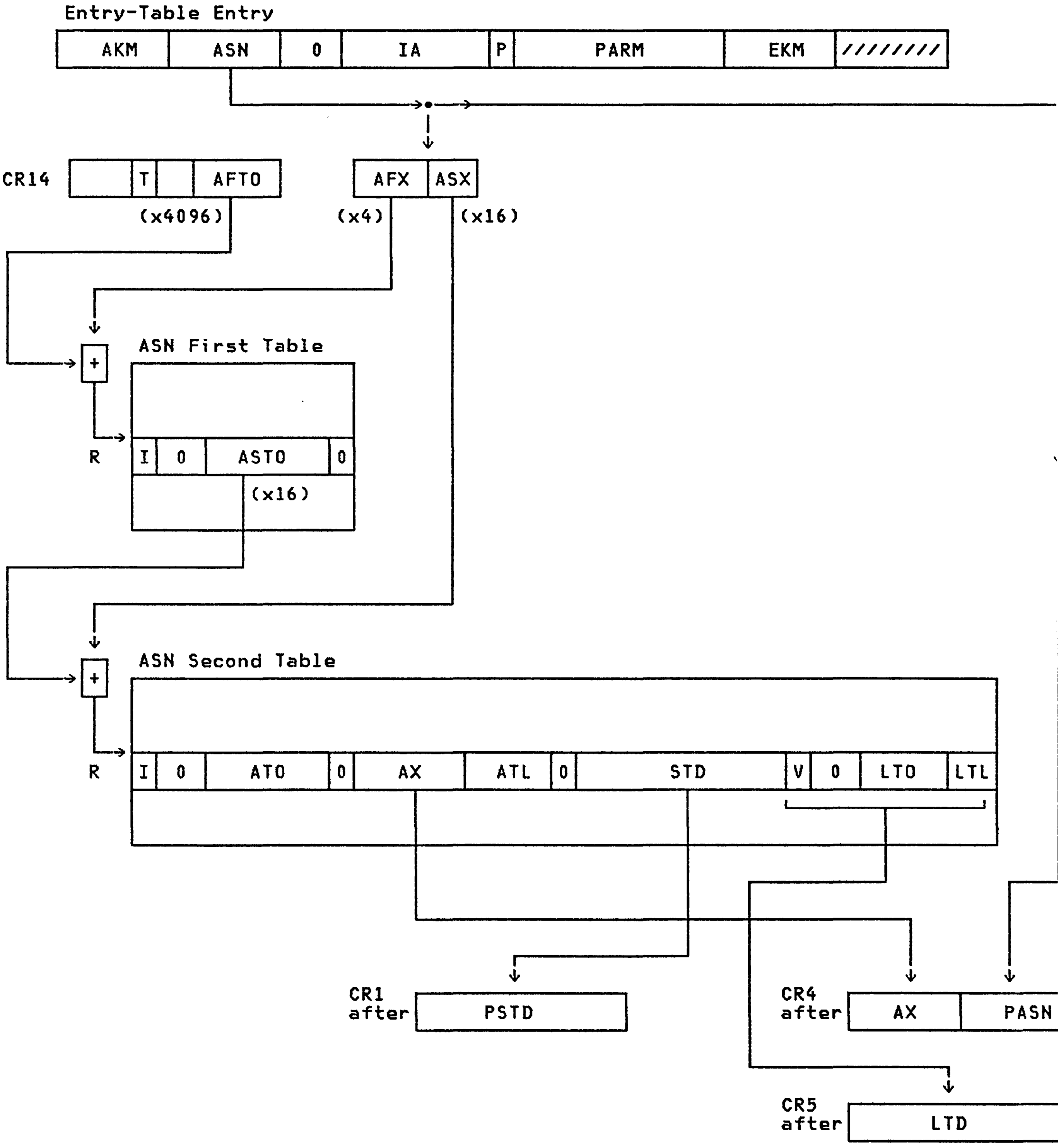


R: Address is real

Execution of PROGRAM CALL (Part 1 of 3): PC-Number Translation



Execution of PROGRAM CALL (Part 2 of 3): PC-cp and PC-ss



R: Address is real

Execution of PROGRAM CALL (Part 3 of 3): ASN Translation for PC-ss

PROGRAM TRANSFER

PT R_1,R_2 [RRE]

'B228'	////////	R_1	R_2
0	16	24	28 31

The contents of general register R_1 are used as the new values for the PSW-key mask, the PASN, and the SASN. The contents of general register R_2 are used as the new values for the problem-state bit and instruction address in the current PSW.

Bits 16-23 of the instruction are ignored.

General registers R_1 and R_2 have the following format:

	PSW-Key Mask	ASN
R_1	0	16 31

	00000000	Instruction Address	P
R_2	0	8	31

When the contents of bit positions 16-31 of general register R_1 are equal to the current PASN, the operation is called PROGRAM TRANSFER to current primary (PT-cp); when the fields are not equal, the operation is called PROGRAM TRANSFER with space switching (PT-ss).

The contents of general register R_2 are used to update the problem-state bit and the instruction address of the current PSW. Bit 31 of general register R_2 is placed in the problem-state bit position, PSW bit position 15, unless the operation would cause PSW bit 15 to change from one to zero (problem state to supervisor state). If such a change would occur, a privileged-operation exception is recognized. Bits 8-30 of general register R_2 replace the instruction address, bits 40-62, of the current PSW. Bit 63 of the PSW is set to zero.

Bits 0-15 of general register R_1 are ANDed with the PSW-key mask, bits 0-15 of control register 3, and the result replaces the contents of the PSW-key mask.

In both the PT-ss and PT-cp instructions, the ASN specified by bits 16-31 of general register R_1 replaces the SASN in control register 3, and the SSTD in control register 7 is replaced by the final contents of control register 1.

PROGRAM TRANSFER to Current Primary (PT-cp)

The PROGRAM TRANSFER to current primary (PT-cp) operation is depicted in part 1 of the figure "Execution of PROGRAM TRANSFER." On a PT-cp operation, the operation is completed when the common portion of the PROGRAM TRANSFER operation, described above, is completed. The authorization index, PASN, primary STD, and linkage-table designation are not changed by PT-cp.

PROGRAM TRANSFER with Space Switching (PT-ss)

If the ASN in bits 16-31 of general register R_1 is not equal to the current PASN, a PROGRAM TRANSFER with space switching (PT-ss) is specified, and the ASN is translated by means of a two-level table lookup.

The PT-ss operation is depicted in parts 1 and 2 of the figure "Execution of PROGRAM TRANSFER." The PT-ss operation is completed as follows:

For a PT-ss, the contents of bit positions 16-31 of general register R_1 are used as an ASN, which is translated by means of a two-level table lookup.

Bits 16-25 of general register R_1 are a 10-bit AFX which is used to select an entry from the ASN first table. Bits 26-31 are a six-bit ASX which is used to select an entry from the ASN second table. The ASN table-lookup process is described in the section "ASN Translation" in Chapter 3, "Storage." The exceptions associated with ASN translation are collectively called "ASN-translation exceptions." These exceptions and their priority are described in Chapter 6, "Interruptions."

The authority-table origin from the ASN-second-table entry is used as the base for a third table lookup. The current authorization index, bits 0-15 of control register 4, is used, after it has been checked against the authority-table length, as the index to locate the entry in the authority table. The authority-table lookup is described in the section "ASN Authorization" in Chapter 3, "Storage."

The PT-ss operation is completed by placing bits 64-95 of the ASN-second-table entry in both the PSTD and SSTD, bit positions 0-31 of control registers 1 and 7, respectively. The contents of bit positions 32-47 of the ASN-second-table entry are placed in the authorization index, bit positions 0-15 of control register 4. The contents of bit positions 96-127 of the ASN-second-table entry are placed in the LTD, bit positions 0-31 of control register 5. The ASN, bits 16-31 of general register R_1, is placed in the SASN and PASN, bit positions 16-31 of control registers 3 and 4.

For both the PT-cp and PT-ss operations, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Special Conditions

The instruction can be executed only when the CPU is in primary-space mode and the subsystem-linkage control, bit 0 of control register 5, is one. If the CPU is in real mode or secondary-space mode, or if the subsystem-linkage control is zero, a special-operation exception is recognized.

Bit 31 of general register R_2 is placed in the problem-state bit position, PSW bit position 15, unless the operation would cause PSW bit 15 to change from one to zero (problem state to supervisor state). If such a change would occur, a privileged-operation exception is recognized.

The instruction is completed only if bits 0-7 of general register R_2 are all zeros; if not, a specification exception is recognized.

In addition to the above requirements, when a PT-ss instruction is specified, the ASN-translation control, bit 12 of control register 14, must be one; otherwise, a special-operation exception is recognized.

When, for PT-ss, the space-switch-event-control bit, bit 31 of control register 1, is one either before or after the execution of the instruction, a space-switch-event program interruption occurs after the operation is completed. A space-switch-event program interruption also occurs after the completion of a PT-ss operation if a PER event is reported.

The operation is suppressed on all addressing exceptions.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: PROGRAM TRANSFER."

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (authority-table entry, PT-ss only)
- ASN translation (PT-ss only)
- Operation (if the dual-address-space facility is not installed)
- Primary authority (PT-ss only)
- Privileged operation (attempt to set the supervisor state when in the problem state)
- Space-switch event (PT-ss only)
- Special operation
- Specification
- Trace

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B.1 Operation exception if the dual-address-space facility is not installed.

7.B.2 Special-operation exception due to DAT being off, the CPU being in secondary-space mode, or the subsystem-linkage-control bit in control register 5 being zero.

8.A Trace exceptions.

8.B.1 Privileged-operation exception due to attempt to set the supervisor state when in the problem state.

8.B.2 Specification exception due to nonzero value in bits 0-7 of general register R_2.

8.B.3 Special-operation exception due to the ASN-translation control, bit 12 of control register 14, being zero (PT-ss only).

8.B.4 ASN-translation exceptions (PT-ss only).

8.B.5 Primary-authority exception due to authority-table entry being outside table (PT-ss only).

8.B.6 Addressing exception for access to authority-table entry (PT-ss only).

8.B.7 Primary-authority exception due to P bit in authority-table entry being zero (PT-ss only).

9. Space-switch event (PT-ss only).

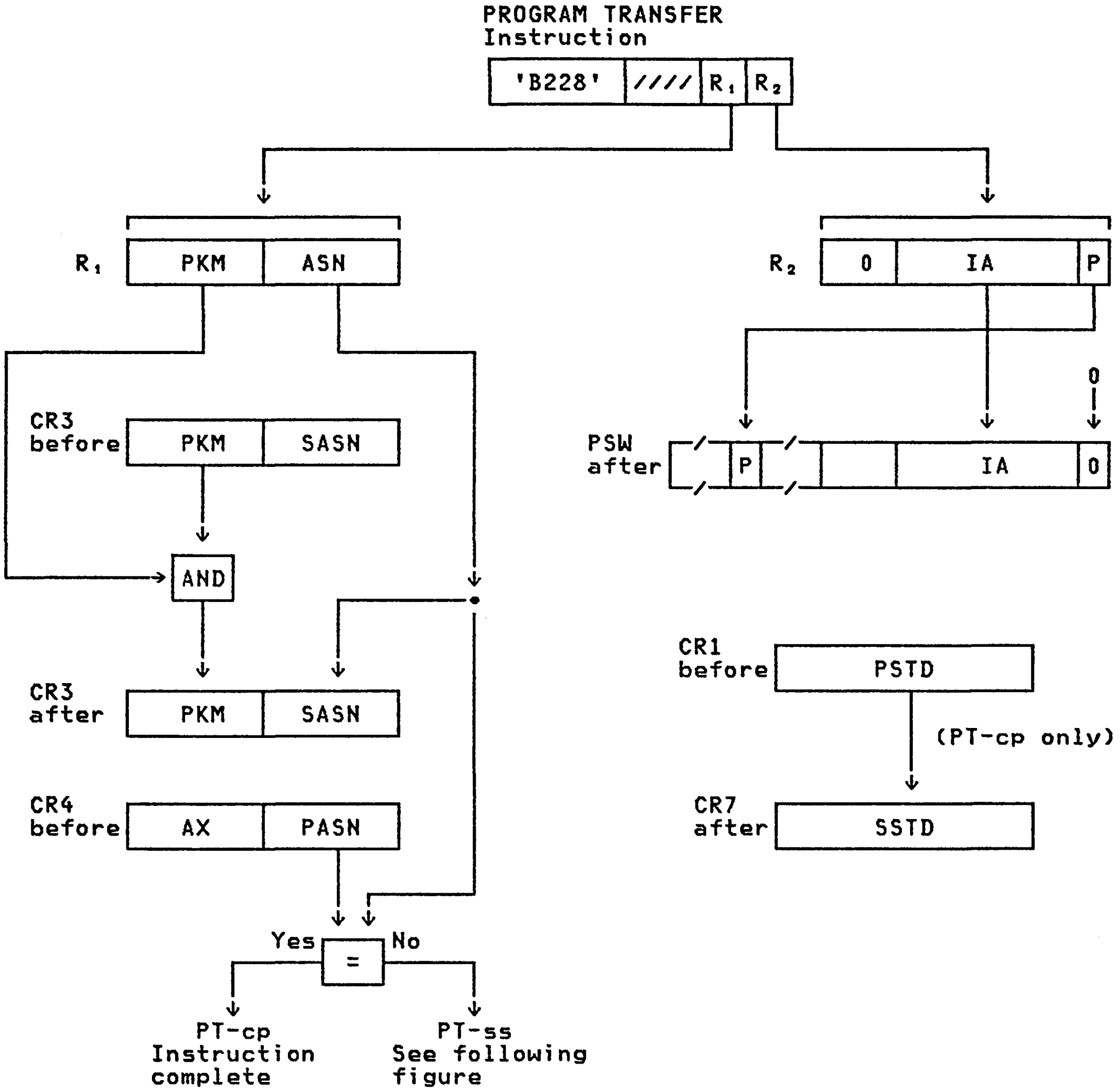
Priority of Execution: PROGRAM TRANSFER

Programming Notes

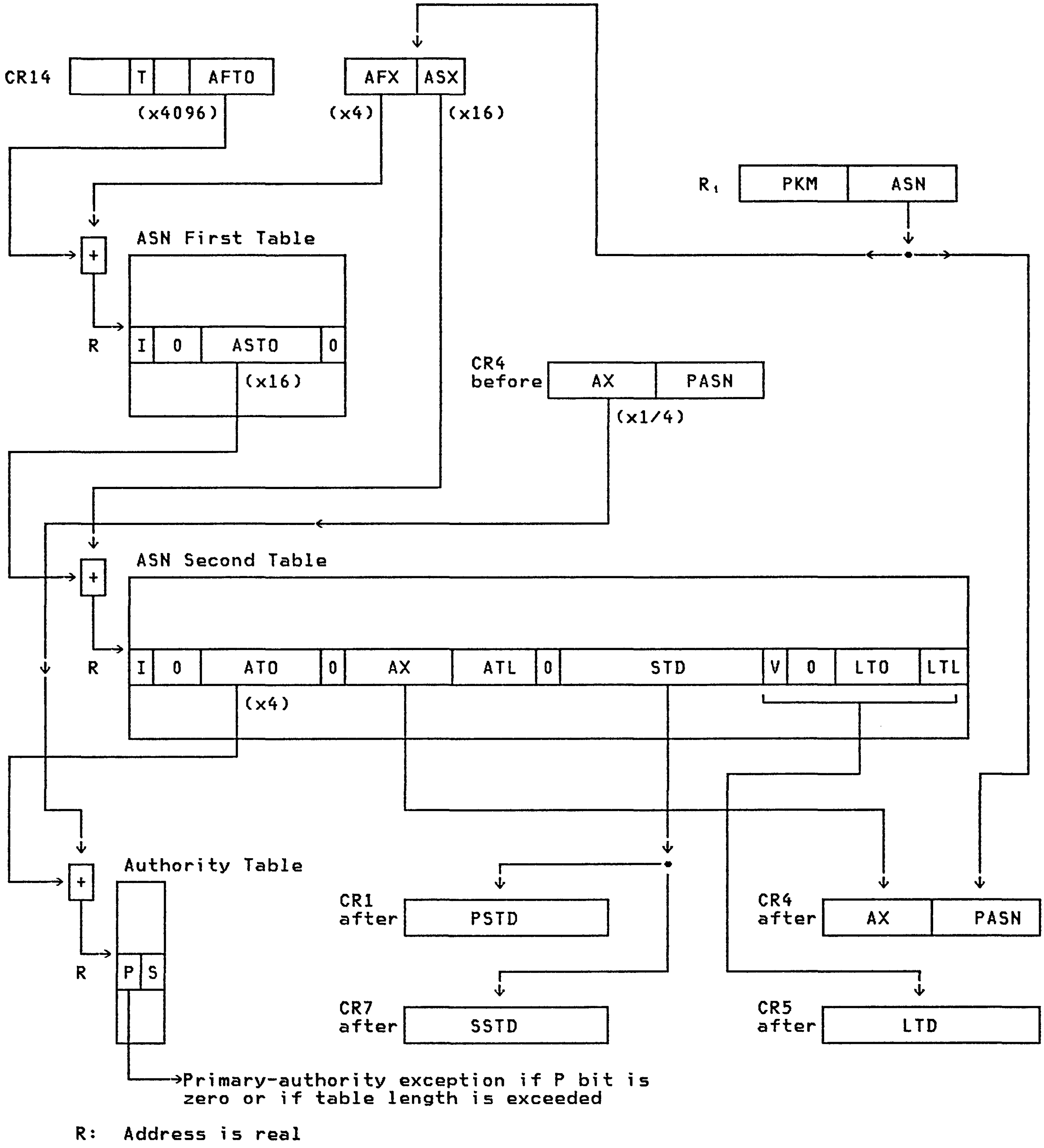
1. The operation of PROGRAM TRANSFER (PT) is such that it may be used to restore the CPU to the state saved by a previous PROGRAM CALL. This restoration is accomplished by issuing PT 3,14. Though general registers 3 and 14 are not restored to their original values, the PASN, PSW-key mask, problem-state bit, and instruction address are restored, and the authorization index, PSTD, and LTD are made consistent with the restored PASN.

2. With proper authority, and while executing in a common area, PROGRAM TRANSFER may be used to change the primary address space to any desired space. The secondary address space is also changed to be the same as the new primary address space.

3. Unlike the RR-format branch instructions, a value of zero in the R_2 field for PROGRAM TRANSFER designates general register 0, and branching occurs.



Execution of PROGRAM TRANSFER (Part 1 of 2): PT-cp and PT-ss



Execution of PROGRAM TRANSFER (Part 2 of 2): PT-ss

PURGE TLB

PTLB [S]

'B20D'	/////////////////
0	16 31

The translation-lookaside buffer (TLB) of this CPU is cleared of entries. No change is made to the contents of addressable storage or registers.

Bits 16-31 of the instruction are ignored.

The TLB appears cleared of its original contents beginning with the fetching of the next sequential instruction. The operation is not signaled to any other CPU.

A serialization function is performed.

<u>Condition Code</u>: The code remains unchanged.

<u>Program Exceptions</u>:

Operation (if the translation facility 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 at the location designated by the first-operand address.

When the INVALIDATE PAGE TABLE ENTRY instruction is not installed, the first-operand address is a logical address, and is subject to the normal access exceptions and to the PER storage-alteration event.

When the INVALIDATE PAGE TABLE ENTRY instruction is installed, the first-operand address is a real address and is not subject to dynamic address translation. Addressing, key-controlled-protection, and low-address-protection exceptions apply. The PER storage-alteration event does not apply.

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 the WRITE DIRECT instruction. 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 the WRITE DIRECT instruction. 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.

An excessively long instruction execution may result in omission of updating of the interval timer.

<u>Condition Code</u>: The code remains unchanged.

<u>Program Exceptions</u>:

Access (store, operand 1; access applies only if the INVALIDATE PAGE TABLE ENTRY instruction is not installed)
Addressing (operand 1)
Operation (if the direct-control facility is not installed)
Privileged operation
Protection (store, operand 1; key-controlled protection and low-address protection)

RESET REFERENCE BIT

RRB $D_2(B_2)$ [S]

'B213'	B_2	D_2
0	16	20 31

The reference bit in the storage key for the 2K-byte block that is designated by the second-operand address is set to zero.

Bits 8-20 of the second-operand address designate a 2K-byte block in real storage. Bits 0-7 and 21-31 of the address are ignored.

When the storage-key 4K-byte-block facility is not installed, all blocks are double-key 4K-byte blocks, and the operation proceeds normally.

When the storage-key 4K-byte-block facility is installed, all blocks are single-key 4K-byte blocks, and the action depends on the setting of the storage-key-exception-control bit, bit 7 of control register 0. If the bit is zero, a special-operation exception is recognized. If the bit is one, the operation is performed on the single key for the 4K-byte block.

Because it is a real address, the address designating the storage block is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The values of the remaining bits of the storage key, including the change bit, are not affected.

The condition code is set to reflect the state of the reference and change bits before the reference bit is set to zero.

<u>Special Conditions</u>

When the storage-key 4K-byte-block facility is installed and the storage-key exception-control bit (bit 7 of control register 0) is zero, a special-operation exception is recognized.

<u>Resulting Condition Code</u>:

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

<u>Program Exceptions</u>:

Addressing (operand 2)
Operation (if the translation facility is not installed)
Privileged operation
Special operation

RESET REFERENCE BIT EXTENDED

RRBE R_1,R_2 [RRE]

```
┌─────────────────┬────────┬────┬────┐
│     'B22A'      │////////│ R1 │ R2 │
└─────────────────┴────────┴────┴────┘
0                 16       24   28   31
```

The reference bits in the storage keys for the 4K-byte block that is addressed by the contents of general register R_2 are set to zeros. The contents of general register R_1 are ignored.

Bits 16-23 of the instruction are ignored.

The contents of general register R_2 are treated as a 31-bit real address of a 4K-byte block in storage. Bits 1-19 of the register designate the 4K-byte block, and bits 0 and 20-31 of the register are ignored.

When the storage-key 4K-byte-block facility is not installed, all blocks are double-key 4K-byte blocks. The key for the first 2K-byte block within the 4K-byte block designated by the instruction is called the low-order key. The key for the second 2K-byte block is called the high-order key. The reference bits of both the low-order and high-order keys are set to zeros.

When the storage-key 4K-byte-block facility is installed, all blocks are single-key 4K-byte blocks. The reference bit in the single key is set to zero.

Because it is a real address, the address designating the storage block is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The remaining bits of the storage key, including the change bit, are not affected.

The condition code is set to reflect the state of the reference and change bits before the reference bit is set to zero. If the addressed block is a single-key 4K-byte block, the reference and change bits in the single key are used. If the block is a double-key 4K-byte block, the condition code is set as a function of the OR of the change bits from the low-order and high-order keys and as a function of the OR of the reference bits from the low-order and high-order keys.

<u>Resulting Condition Code</u>:

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

<u>Program Exceptions</u>:

Addressing (address specified by general register R_2)

Operation (if the storage-key-instruction-extension facility is not installed)
Privileged operation

SET ADDRESS SPACE CONTROL

SAC $D_2(B_2)$ [S]

'B219'	B_2	D_2
0	16	20 31

Bits 20-23 of the second-operand address are used as a code to set the address-space-control bit in the PSW. The second-operand address is not used to address data; instead, bits 20-23 form the code. Bits 8-19 and 24-31 of the second-operand address are ignored. Bits 20-22 of the second-operand address must be zeros; otherwise, a specification exception is recognized.

The following figure summarizes the operation of SET ADDRESS SPACE CONTROL:

Second-Operand Address

/////////////////////	Code	////////
0	20	24 31

Code	Name of Mode	Result in PSW Bit 16
0000	Primary space	0
0001	Secondary space	1
All others	Invalid	Unchanged

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Special Conditions

The operation is performed only when the secondary-space control, bit 5 of control register 0, is one and DAT is on. When either the secondary-space control is zero or DAT is off, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: SET ADDRESS SPACE CONTROL."

Condition Code: The code remains unchanged.

Program Exceptions:

Operation (if the dual-address-space facility is not installed)
Special operation
Specification

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B.1	Operation exception if the dual-address-space facility is not installed.
7.B.2	Special-operation exception due to DAT being off or the secondary-space control, bit 5 of control register 0, being zero.
8.	Specification exception due to nonzero value in bits 20-22 of the second-operand address.

Priority of Execution: SET ADDRESS SPACE CONTROL

Programming Notes

1. SET ADDRESS SPACE CONTROL is defined in such a way that the mode to be set can be placed directly in the displacement field of the instruction or can be specified from the same bit positions of a general register as saved by INSERT ADDRESS SPACE CONTROL.

2. Predictable program operation is ensured in secondary mode only when the instructions are fetched from virtual-address locations which translate to the same real address by means of both the primary and secondary segment tables. Thus, a program should not enter secondary-space mode if it is not aware of the virtual-to-real mapping in both the primary and secondary spaces.

SET CLOCK

SCK $D_2(B_2)$ [S]

'B204'	B_2	D_2
0	16	20 31

The current value of the TOD clock is replaced by the contents of the doubleword designated by the second-operand address, and the clock enters the stopped state.

The doubleword operand replaces the contents of the clock, as determined by the resolution of the clock. Only those bits of the operand are set in the clock that correspond to the bit positions which are updated by the clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock. In some models, starting at or to the right of bit position 52, the rightmost bits of the second operand are ignored, and the corresponding positions of the clock which are implemented are set to zeros.

After the clock value is set, the clock enters the stopped state. The clock leaves the stopped state to enter the set state and resume incrementing under control of the TOD-clock-sync control (bit 2 of control register 0). When the bit is zero or the TOD-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 TOD clock in the configuration is incremented to a value of all zeros in bit positions 32-63.

When the TOD clock is shared by another CPU, the clock remains in the stopped state under control of the TOD-clock-sync control bit of the CPU which set the clock. If, while the clock is stopped, it is set by another CPU, then the clock comes under control of the TOD-clock-sync control bit of the CPU which last set the clock.

The value of the clock is changed and the clock is placed in the stopped state only if the manual TOD-clock control of any CPU in the configuration is set to the enable-set position. If the TOD-clock control is set to 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, the value and state of the clock are not changed, regardless of the setting of the TOD-clock control, and condition code 3 is set.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Resulting Condition Code:

0	Clock value set
1	Clock value secure
2	--
3	Clock in not-operational state

Program Exceptions:

Access (fetch, operand 2)
Privileged operation
Specification

Programming Note

In an installation with more than one CPU, each CPU may have a separate TOD clock, or more than one CPU may share a TOD clock, depending on the model. When multiple TOD clocks exist, special procedures are required to synchronize the clocks. See the section "TOD-Clock Synchronization" in Chapter 4, "Control."

SET CLOCK COMPARATOR

SCKC $D_2(B_2)$ [S]

'B206'	B_2	D_2
0	16	20 31

The current value of the clock comparator is replaced by the contents of the doubleword designated by the second-operand address.

Only those bits of the operand are set in the clock comparator that correspond to the bit positions to be compared with the TOD clock; the contents of the remaining rightmost bit positions of the operand are ignored and are not preserved in the clock comparator.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Operation (if the CPU-timer and clock-comparator facility is not installed)
- Privileged operation
- 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 of the operand are ignored and are not preserved in the CPU timer.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (fetch, operand 2)
- Operation (if the CPU-timer and clock-comparator facility is not installed)
- Privileged operation
- 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. The translation-lookaside buffer (TLB) of this CPU is cleared of entries.

After the second operand is fetched, depending on the model, the prefix value may or may not be tested to determine whether the corresponding 4K-byte block in absolute storage is available before the value is used to replace the contents of the prefix register.

On models which do not test the value, the instruction is completed after setting the prefix register. If the address loaded designates a location which is not available in the configuration, then, when an instruction or interruption procedure is attempted that requires prefixing to be applied to the storage address, the CPU suspends operation. Correction of this condition and allowing processing to be reinitiated requires that a reset be performed, either by means of manual intervention or by receipt of a SIGNAL PROCESSOR reset order.

On models which do test the value, some or all of the necessary checks are performed to ensure that the entire 4K-byte block designated by the prefix address is available. If the storage area is not available, an addressing exception is recognized, and the operation is suppressed. The check to determine that the 4K-byte block is available may involve accessing the location. This access is not subject to protection; however, the access may cause the reference bits to be set to ones.

If the operation is completed, the new prefix is used for any interruptions following the execution of the instruction and for the execution of subsequent instructions. The contents of bit positions 0-7 and 20-31 of the operand are ignored.

The translation-lookaside buffer (TLB) is cleared of entries. The TLB appears cleared of its original contents, beginning with the fetching of the next sequential instruction.

A serialization function is performed before or after the operand is fetched and again after the operation is completed.

Special Conditions

The operand must be designated on a word boundary; otherwise, a specification exception is recognized.

The operation is suppressed on all addressing and protection exceptions.

<u>Condition Code</u>: The code remains unchanged.

<u>Program Exceptions</u>:

Access (fetch, operand 2)
Addressing (new prefix area)
Operation (if the multiprocessing facility is not installed)
Privileged operation
Specification

SET PSW KEY FROM ADDRESS

SPKA $D_2(B_2)$ [S]

'B20A'	B_2	D_2

0 16 20 31

The four-bit PSW key, bits 8-11 of the current PSW, is replaced by bits 24-27 of the second-operand address.

The second-operand address is not used to address data; instead, bits 24-27 of the address form the new PSW key. Bits 8-23 and 28-31 of the second-operand address are ignored.

<u>Special Conditions</u>

In the problem state, when DAS is installed, the execution of the instruction is subject to control by the PSW-key mask in control register 3. When the bit in the PSW-key mask corresponding to the PSW-key value to be set is one, the instruction is executed successfully. When the selected bit in the PSW-key mask is zero, a privileged-operation exception is recognized. When DAS is not installed, execution of the instruction in the problem state results in a privileged-operation exception regardless of the contents of control register 3. In the supervisor state, any value for the PSW key is valid.

<u>Condition Code</u>: The code remains unchanged.

<u>Program Exceptions</u>:

Operation (if the PSW-key-handling facility is not installed)
Privileged operation (executed in the problem state, and either DAS is not installed or selected PSW-key-mask bit is zero)

<u>Programming Notes</u>

1. The format of SET PSW KEY FROM ADDRESS permits the program to set the PSW key either from the general register designated by the B_2 field or from the D_2 field in the instruction itself.

2. When one program requests another program to access a location designated by the requesting program, SET PSW KEY FROM ADDRESS can be used by the called program to verify that the requesting program is authorized to make this access, provided the storage location of the called program is not protected against fetching. The called program can perform the verification by replacing the PSW key with the requesting-program PSW key before making the access and subsequently restoring the called-program PSW key to its original value. Caution must be exercised, however, in handling any resulting protection exceptions since such exceptions may cause the operation to be terminated. See TEST PROTECTION and the associated programming notes for an alternative approach to the testing of addresses passed by a calling program.

SET SECONDARY ASN

SSAR R_1 [RRE]

'B225'	////////	R_1	////

0 16 24 28 31

The ASN specified in bit positions 16-31 of general register R_1 replaces the secondary ASN in control register 3, and the segment-table designation corresponding to that ASN replaces the SSTD in control register 7.

Bits 16-23 and 28-31 of the instruction are ignored.

The contents of bit positions 16-31 of general register R_1 are called the new ASN. The contents of bit positions 0-15 of the register are ignored.

First the new ASN is compared with the current PASN. If the new ASN is equal to the PASN, the operation is called SET SECONDARY ASN to current primary (SSAR-cp). If the new ASN is not equal to the current PASN, the operation is called SET SECONDARY ASN with space switching (SSAR-ss). The SSAR-cp and SSAR-ss operations are depicted in the figure "Execution of SET SECONDARY ASN."

SET SECONDARY ASN to Current Primary (SSAR-cp)

The new ASN replaces the SASN, bits 16-31 of control register 3; the PSTD, bits 0-31 of control register 1, replaces the SSTD, bits 0-31 of control register 7; and the operation is completed.

SET SECONDARY ASN with Space Switching (SSAR-ss)

The new ASN is translated by means of the ASN translation tables, and then the current AX, bits 0-15 of control register 4, is used to test whether the program is authorized to access the specified ASN.

The new ASN is translated by means of a two-level table lookup. Bits 0-9 of the new ASN (bits 16-25 of the register) are a 10-bit AFX which is used to select an entry from the ASN first table. Bits 10-15 of the new ASN (bits 26-31 of the register) are a six-bit ASX which is used to select an entry from the ASN second table. The two-level lookup is described in the section "ASN Translation" in Chapter 3, "Storage." The exceptions associated with ASN translation are collectively called "ASN-translation exceptions." These exceptions and their priority are described in Chapter 6, "Interruptions."

The AST entry obtained as a result of the second lookup contains the segment-table designation and the authority-table origin and length associated with the ASN. All bit positions in the AST entry requiring zeros are inspected for zeros. This includes bits 97-103, even though the linkage-table-designation portion of the entry is not used.

The authority-table origin from the ASN second-table entry is used as a base for a third table lookup. The current authorization index, bits 0-15 of control register 4, is used, after it has been checked against the authority-table length, as the index to locate the entry in the authority table. The authority-table lookup is described in the section "ASN Authorization" in Chapter 3, "Storage."

The new ASN, bits 16-31 of general register R_1, is placed in the SASN, bit positions 16-31 of control register 3. The segment-table designation, bits 64-95 of the AST entry, is placed in the SSTD, bits 0-31 of control register 7.

For both the SSAR-cp and SSAR-ss operations, a serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Special Conditions

The operation is performed only when the ASN-translation control, bit 12 of control register 14, is one and DAT is on. When either the ASN-translation-control bit is zero or DAT is off, a special-operation exception is recognized. The special-operation exception is recognized in both the problem and supervisor states.

The priority of recognition of program exceptions for the instruction is shown in the figure "Priority of Execution: SET SECONDARY ASN."

Condition Code: The code remains unchanged.

Program Exceptions:

Addressing (authority-table entry, SSAR-ss only)
ASN translation (SSAR-ss only)
Operation (if the dual-address-space facility is not installed)
Secondary authority (SSAR-ss only)
Special operation
Trace

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B.1 Operation exception if the dual-address-space facility is not installed.

7.B.2 Special-operation exception due to DAT being off, or the ASN-translation control, bit 12 of control register 14, being zero.

8.A Trace exceptions.

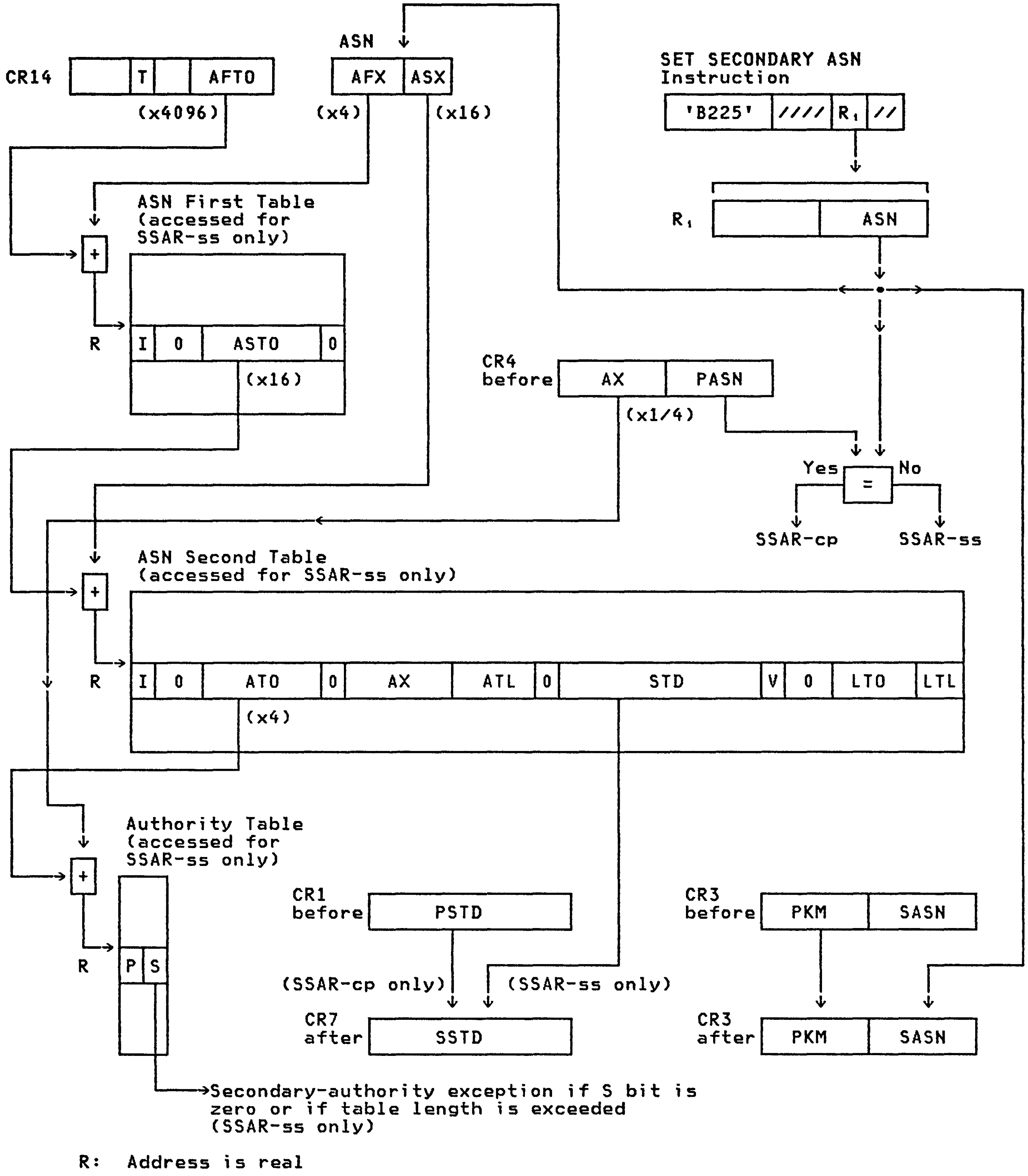
8.B.1 ASN-translation exceptions (SSAR-ss only).

8.B.2 Secondary-authority exception due to authority-table entry being outside table (SSAR-ss only).

8.B.3 Addressing exception for access to authority-table entry (SSAR-ss only).

8.B.4 Secondary-authority exception due to S bit in authority-table entry being zero (SSAR-ss only).

Priority of Execution: SET SECONDARY ASN


ASN
SET SECONDARY ASN Instruction
CR14
T
AFTO
AFX
ASX
'B225'
////
R1
//
(x4096)
(x4)
(x16)
R1
ASN
ASN First Table (accessed for SSAR-ss only)
+
R
I
0
ASTO
0
(x16)
CR4 before
AX
PASN
(x1/4)
Yes
=
No
SSAR-cp
SSAR-ss
ASN Second Table (accessed for SSAR-ss only)
+
R
I
0
ATO
0
AX
ATL
0
STD
V
0
LTO
LTL
(x4)
Authority Table (accessed for SSAR-ss only)
+
R
P
S
CR1 before
PSTD
(SSAR-cp only)
(SSAR-ss only)
CR7 after
SSTD
CR3 before
PKM
SASN
CR3 after
PKM
SASN
Secondary-authority exception if S bit is zero or if table length is exceeded (SSAR-ss only)
R: Address is real

Execution of SET SECONDARY ASN

SET STORAGE KEY

SSK R_1,R_2 [RR]

'08'	R_1	R_2
0	8	12 15

The storage key for the 2K-byte block that is addressed by the contents of general register R_2 is replaced by bits from general register R_1.

Bits 8-20 of general register R_2 designate a 2K-byte block in real 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.

When the storage-key 4K-byte-block facility is not installed, all blocks are double-key 4K-byte blocks, and the operation proceeds normally.

When the storage-key 4K-byte-block facility is installed, all blocks are single-key 4K-byte blocks, and the operation depends on the setting of the storage-key-exception-control bit, bit 7 of control register 0. If the bit is zero, a special-operation exception is recognized. If the bit is one, the operation is performed on the single key for the 4K-byte block.

Because it is a real address, the address designating the storage block is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The new seven-bit storage-key value is obtained from bit positions 24-30 of general register R_1. The contents of bit positions 0-23 and 31 of the register are ignored. When the translation facility is not installed, bits 29 and 30 are ignored.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Special Conditions

Bits 28-31 of general register R_2 must be zeros; otherwise, a specification exception is recognized.

When the storage-key 4K-byte-block facility is installed and the storage-key-exception-control bit (bit 7 of control register 0) is zero, a special-operation exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Addressing (address specified by general register R_2)
- Privileged operation
- Special operation
- Specification

SET STORAGE KEY EXTENDED

SSKE R_1,R_2 [RRE]

'B22B'	/////////	R_1	R_2
0	16	24	28 31

The storage keys for the 4K-byte block that is addressed by the contents of general register R_2 are replaced by bits from general register R_1.

Bits 16-23 of the instruction are ignored.

The contents of general register R_2 are treated as a 31-bit real address of a 4K-byte block in storage. Bits 1-19 of the register designate the 4K-byte block, and bits 0 and 20-31 of the register are ignored.

When the storage-key 4K-byte-block facility is not installed, all blocks are double-key 4K-byte blocks. The key for the first 2K-byte block within the 4K-byte block designated by the instruction is called the low-order key. The key for the second 2K-byte block is called the high-order key. Both the low-order key and the high-order key are replaced. The two keys are not necessarily updated concurrently.

When the storage-key 4K-byte-block facility is installed, all blocks are single-key 4K-byte blocks, and the single key is replaced.

Because it is a real address, the address designating the storage block is not subject to dynamic address translation. The reference to the storage key is not subject to a protection exception.

The new seven-bit storage-key value is obtained from bit positions 24-30 of general register R_1. The contents of bit positions 0-23 and 31 of the register are ignored.

A serialization and checkpoint-synchronization function is performed before the operation begins and again after the operation is completed.

Condition Code: The code remains unchanged.

Program Exceptions:

Addressing (address specified by general register R_2)
Operation (if the storage-key-instruction-extension facility is not installed)
Privileged operation

SET SYSTEM MASK

SSM $D_2(B_2)$ [S]

'80'	////////	B_2	D_2	
0	8	16	20	31

Bits 0-7 of the current PSW are replaced by the byte at the location designated by the second-operand address.

Bits 8-15 of the instruction are ignored.

Special Conditions

When the translation facility is installed, the execution of the instruction is subject to the SSM-suppression-control 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.

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 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 specification exception, which is listed as a program exception for this instruction, is described in the section "Early Exception Recognition" in Chapter 6, Interruptions." This exception may be considered as caused by execution of this instruction or as occurring early in the process of preparing to execute the subsequent instruction.

The operation is suppressed on all addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 2)
Privileged operation
Special operation
Specification

Programming Note

SET SYSTEM MASK is frequently used in the BC mode to disable or enable the CPU for I/O or external interruptions. Hence, suppressing the execution of SET SYSTEM MASK by means of the SSM-suppression-control bit, bit 1 of control register 0, may be useful when converting a program written for a BC-mode PSW to operate with an EC-mode PSW.

SIGNAL PROCESSOR

SIGP $R_1,R_3,D_2(B_2)$ [RS]

'AE'	R_1	R_3	B_2	D_2	
0	8	12	16	20	31

An eight-bit order code is transmitted to the CPU designated by the CPU address contained in the third operand. The result is indicated by the condition code and may be detailed by status assembled in 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 appear in the section "CPU Signaling and Response" in Chapter 4, "Control."

The 16-bit binary number contained in bit positions 16-31 of general register R_3 forms the CPU address. Bits 0-15 of the register are ignored.

The operands just described have the following formats:

General register designated by R_1:

Status	
0	31

General register designated by R_3:

/////////////////	CPU Address	
0	16	31

Second-operand address:

//////////////////////////	Order Code	
0	24	31

A serialization function is performed before the operation begins and again after the operation is completed.

When the order code is accepted and no nonzero status is returned, condition code 0 is set. When status information is generated by this CPU or returned by the addressed CPU, the status is placed in general register R_1, and condition code 1 is set.

When the access path to the addressed CPU is busy, or the addressed CPU is operational but in a state where it cannot respond to the order code, condition code 2 is set.

When the addressed CPU is not operational (that is, it is not provided in the installation, it is not in the configuration, it is in any of certain customer-engineer test modes, or its power is off), condition code 3 is set.

Resulting Condition Code:

0 Order code accepted
1 Status stored
2 Busy
3 Not operational

Program Exceptions:

Operation (if the multiprocessing facility is not installed)
Privileged operation

Programming Notes

1. A more detailed discussion of the condition-code settings for SIGNAL PROCESSOR is contained in the section "CPU Signaling and Response" in Chapter 4, "Control."

2. To ensure that presently written programs will be executed properly when new facilities using additional bits are installed, only zeros should appear in the unused bit positions of the second-operand address and in bit positions 0-15 of general register R_3.

3. Certain SIGNAL PROCESSOR orders are provided with the expectation that they will be used primarily in special circumstances. Such orders may be implemented with the aid of an auxiliary maintenance or service processor, and, thus, the execution time may take several seconds. Unless all of the functions provided by the order are required, combinations of other orders, in conjunction with appropriate programming support, can be expected to provide a specific function more rapidly. The emergency-signal, external-call, and sense orders are the only orders which are intended for frequent use. The following orders are intended for infrequent use, and performance therefore may be much slower than for frequently used orders: IML, restart, start, stop, stop and store status, and all the reset orders.

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 location designated by the second-operand address.

Zeros are provided for the rightmost bit positions of the clock comparator that are not compared with the TOD clock.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)
Operation (if the CPU-timer and clock-comparator facility is not installed)
Privileged operation
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 control register R_1 and ending with control register R_3 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 register numbers, starting with control register R_1 and continuing up to and including control register R_3, with control register 0 following control register 15. The contents of the control registers remain unchanged.

The information stored for unassigned control-register positions, or positions associated with a facility which is not installed, is unpredictable.

Special Conditions

The second operand must be designated on a word boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)
Privileged operation
Specification

Programming Note

Although 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 CPU address by which this CPU is identified in a multiprocessing configuration is stored at the halfword location designated by the second-operand address.

Special Conditions

The operand must be designated on a halfword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (store, operand 2)
Operation (if the multiprocessing facility is not installed)
Privileged operation
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 information stored has the following format:

Version Code	CPU Identification Number

0 8 31

Model Number	Maximum MCEL Length

32 48 63

Bit positions 0-7 contain the version code. The format and significance of the version code depend on the model.

Bit positions 8-31 contain the CPU identification number, consisting of six four-bit digits. Some or all of these digits are selected from the physical serial number stamped on the CPU. The 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: leftmost zero digits, if necessary, followed by the digits of the System/370 model number. For example, a Model 145 or 3033 system would store 0145 hex or 3033 hex, respectively.

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.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Privileged operation
- Specification

Programming Notes

1. The program should allow for the possibility that the CPU identification number may contain the digits A-F as well as the digits 0-9.

2. The principal uses of the information stored by STORE CPU ID are the following:

   a. The CPU identification number, in conjunction 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.

   b. 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.

   c. 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 location designated by the second-operand address.

Zeros are provided for the rightmost bit positions that are not updated by the CPU timer.

Special Conditions

The operand must be designated on a doubleword boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (if the CPU-timer and clock-comparator facility is not installed)
- Privileged operation
- 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.

Special Conditions

The operand must be designated on a word boundary; otherwise, a specification exception is recognized.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 2)
- Operation (if the multiprocessing facility is not installed)
- Privileged operation
- 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 AND of their original contents and the second operand.

Special Conditions

The operation is suppressed on addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 1)
- Operation (if the translation facility is not installed)
- Privileged operation

Programming Note

STORE THEN AND SYSTEM MASK permits the program to set selected bits in the system mask to zeros while retaining the original contents for later restoration. For example, it may be necessary that a program, which has no record of the present status, disable program-event recording for a few instructions.

STORE THEN OR SYSTEM MASK

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 OR of their original contents and the second operand.

Special Conditions

The value to be loaded into the PSW is not checked for validity before loading. However, immediately after loading, a specification exception is recognized, and a program interruption occurs, if the 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 specification exception, which is listed as a program exception for this instruction, is described in the section "Early Exception Recognition" in Chapter 6, "Interruptions." This exception may be considered as caused by execution of this instruction or as occurring early in the process of preparing to execute the subsequent instruction.

The operation is suppressed on addressing and protection exceptions.

Condition Code: The code remains unchanged.

Program Exceptions:

- Access (store, operand 1)
- Operation (if the translation facility is not installed)
- Privileged operation
- Specification

Programming Note

STORE THEN OR SYSTEM MASK permits the program to set selected bits in the system mask to ones while retaining the original contents for later restoration. For example, the program may enable the CPU for I/O interruptions without having available the current status of the external-mask bit.

TEST BLOCK

TB R_1,R_2 [RRE]

'B22C'	////////	R_1	R_2

0 16 24 28 31

The storage locations and storage keys of a 4K-byte block are tested for usability, and the result of the test is indicated in the condition code. The test for usability is based on the susceptibility of the block to the occurrence of invalid checking-block code.

Bits 16-23 of the instruction are ignored.

The block tested is addressed by the contents of general register R_2. The contents of general register R_1 are ignored.

When the storage-key 4K-byte-block facility is not installed, all blocks are double-key 4K-byte blocks, and two keys are tested.

When the storage-key 4K-byte-block facility is installed, all blocks are single-key 4K-byte blocks, and only one key is tested. In this instruction definition, the term "storage keys" is used whether one or two storage keys are affected.

A complete testing operation is necessarily performed only when the initial contents of general register 0 are zero. The contents of general register 0 are set to zero at the completion of the operation.

If the block is found to be usable, the 4K bytes of the block are cleared to zeros, the contents of the storage keys are unpredictable, and condition code 0 is set. If the block is found to be unusable, the data and the storage keys are set, as far as is possible by the model, to a value such that subsequent fetches to the area do not cause a machine-check condition, and condition code 1 is set.

The contents of general register R_2 are treated as a 31-bit real address of a 4K-byte block in storage. Bits 1-19 of the register designate the 4K-byte block, and bits 0 and 20-31 of the register are ignored.

The address of the block is a real address, and the accesses to the block designated by the second-operand address are not subject to key-controlled and segment protection. Low-address protection does apply. The operation is terminated on addressing and protection exceptions. If termination occurs, the condition code and the contents of general register 0 are unpredictable. The contents of the storage block and its associated storage keys are not changed when these exceptions occur.

Depending on the model, the test for usability may be performed (1) by alternately storing and reading out test patterns to the data and storage keys in the block or (2) by reference to an internal record of the usability of the blocks which are available in the configuration, or (3) by using a combination of both mechanisms.

In models in which an internal record is used, the block is indicated as unusable if a solid failure has been previously detected, or if intermittent failures in the block have exceeded the threshold implemented by the model. In such models, depending on the criteria, attempts to store may or may not occur. Thus, if block 0 is not usable, and no store occurs, low-address protection may or may not be indicated.

In models in which test patterns are used, TEST BLOCK may be interruptible. When an interruption occurs after a unit of operation, other than the last one, the condition code is unpredictable, and the contents of general register 0 may contain a record of the state of intermediate steps. When execution is resumed after an interruption, the condition code is ignored, but the contents of general register 0 may be used to determine the resumption point.

If (1) TEST BLOCK is executed with an initial value other than zero in general register 0, or (2) the interrupted instruction is resumed after an interruption with a value in general register 0 other than the value which was present at the time of the interruption, or (3) the block is accessed by another CPU or by a channel during the execution of the instruction, then the contents of the storage block, its associated storage keys, and general register 0 are unpredictable, along with the resultant condition-code setting.

Invalid checking-block-code errors initially found in the block or encountered during the test do not normally result in machine-check conditions. The test-block function is implemented in such a way that the frequency of machine-check interruptions due to the instruction execution is not significant. However, if, during the execution of TEST BLOCK for an unusable block, that block is accessed by another CPU (or by a channel), error conditions may be reported both to this CPU and to the other CPU (or to the channel).

A serialization function is performed before the block is accessed and again after the operation is completed (or partially completed).

The priority of the recognition of exceptions and condition codes is shown in the figure "Priority of Execution: TEST BLOCK."

<u>Resulting</u> <u>Condition</u> <u>Code</u>:

0	Block usable
1	Block not usable
2	--
3	--

Program Exceptions:

Addressing (fetch and store, operand 2)
Operation (if the test-block facility is not installed)
Privileged operation
Protection (store, operand 2, low-address protection only)

1.-6. Exceptions with the same priority as the priority of program-interruption conditions for the general case.

7.A Access exceptions for second instruction halfword.

7.B Privileged-operation exception.

8. Addressing exception due to block not being available in the configuration.*

9.A Condition code 1, block not usable.

9.B Protection exception due to low-address protection.*

10. Condition code 0, block usable and set to zeros.

Explanation:

* The operation is terminated on addressing and protection exceptions, and the condition code may be unpredictable.

Priority of Execution: TEST BLOCK

Programming Notes

1. The execution of TEST BLOCK on most models is significantly slower than that of the MOVE LONG instruction with padding; therefore, the instruction should not be used for the normal case of clearing storage.

2. The program should use TEST BLOCK at initial program loading and as part of the vary-storage-online procedure to determine if blocks of storage exist which should not be used.

3. The program should use TEST BLOCK when an uncorrected error is reported in either the data or storage keys of a block. This is because in the execution of TEST BLOCK the attempt is made, as far as is possible on the model, to leave the contents of a block in a state such that subsequent prefetches or unintended references to the block do not cause machine-check conditions. The program may use the resulting condition code in this case to determine if the block can be reused. (The block could be indicated as usable if, for example, the error were an externally generated error or an indirect storage error.) This procedure should be followed regardless of whether the indirect-storage-error indication is reported.

4. The model may or may not be successful in removing the errors from a block when TEST BLOCK is executed. The program therefore should take every reasonable precaution to avoid referencing an unusable block. For example, the program should not place the page-frame real address of an unusable block in an attached and valid page-table entry.

5. On some models, machine checks may be reported for a block even though the block is not referenced by the program. When a machine check is reported for a storage-key error in a block which has been marked as unusable by the program, it is possible that SET STORAGE KEY or SET STORAGE KEY EXTENDED may be more effective than TEST BLOCK in validating the storage key.

6. The storage-operand references for TEST BLOCK may be multiple-access references. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

TEST PROTECTION

TPROT $D_1(B_1),D_2(B_2)$ [SSE]

'E501'	B_1	D_1	B_2	D_2
0	16	20	32	36 47

The location designated by the first-operand address is tested for protection exceptions using the access key specified in bits 24-27 of the second-operand address.

The second-operand address is not used to address data; instead, bits 24-27 of the address form the access key to be used in testing. Bits 8-23 and 28-31 of the second-operand address are ignored.

The first-operand address is a logical address and thus is subject to translation when DAT is on. When DAT is on and the first-operand address cannot be translated because of a situation that would normally cause a page-translation or segment-translation exception, the instruction is completed by setting condition code 3.

When translation of the first-operand address can be completed, or when DAT is off, the storage key for the block designated by the first-operand address is tested against the access key specified in bits 24-27 of the second-operand address, and the condition code is set to indicate whether store and fetch accesses are permitted, taking into consideration all applicable protection mechanisms. Thus, for example, if the low-address-protection facility is installed and active and if the first-operand effective address is less than 512, then a store access is not permitted. Segment protection, when installed, is also taken into account.

The contents of storage, including the change bit, are not affected. Depending on the model, the reference bit for the first-operand address may be set to one, even for the case in which the location is protected against fetching.

Special Conditions

When DAT is on, an addressing exception is recognized when the address of the segment-table entry, the page-table entry, or the operand real address after translation designates a location which is not available in the configuration. Also, when DAT is on, a translation-specification exception is recognized when the segment-table entry or page-table entry has a format error. When DAT is off, only the addressing exception due to the operand real address applies. For all of these cases, the operation is suppressed.

Resulting Condition Code:

0	Fetching permitted; storing permitted
1	Fetching permitted; storing not permitted
2	Fetching not permitted; storing not permitted
3	Translation not available

Program Exceptions:

Addressing (operand 1)
Operation (if the extended facility is not installed)
Privileged operation
Translation specification

Programming Notes

1. TEST PROTECTION permits a program to check the validity of an address passed from a calling program without incurring program exceptions. The instruction sets a condition code to indicate whether fetching or storing is permitted at the location designated by the first-operand address of the instruction. The instruction takes into consideration all of the protection mechanisms installed in the machine: key-controlled, segment, and low-address protection. Additionally, since segment translation and page translation may be a program substitute for a protection violation, these situations are used to set the condition code rather than cause a program exception.

2. See the programming notes under SET PSW KEY FROM ADDRESS for more details and for an alternative approach to testing validity of addresses passed by a calling program. The approach using TEST PROTECTION has the advantage of a test which does not result in interruptions; however, the test and use are separated in time and may not be accurate if the possibility exists that the storage key of the location in question can change between the time it is tested and the time it is used.

3. In the handling of dynamic address translation, TEST PROTECTION is similar to LOAD REAL ADDRESS in that the instructions do not cause page-translation and segment-translation exceptions. Instead, these situations are indicated by means of a condition-code setting. Situations which result in condition codes 1, 2, and 3 for LOAD REAL ADDRESS result in condition code 3 for TEST PROTECTION. The instructions also differ in several other respects. The first-operand address of TEST PROTECTION is a logical address and thus is not subject to translation when DAT is off. The second-operand address of LOAD REAL ADDRESS is a virtual address which is always translated. TEST PROTECTION may use the TLB for translation of the address, whereas LOAD REAL ADDRESS does not use the TLB. (LOAD REAL ADDRESS is the only instruction which must perform translation without use of the TLB.)

   When DAT is off for LOAD REAL ADDRESS, the translation-specification exception for an invalid value of bits 8-12 of control register 0 occurs after instruction fetching as part of the execution portion of the instruction. This situation cannot occur for TEST PROTECTION since the operand address is a logical address and does not result in examination of control register 0 when DAT is off. When DAT is on, the exception would be recognized during instruction fetching. Since the instruction-fetching portion of an instruction is common for all instructions, descriptions of access exceptions associated with instruction fetching do not appear in the individual instruction definitions.

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 first-operand address is made available as a set of direct-out static signals. Eight instruction bits are made available as signal-out timing signals.

When INVALIDATE PAGE TABLE ENTRY is not installed, the first-operand address is a logical address and subject to normal access exceptions. When INVALIDATE PAGE TABLE ENTRY is installed, the first-operand address is a real address and therefore not subject to translation; only addressing and key-controlled-protection exceptions apply.

The eight data bits of the byte fetched from the real storage location designated by the first-operand address are presented on a set of eight direct-out lines as static signals. These signals remain until WRITE DIRECT is again 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 the READ DIRECT instruction. 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.

Condition Code: The code remains unchanged.

Program Exceptions:

Access (fetch, operand 1; access applies only if the INVALIDATE PAGE TABLE ENTRY instruction is not installed)
Addressing (fetch, operand 1)
Operation (if the direct-control facility is not installed)
Privileged operation
Protection (fetch, operand 1)

CHAPTER 11. MACHINE-CHECK HANDLING

The machine-check-handling mechanism provides extensive equipment-malfunction detection to ensure the integrity of system operation and to permit automatic recovery from some malfunctions. Equipment malfunctions and certain external disturbances are reported by means of a machine-check interruption to assist in program-damage assessment and recovery. The interruption supplies the program with information about the extent of the damage and the location and nature of the cause. Equipment malfunctions, errors, and other situations which can cause machine-check interruptions are referred to as machine checks.

MACHINE-CHECK DETECTION

Machine-check-detection mechanisms may take many forms, especially in control functions for arithmetic and logical processing, addressing, sequencing, and execution. For program-addressable information, detection is normally accomplished by encoding redundancy into the information in such a manner that most failures in the retention or transmission of the information result in an invalid code. The encoding normally takes the form of one or more redundant bits, called check bits, appended to a group of data bits. Such a group of data bits and the associated check bits are called a checking block. The size of the checking block depends on the model.

The inclusion of a single check bit in the checking block allows the detection of any single-bit failure within the checking block. In this arrangement, the check bit is sometimes referred to as a "parity bit." In other arrangements, a group of check bits is included to permit detection of multiple errors, to permit error correction, or both.

For checking purposes, the contents of the entire checking block, including the redundancy, are called the checking-block code (CBC). When a CBC completely meets the checking requirements (that is, no failure is detected), it is said to be valid. When both detection and correction are provided and a CBC is not valid but satisfies the checking requirements for correction (the failure is correctable), it is said to be near-valid. When a CBC does not satisfy the checking requirements (the failure is uncorrectable), it is said to be invalid.

CORRECTION OF MACHINE MALFUNCTIONS

Three mechanisms may be used to provide recovery from machine-detected malfunctions: error checking and correction, CPU retry, and unit deletion.

Machine failures which are corrected successfully may or may not be reported as machine-check interruptions. If reported, they are system-recovery conditions, which permit the program to note the cause of CPU delay and to keep a log of such incidents.

ERROR CHECKING AND CORRECTION

When sufficient redundancy is included in circuitry or in a checking block, failures can be corrected. For example, circuitry can be triplicated, with a voting circuit to determine the correct value by selecting two matching results out of three, thus correcting a single failure. An arrangement for correction of failures of one order and for detection of failures of a higher order is called error checking and correction (ECC). Commonly, ECC allows correction of single-bit failures and detection of double-bit failures.

Depending on the model and the portion of the machine in which ECC is applied, correction may be reported as system recovery, or no report may be given.

Uncorrected errors in storage and in the storage key may be reported, along with a failing-storage address, to indicate where the error occurred. Depending on the situation, these errors may be reported along with system recovery, with external secondary report, or with the damage or backup condition resulting from the error.

CPU RETRY

In some models, information about some portion of the state of the machine is saved periodically. The point in the processing at which this information is saved is called a checkpoint. The information saved is referred to as the checkpoint information. The action of saving the information is referred to as establishing a checkpoint. The action of discarding previously saved information is called invalidation of the checkpoint information. The length of the interval between establishing checkpoints is model-dependent. Checkpoints may be established at the beginning of each instruction or several times within a single instruction, or checkpoints may be established less frequently.

Subsequently, this saved information may be used to restore the machine to the state that existed at the time when the checkpoint was established. After restoring the appropriate portion of the machine state, processing continues from the checkpoint. The process of restoring to a checkpoint and then continuing is called CPU retry.

CPU retry may be used for machine-check recovery, to effect nullification and suppression of instruction execution when certain program interruptions occur, and in other model-dependent situations.

Effects of CPU Retry

CPU retry is, in general, performed so that there is no effect on the program. However, change bits which have been changed from zeros to ones are not necessarily set back to zeros. As a result, change bits may appear to be set to ones for blocks which would have been accessed if restoring to the checkpoint had not occurred. If the path taken by the program is dependent on information that may be changed by another CPU or by a channel or if an interruption occurs, then the final path taken by the program may be different from the earlier path; therefore, change bits may be ones because of stores along a path apparently never taken.

Checkpoint Synchronization

Checkpoint synchronization consists in the following steps.

1. The CPU operation is delayed until all conceptually previous accesses by this CPU to storage have been completed, both for purposes of machine-check detection and as observed by other CPUs and by channels.

2. All previous checkpoints, if any, are canceled.

3. Optionally, a new checkpoint is established. The CPU operation is delayed until all of these actions appear to be completed, as observed by other CPUs and by channels.

Handling of Machine Checks during Checkpoint Synchronization

When, in the process of completing all previous stores as part of the checkpoint-synchronization action, the machine is unable to complete all stores successfully but can successfully restore the machine to a previous checkpoint, processing backup is reported.

When, in the process of completing all stores as part of the checkpoint-synchronization action, the machine is unable to complete all stores successfully and cannot successfully restore the machine to a previous checkpoint, the type of machine-check-interruption condition reported depends on the origin of the store. Failure to successfully complete stores associated with instruction execution may be reported as instruction-processing damage, or some less critical machine-check-interruption condition may be reported with the storage-logical-validity bit set to zero. A failure to successfully complete stores associated with the execution of an interruption, other than program or supervisor call, is reported as system damage.

When the machine check occurs as part of a checkpoint-synchronization action before the execution of an instruction, the execution of the instruction is nullified. When it occurs before the execution of an interruption, the interruption condition, if the interruption is external, I/O, or restart, is held pending. If the checkpoint-synchronization operation was a machine-check interruption, then along with the originating condition, either the storage-logical-validity bit is set to zero or instruction-processing damage is also reported. Program interruptions, if any, are lost.

Checkpoint-Synchronization Operations

All interruptions and the execution of certain instructions cause a checkpoint-synchronization action to be performed. The operations which cause a checkpoint-synchronization action are called checkpoint-synchronization operations and include:

- CPU reset
- All interruptions: external, I/O, machine check, program, restart, and supervisor call
- The BRANCH ON CONDITION (BCR) instruction with the M_1 and R_2 fields containing all ones and all zeros, respectively
- The instructions LOAD PSW, SET STORAGE KEY, SET STORAGE KEY EXTENDED, and SUPERVISOR CALL
- All I/O instructions
- The instructions MOVE TO PRIMARY, MOVE TO SECONDARY, PROGRAM CALL, PROGRAM TRANSFER, SET ADDRESS SPACE CONTROL, and SET SECONDARY ASN
- The DAS-tracing function

Programming Note

The instructions which are defined to cause the checkpoint-synchronization action invalidate checkpoint information but do not necessarily establish a new checkpoint. Additionally, the CPU may establish a checkpoint between any two instructions or units of operation, or within a single unit of operation. Thus, the point of interruption for the machine check is not necessarily at an instruction defined to cause a checkpoint-synchronization action.

Checkpoint-Synchronization Action

For all interruptions except I/O interruptions, a checkpoint-synchronization action is performed at the completion of the interruption. For I/O interruptions, a checkpoint-synchronization action may or may not be performed at the completion of the interruption. For all interruptions except program, supervisor-call, and exigent machine-check interruptions, a checkpoint-synchronization action is also performed before the interruption. The fetch access to the new PSW may be performed either before or after the first checkpoint-synchronization action. The store accesses and the changing of the current PSW associated with the interruption are performed after the first checkpoint-synchronization action and before the second.

For all checkpoint-synchronization instructions except BRANCH ON CONDITION (BCR), I/O instructions, and SUPERVISOR CALL, checkpoint-synchronization actions are performed before and after the execution of the instruction. For BCR, only one checkpoint-synchronization action is necessarily performed, and it may be performed either before or after the instruction address is updated. For SUPERVISOR CALL, a checkpoint-synchronization action is performed before the instruction is executed, including the updating of the instruction address in the PSW. The checkpoint-synchronization action taken after the supervisor-call interruption is considered to be part of the interruption action and not part of the instruction execution. For I/O instructions, a checkpoint-synchronization action is always performed before the instruction is executed and may or may not be performed after the instruction is executed.

The DAS-tracing function causes checkpoint-synchronization actions to be performed before the trace action and after completion of the trace action.

UNIT DELETION

In some models, malfunctions in certain units of the system can be circumvented by discontinuing the use of the unit. Examples of cases where unit deletion may occur include the disabling of all or a portion of a cache or of a translation-lookaside buffer (TLB). Unit deletion may be reported as a degradation machine-check-interruption condition.

HANDLING OF MACHINE CHECKS

A machine check is caused by a machine malfunction and not by data or instructions. This is ensured during the power-on sequence by initializing the machine controls to a valid state and by placing valid CBC in the CPU registers, in the storage keys, and, if it is volatile, also in main storage.

Designation of an unavailable 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 condition-code setting. In particular, an attempt to access a storage location which is not in the configuration, or which has power off at the storage unit, results in an addressing exception when detected by the CPU and does not generate a machine-check condition, even though the storage location or its associated storage key has invalid CBC. Similarly, if the channel attempts to access such a location, an I/O-interruption condition indicating program check is generated rather than a machine-check condition.

A machine check is indicated whenever the result of an operation could be affected by information with invalid CBC, or when any other malfunction makes it impossible to establish reliably that an operation can be, or has been, performed correctly. When information with invalid CBC is fetched but not used, the condition may or may not be indicated, and the invalid CBC is preserved.

When a machine malfunction is detected, the action taken depends on the model, the nature of the malfunction, and the situation in which the malfunction occurs. Malfunctions affecting operator-facility actions may result in machine checks or may be indicated to the operator. Malfunctions affecting certain other operations such as SIGNAL PROCESSOR may be indicated by means of a condition code or may result in a machine-check-interruption condition.

A malfunction detected as part of an I/O operation may cause a machine-check-interruption condition, an I/O-error condition, or both. I/O-error conditions are indicated by an I/O interruption or by the appropriate condition-code setting during the execution of an I/O instruction. When the machine reports a failing-storage location detected during an I/O operation, both I/O-error and machine-check conditions may be indicated. The I/O-error condition is the primary indication to the program. The machine-check condition is a secondary indication, which is presented as system recovery or as an external secondary report, together with a failing-storage address.

VALIDATION

Machine errors can be generally classified as solid or intermittent, according to the persistence of the malfunction. A persistent machine error is said to be solid, and one that is not persistent is said to be intermittent. In the case of a register or storage location, a third type of error must be considered, called externally generated. An externally generated error is one where no failure exists in the register or storage location but invalid CBC has been introduced into the location by actions external to the location. For example, the value could be affected by a power transient, or an incorrect value may have been introduced when the information was placed at the location.

Invalid CBC is preserved as invalid when information with invalid CBC is fetched or when an attempt is made to update only a portion of the checking block. When an attempt is made to replace the contents of the entire checking block and the block contains invalid CBC, it depends on the operation and the model whether the block remains with invalid CBC or is replaced. An operation which replaces the contents of a checking block with valid CBC, while ignoring the current contents, is called a validation operation. Validation is used to place a valid CBC in a register or at a location which has an intermittent or externally generated error.

Validating a checking block does not ensure that a valid CBC will be observed the next time the checking block is accessed. If the failure is solid, validation is effective only if the information placed in the checking block is such that the failing bits are set to the value to which they fail. If an attempt is made to set the bits to the state opposite to that in which they fail, then the validation will not be effective. Thus, for a solid failure, validation is only useful to eliminate the error condition, even though the underlying failure remains, thereby reducing the exposure to additional reports. The locations, however, cannot be used, since invalid CBC will result from attempts to store other values at the location. For an intermittent failure, however, validation is useful to restore a valid CBC such that a subsequent partial store into the checking block will be permitted. (A partial store is a store into a checking block without replacing the entire checking block.)

When a checking block consists of multiple bytes in storage, or multiple bits in CPU registers, the invalid CBC can be made valid only when all of the bytes or bits are replaced simultaneously.

For each type of field in the system, certain instructions are defined to validate the field. Depending on the model, additional instructions may also perform validation; or, in some models, a register is automatically validated as part of the machine-check-interruption sequence after the original contents of the register are placed in the appropriate save area.

When an error occurs in a checking block, the original information contained in the checking block should be considered lost even after validation. Automatic register validation leaves the contents unpredictable. Programmed and manual validation of checking blocks causes the contents to be changed explicitly.

Programming Note

The machine-check-interruption handler must assume that the registers require validation. Thus, each register should be loaded, using an instruction defined to validate, before the register is used or stored.

INVALID CBC IN STORAGE

The size of the checking block in storage depends on the model but is never more than 2K bytes.

When invalid CBC is detected in storage, a machine-check condition may occur; depending on the circumstances, the machine-check condition may be system damage, instruction-processing damage, external damage, or system recovery. If the invalid CBC is detected as part of the execution of a channel program, the error is normally reported as an I/O-error condition. When a CCW, indirect-data-address word, or data is prefetched from storage, is found to have invalid CBC, but is not used in the channel program, the condition is normally not reported as an I/O-error condition. The condition may or may not be reported as a machine-check-interruption condition. Invalid CBC detected during accesses to storage for other than CPU-related accesses may be reported as system recovery with storage error uncorrected indicated, or as external secondary report, since the primary error indication is reported by some other means.

When the storage checking block consists of multiple bytes and contains invalid CBC, special storage-validation procedures are generally necessary to restore or place new information in the checking block. Validation of storage is provided with the manual load-clear and system-reset-clear operations and may also be provided as a program function. Manual storage validation by clear reset validates all blocks which are available in the configuration.

A checking block with invalid CBC is never validated unless the entire contents of the checking block are replaced. An attempt to store into a checking block having invalid CBC, without replacing the entire checking block, leaves the data in the checking block (including the check bits) unchanged. Even when an instruction or a channel program input operation specifies that the entire contents of a checking block are to be replaced, validation may or may not occur, depending on the operation and the model.

Programming Note

Machine-check conditions may be reported for prefetched and unused data. Depending on the model, such situations may, or may not, be successfully retried. For example, a BRANCH AND LINK (BALR) instruction which specifies an R_2 field of zero will never branch, but on some models a prefetch of the location designated by register zero may occur. Access exceptions associated with this prefetch will not be reported. However, if an invalid checking-block code is detected, CPU retry may be attempted. Depending on the model, the prefetch may recur as part of the retry, and thus the retry will not be successful. Even when the CPU retry is successful, the performance degradation of such a retry is significant, and system recovery may be presented, normally with a failing-storage address. To avoid continued degradation, the program should initiate proceedings to eliminate use of the location and to validate the location.

Programmed Validation of Storage

Provided that an invalid CBC does not exist in the storage key associated with a 4K-byte block, the instruction TEST BLOCK causes the entire 4K-byte block to be set to zeros with a valid CBC, regardless of the current contents of the storage. TEST BLOCK thus removes an invalid CBC from a location in storage which has an intermittent, or one-time, failure. However, if a permanent failure exists in a portion of the storage, a subsequent fetch may find an invalid CBC.

When TEST BLOCK is installed, it will, in most cases, be the most effective instruction in validating storage. When TEST BLOCK is not installed, MOVE LONG, depending on the model, may prove effective.

Programming Note

The effectiveness of the following guideline depends on the model. On some models, instructions may be implemented that are more effective than the one listed here; however, the following approach is recommended when a model-dependent routine cannot be justified.

Execution of MOVE LONG will be most effective in validating 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 2K-byte boundary and is 2K bytes (or a multiple of 2K bytes) in length.
- The second-operand field, if nonzero in length, starts on a 2K-byte boundary and is 2K bytes (or a multiple of 2K bytes) in length.
- In general, the validation will be more effective if the second-operand field is of zero length. A nonzero-length second operand should be specified only if it is required to restore the contents of the block without introducing intermediate values.

An interruption or stopping of the CPU during execution of MOVE LONG does not affect the validation function performed.

INVALID CBC IN STORAGE KEYS

Depending on the model, each storage key may be contained in a single checking block, or the access-control and fetch-protection bits and the reference and change bits may be in separate checking blocks.

The figure "Invalid CBC in Storage Keys" describes the action taken when the storage key has invalid CBC. The figure indicates the action taken for the case when the access-control and fetch-protection bits are in one checking block and the reference and change bits are in a separate checking block. In machines where both fields are included in a single checking block, the action taken is the combination of the actions for each field in error, except that completion is permitted only if an error in all affected fields permits completion. References to main storage to which key-controlled protection does not apply are treated as if an access key of zero is used for the reference. This includes such references as channel-program references during initial program loading and implicit references, such as interruption action and DAT-table accesses.

Type of Reference	Action Taken on Invalid CBC	
	For Access-Control and Fetch-Protection Bits	For Reference and Change Bits
SET STORAGE KEY or SET STORAGE KEY EXTENDED	Complete; validate.	Complete; validate.
INSERT STORAGE KEY	PD; preserve.	PD in EC mode, CPF in BC mode; preserve.
INSERT STORAGE KEY EXTENDED	PD; preserve.	PD; preserve.
RESET REFERENCE BIT or RESET REFERENCE BIT EXTENDED	PD or complete; preserve.	PD; preserve.
INSERT VIRTUAL STORAGE KEY or TEST PROTECTION	PD; preserve.	CPF; preserve.
CPU prefetch (information not used)	CPF; preserve.	CPF; preserve.
Channel-program prefetch (information not used)	IPF; preserve.	IPF; preserve.
Fetch, nonzero access key	MC; preserve.	MC or complete; preserve.
Store, nonzero access key	MC[1]; preserve.	MC and preserve; or complete[3] and correct.
Fetch, zero access key[2]	MC or complete; preserve.	MC or complete; preserve.
Store, zero access key[2]	MC or complete; preserve.	MC and preserve; or complete[3] and correct.

Explanation:

[1] The contents of the main-storage location are not changed.

[2] The action shown for an access key of zero is also applicable to references to which key-controlled protection does not apply.

[3] The reference and change bits are set to ones if the "complete" action is taken.

Complete The condition does not cause termination of the execution of the instruction and, unless an unrelated condition prohibits it, the execution of the instruction is completed, ignoring the error condition. No machine-check-damage conditions are reported, but system recovery may be reported.

Correct The reference and change bits are set to ones with valid CBC.

Invalid CBC in Storage Keys (Part 1 of 2)

Explanation (Continued):

Preserve The contents of the entire checking block having invalid CBC are left unchanged.

Validate The entire key is set to the new value with valid CBC.

CPF Invalid CBC in the storage key for a CPU prefetch which is unused, or for instructions which do not examine the reference and change bits, may result in any of the following situations:

- The operation is completed; no machine-check condition is reported.
- The operation is completed; system recovery, with storage-key error uncorrected, is reported.
- Instruction-processing damage, with or without backup and with storage-key error uncorrected, is reported.

IPF Invalid CBC in the storage key for a channel-program prefetch which is unused may result in any of the following:

- The I/O operation is completed; no machine-check condition is reported.
- The I/O operation is completed; system recovery, with storage-key error uncorrected, is reported.
- An I/O-error condition is reported; no machine-check condition is reported.
- An I/O-error condition is reported; system recovery, with storage-key error uncorrected, is reported.
- The I/O operation is completed, or an I/O-error condition is reported; external damage, with or without storage-key error uncorrected, is reported.
- The I/O operation is completed, or an I/O-error condition is reported; external damage, with a valid external-damage code, with external secondary report, and with storage-key error uncorrected, is reported.

MC Same as PD for CPU references, but a channel-program reference may result in the following combinations of I/O-error conditions and machine-check conditions:

- An I/O-error condition is reported; no machine-check condition is reported.
- An I/O-error condition is reported; system recovery, with or without storage-key error uncorrected, is reported.
- The I/O operation is completed, or an I/O-error condition is reported; external damage, with or without storage-key error uncorrected, is reported.
- An I/O-error condition is reported; external damage, with a valid external-damage code, with external secondary report, and with storage-key error uncorrected, is reported.

PD Instruction-processing damage, with or without backup and with or without a storage-key error uncorrected, is reported.

Note: When storage-key error uncorrected is reported, a failing-storage address may or may not also be reported.

Invalid CBC in Storage Keys (Part 2 of 2)

INVALID CBC IN REGISTERS

When invalid CBC is detected in a CPU register, a machine-check condition may be recognized. CPU registers include the general, floating-point, and control registers, the current PSW, the prefix register, the TOD clock, the CPU timer, and the clock comparator.

When a machine-check interruption occurs, whether or not it is due to invalid CBC in a CPU register, the following actions affecting the CPU registers, other than the prefix register and the TOD-clock, are taken as part of the interruption.

1. The contents of the registers are saved in assigned storage locations. Any register which is in error is identified by a corresponding validity bit of zero in the machine-check-interruption code. Malfunctions detected during register saving do not result in additional machine-check-interruption conditions; instead, the correctness of all the information stored is indicated by the appropriate setting of the validity bits.

2. On some models, registers with invalid CBC are then validated, their actual contents being unpredictable. On other models, programmed validation is required.

The prefix register and the TOD clock are not stored during a machine-check interruption, have no corresponding validity bit, and are not validated.

On those models in which registers are not automatically validated as part of the machine-check interruption, a register with invalid CBC will not cause a machine-check-interruption condition unless the contents of the register are actually used. In these models, each register may consist of one or more checking blocks, but multiple registers are not included in a single checking block. When only a portion of a register is accessed, invalid CBC in the unused portion of the same register may cause a machine-check-interruption condition. For example, invalid CBC in the right half of a floating-point register may cause a machine-check-interruption condition if a LOAD (LE) operation attempts to replace the left half, or short form, of the register.

Invalid CBC associated with the 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 enter the check-stop state immediately 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 safely be reported by the machine-check interruption, since the interruption itself requires that the prefix value be applied to convert real addresses to the corresponding absolute addresses. Invalid CBC in the prefix register causes the CPU to enter the check-stop state immediately when the check-stop-control bit (control register 14, bit 0) is one. When the check-stop-control bit is zero, the machine is permitted to ignore even the most severe errors; thus, invalid CBC in the prefix register may be ignored or may cause the CPU to enter the check-stop state.

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. LOAD (L) and LOAD MULTIPLE validate if the operand is on a word boundary, and LOAD HALFWORD 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.

The CPU timer, clock comparator, and prefix register are validated by SET CPU TIMER, SET CLOCK COMPARATOR, and SET PREFIX, respectively.

The TOD clock is validated by SET CLOCK if the TOD-clock control is in the enable-set position.

Programming Note

Depending on the register, and the model, the contents of a register may be validated by the machine-check interruption or the model may require that a program execute a validating instruction after the machine-check interruption has occurred. In the case of the CPU timer, depending on the model, both the machine-check interruption and validating instructions may be required to restore the CPU timer to full working order.

CHECK-STOP STATE

In certain situations it is impossible or undesirable to continue operation when a machine error occurs. In these cases, the CPU may enter the check-stop state, which is indicated by the check-stop indicator.

In general, the CPU may enter the check-stop state whenever an uncorrectable error or other malfunction occurs and the machine is unable to recognize a specific machine-check-interruption condition.

The CPU always enters the check-stop state if the check-stop-control bit, bit 0 of control register 14, is one and if any of the following conditions exists:

- PSW bit 13 is zero and an exigent machine-check condition is generated.
- During the execution of an interruption due to one exigent machine-check condition, another exigent machine-check condition is detected.
- During a machine-check interruption, the machine-check-interruption code cannot be stored successfully, or the new PSW cannot be fetched successfully.
- Invalid CBC is detected in the prefix register.
- A malfunction in the receiving CPU, which is detected after accepting the order, prevents the successful completion of a SIGNAL PROCESSOR order and the order was a reset, or the receiving CPU cannot determine what the order was. The receiving CPU enters the check-stop state.

If the check-stop-control bit is zero when one of these conditions occurs, the CPU may or may not enter the check-stop state, depending on the model. There may be many other conditions for particular models when an error may cause check stop.

When the CPU is in the check-stop state, instructions and interruptions are not executed, the interval timer is not updated, and channel operations may be stopped. In systems with channel-set switching, I/O operations are normally not affected. The TOD clock is normally not affected by the check-stop state. The CPU timer may or may not run in the check-stop state, depending on the error and the model. The start key and stop key are not effective in this state.

The CPU may be removed from the check-stop state by CPU reset.

In a multiprocessing configuration, a CPU entering the check-stop state generates a request for a malfunction-alert external interruption to all CPUs in the configuration. Except for the reception of a malfunction alert, other CPUs and channels not connected to the malfunctioning CPU are normally unaffected by the check-stop state in a CPU. However, depending on the nature of the condition causing the check stop, other CPUs may also be delayed or stopped, and I/O activity for channels connected to other CPUs may be affected.

System Check Stop

In a multiprocessing configuration, some errors, malfunctions, and damage conditions are of such severity that the condition causes all CPUs in the configuration to enter the check-stop state. This condition is called a system check stop. The state of the channels is unpredictable.

Programming Note

The program should avoid setting the check-stop control, bit 0 of control register 14, to zero, since the machine may continue to operate rather than enter the check-stop state when extremely serious conditions, such as an error in the prefix register, occur.

MACHINE-CHECK INTERRUPTION

A request for a machine-check interruption, which is made pending as the result of a machine check, is called a machine-check-interruption condition. There are two types of machine-check-interruption conditions: exigent conditions and repressible conditions.

EXIGENT CONDITIONS

Exigent machine-check-interruption conditions are those in which damage has or would have occurred such that execution of the current instruction or interruption sequence cannot safely continue. Exigent conditions include two subclasses: instruction-processing damage and system damage. In addition to indicating specific exigent conditions, system damage is used to report any malfunction or error which cannot be isolated to a less severe report.

Exigent conditions for instruction sequences can be either nullifying exigent conditions or terminating exigent conditions, according to whether the instructions affected are nullified or terminated. Exigent conditions for interruption sequences are terminating exigent conditions. The terms "nullification" and "termination" have the same meaning as that used in Chapter 6, "Interruptions," except that more than one instruction may be involved. Thus, a nullifying exigent condition indicates that the CPU has returned to the beginning of a unit of operation prior to the error. A terminating exigent condition means that the results of one or more instructions may have unpredictable values.

REPRESSIBLE CONDITIONS

Repressible machine-check-interruption conditions are those in which the results of the instruction-processing sequence have not been affected. Repressible conditions can be delayed, until the completion of the current instruction or even longer, without affecting the integrity of CPU operation. Repressible conditions are of three groups: recovery, alert, and repressible damage. Each group includes one or more subclasses.

A malfunction in the CPU, storage, channel, or operator facilities which has been successfully corrected or circumvented internally without logical damage is called a recovery condition. Depending on the model and the type of malfunction, some or all recovery conditions may be discarded and not reported. Recovery conditions that are reported are grouped in one subclass, system recovery.

A machine-check-interruption condition not directly related to a machine malfunction is called an alert condition. The alert conditions are grouped in two subclasses: degradation and warning.

A malfunction resulting in an incorrect state of a portion of the system not directly affecting sequential CPU operation is called a repressible-damage condition. Repressible-damage conditions are grouped in five subclasses, according to the function affected: timing-facility damage, interval-timer damage, external damage, service-processor damage, and vector-facility failure.

Programming Notes

1. Even though repressible conditions are usually reported only at normal points of interruption, they may also be reported with exigent machine-check conditions. Thus, if an exigent machine-check condition causes an instruction to be abnormally terminated and a machine-check interruption occurs to report the exigent condition, any pending repressible conditions may also be reported. The meaningfulness of the validity bits depends on what exigent condition is reported.

2. Classification of damage as either exigent or repressible does not imply the severity of the damage. The distinction is whether action must be taken as soon as the damage is detected (exigent) or whether the CPU can continue processing (repressible). For a repressible condition, the current instruction can be completed before taking the machine-check interruption if the CPU is enabled for machine checks; if the CPU is disabled for machine checks, the condition can safely be kept pending until the CPU is again enabled for machine checks.

   For example, the CPU may be disabled for machine-check interruptions because it is handling an earlier instruction-processing-damage interruption. If, during that time, an I/O operation encounters a storage error, that condition can be kept pending because it is not expected to interfere with the current machine-check processing. If, however, the CPU also makes a reference to the area of storage containing the error before reenabling machine-check interruptions, another instruction-processing-damage condition is created, which is treated as an exigent condition and causes the CPU to enter the check-stop state, if the check-stop-control bit is set to one.

INTERRUPTION ACTION

A machine-check interruption causes the following actions to be taken. The PSW reflecting the point of interruption is stored as the machine-check old PSW at real location 48. The contents of other registers are stored in register-save areas at real locations 216-231 and 352-511. After the contents of the registers are stored in register-save areas, depending on the model, the registers may be validated with the contents being unpredictable. A failing-storage address may be stored at real location 248, an external-damage code may be stored at real location 244, and a region code may be stored at real location 252. A machine-check-interruption code (MCIC) of eight bytes is placed at real location 232. The new PSW is fetched from real location 112. Additionally, sometime before the storing of the MCIC, one or more machine-check logouts may have occurred. The machine-generated addresses to access the old and new PSW, the MCIC, extended interruption information, and the fixed-logout area are all real addresses. The machine-check extended-logout address is also a real address.

The fields accessed during the machine-check interruption are summarized in the figure "Machine-Check-Interruption Locations."

Information Stored (Fetched)	Starting Location*	Length in Bytes
Old PSW	48	8
New PSW (fetched)	112	8
Machine-check-interruption code	232	8
Register-save areas		
CPU timer	216	8
Clock comparator	224	8
Floating-point registers 0, 2, 4, 6	352	32
General registers 0-15	384	64
Control registers 0-15	448	64
Extended interruption information		
External-damage code	244	4
Failing-storage address	248	4
Region code	252	4
Logout areas		
Fixed logout	256	96
Machine-check extended logout (MCEL)	Note 1	Note 2

Explanation:

* All locations are in real storage.

1. The starting location of the MCEL is determined by the MCEL address in control register 15.

2. The length of the MCEL is model-dependent.

Machine-Check-Interruption Locations

If the machine-check-interruption code cannot be stored successfully or the new PSW cannot be fetched successfully, the CPU enters the check-stop state if the check-stop-control bit is one.

A repressible machine-check condition can initiate a machine-check interruption only if both PSW bit 13 is one and the associated subclass mask bit in control register 14 is also one. When it occurs, the interruption does not terminate the execution of the current instruction; the interruption is taken at a normal point of interruption, and no program or supervisor-call interruptions are eliminated. If the machine check occurs during the execution of a machine function, such as a CPU-timer update, the machine-check interruption takes place after the machine function has been completed.

When the CPU is disabled for a particular repressible machine-check condition, the condition remains pending. Depending on the model and the condition, multiple repressible conditions may be held pending for a particular subclass, or only one condition may be held pending for a particular subclass, regardless of the number of conditions that may have been detected for that subclass. When multiple external-damage conditions occur, each condition is retained.

When a repressible machine-check interruption occurs because the interruption condition is in a subclass for which the CPU is enabled, pending conditions in other subclasses may also be indicated in the same interruption code, even though the CPU is disabled for those subclasses. All indicated conditions are then cleared.

If a machine check which is to be reported as a system-recovery condition is detected during the execution of the interruption procedure due to a previous machine-check condition, the system-recovery condition may be combined with the other conditions, discarded, or held pending.

An exigent machine-check condition can cause a machine-check interruption only when PSW bit 13 is one. When a nullifying exigent condition causes a machine-check interruption, the interruption is taken at a normal point of interruption. When a terminating exigent condition causes a machine-check interruption, the interruption terminates the execution of the current instruction and may eliminate the program and supervisor-call interruptions, if any, that would have occurred if execution had continued. Proper execution of the interruption sequence, including the storing of the old PSW and other information, depends on the nature of the malfunction. When an exigent machine-check condition occurs during the execution of a machine function, such as a CPU-timer update, the sequence is not necessarily completed.

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 is detected, the 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 is detected during an interruption due to a repressible machine-check condition, system damage is 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.

Machine-check-interruption conditions are handled in the same manner regardless of whether the wait-state bit in the PSW is one or zero: a machine-check condition causes an interruption if the CPU is enabled for that condition.

Machine checks which occur while the rate control is set to the instruction-step position are handled in the same manner as when the control is set to the process position; that is, recovery mechanisms are active, and logout and machine-check interruptions occur when allowed. Machine checks occurring during a manual operation may be indicated to the operator, may generate a system-recovery condition, may be reported as an external secondary report, may result in system damage, or may cause a check stop, depending on the model.

Every reasonable attempt is made to limit the side effects of any machine check and the associated interruption. Normally, interruptions, as well as the progress of I/O operations, remain unaffected. The malfunction, however, may affect these activities, and, if the currently active PSW has bit 13 set to one, the machine-check interruption will indicate the total extent of the damage caused, and not just the damage which originated the condition.

POINT OF INTERRUPTION

The point in the processing which is indicated by the interruption and used as a reference point by the machine to determine and indicate the validity of the status stored is referred to as the point of interruption.

Because of the checkpoint capability in models with CPU retry, the interruption resulting from an exigent machine-check-interruption condition may indicate a point in the CPU processing sequence which is logically prior to the error. Additionally, the model may have some choice as to which point in the CPU processing sequence the interruption is indicated, and, in some cases, the status which can be indicated as valid depends on the point chosen.

Only certain points in the processing may be used as a point of interruption. For repressible machine-check interruptions, the point of interruption must be after one unit of operation is completed and any associated program or supervisor-call interruption is taken, and before the next unit of operation is begun.

Exigent machine-check conditions for instruction sequences are those in which damage has or would have occurred to the instruction stream. Thus, the damage can normally be associated with a point part way though an instruction, and this point is called the point of damage. In some cases there may be one or more instructions separating the point of damage and the point of interruption, and the processing associated with one or more instructions may be damaged. When the point of interruption is a point prior to the point of damage due to a nullifiable exigent machine-check condition, the point of interruption can be only at the same points as for repressible machine-check conditions.

Exigent machine-check conditions which are delayed (disallowed and presented later when allowed) can be presented only at the same points of interruption as repressible machine-check conditions. When a terminating exigent machine-check condition is not delayed, the point of interruption may also be after the unit of operation is completed but before any associated program or supervisor-call interruption occurs. In this case, a valid PSW instruction address is defined as that which would have been stored in the old PSW for the program or supervisor-call interruption. Since the operation has been terminated, the values in the result fields, other than the instruction address, are unpredictable. Thus the validity bits associated with fields which are due to be changed by the instruction stream are meaningless when a terminating exigent machine-check condition is reported.

When the point of interruption and the point of damage due to an exigent machine-check condition are separated by a checkpoint-synchronization function, the damage has not been isolated to a particular program, and system damage is indicated.

Programming Note

When an exigent machine-check-interruption condition occurs, the point of interruption which is chosen affects the amount of damage which must be indicated. An attempt is made, when possible, to choose a point of interruption which permits the minimum indication of damage. In general, the preference is the interruption point immediately preceding the error.

When all the status information stored as a result of an exigent machine-check-interruption condition does not reflect the same point, an attempt is made when possible to choose the point of interruption so that the instruction address which is stored in the machine-check old PSW is valid.

MACHINE-CHECK-INTERRUPTION CODE

On all machine-check interruptions, a machine-check-interruption code (MCIC) is stored at the doubleword starting at real location 232 and has the format shown in the figure "Machine-Check Interruption-Code Format."

Bits in the MCIC which are not assigned, or not implemented by a particular model, are stored as zeros.

SD	PD	SR	TD	CD	ED	VF	DG	W	0	SP	0	0	VS	B	D	SE	SC	KE	DS	WP	MS	PM	IA	FA	RC	EC	FP	GR	CR	LG	ST
0								8		10			13			16								24							31

IE	0	DA	0 0 0 0 0	0 0 0 0 0 0	CT	CC	MCEL Length	
32		34		40	46		48	63

Bits	Name
0	System damage (SD)
1	Instruction-processing damage (PD)
2	System recovery (SR)
3	Interval-timer damage (TD)
4	Timing-facility damage (CD)
5	External damage (ED)
6	Vector-facility failure (VF)
7	Degradation (DG)
8	Warning (W)
10	Service-processor damage (SP)
13	Vector-facility source (VS)
14	Backed up (B)
15	Delayed (D)
16	Storage error uncorrected (SE)
17	Storage error corrected (SC)
18	Storage-key error uncorrected (KE)
19	Storage degradation (DS)
20	PSW-EMWP validity (WP)
21	PSW mask and key validity (MS)
22	PSW program-mask and condition-code validity (PM)
23	PSW-instruction-address validity (IA)
24	Failing-storage-address validity (FA)
25	Region-code validity (RC)
26	External-damage-code validity (EC)
27	Floating-point-register validity (FP)
28	General-register validity (GR)
29	Control-register validity (CR)
30	Logout validity (LG)
31	Storage logical validity (ST)
32	Indirect storage error (IE)
34	Delayed-access exception (DA)
46	CPU-timer validity (CT)
47	Clock-comparator validity (CC)
48-63	Machine-check-extended-logout (MCEL) length

Note: All other bits of the MCIC are unassigned and stored as zeros.

Machine-Check Interruption-Code Format

SUBCLASS

Bits 0-8 and 10 are the subclass bits which identify the type of machine-check condition causing the interruption. At least one of the subclass bits is stored as a one. When multiple errors have occurred, several subclass bits may be set to ones.

System Damage

Bit 0 (SD), when one, indicates that damage has occurred which cannot be isolated to one or more of the less severe machine-check subclasses. When system damage is indicated, the remaining bits in the machine-check-interruption code are not meaningful, and information stored in the register-save areas and machine-check extended-interruption fields is not meaningful.

System damage is a terminating exigent condition and has no subclass-mask bit.

Instruction-Processing Damage

Bit 1 (PD), when one, indicates that damage has occurred to the instruction processing of the CPU.

The exact meaning of bit 1 depends on the setting of the backed-up bit, bit 14. When the backed-up bit is one, the condition is called processing backup. When the backed-up bit is zero, the condition is called processing damage. These two conditions are described in the section "Synchronous Machine-Check-Interruption Conditions" in this chapter.

Instruction-processing damage can be a nullifying or a terminating exigent condition and has no subclass-mask bit.

System Recovery

Bit 2 (SR), when one, indicates that malfunctions were detected but did not result in damage or have been successfully corrected. Some malfunctions detected as part of an I/O operation may result in a system-recovery condition in addition to an I/O-error condition. The presence and extent of the system-recovery capability depend on the model.

System recovery is a repressible condition. It is masked by the recovery subclass-mask bit, which is in bit position 4 of control register 14.

Programming Notes

1. System recovery may be used to report a failing-storage address detected by a CPU prefetch or by an I/O operation.

2. Unless the corresponding validity bits are ones, the indication of system recovery does not imply storage logical validity, or that the fields stored as a result of the machine-check interruption are valid.

Interval-Timer Damage

Bit 3 (TD), when one, indicates that damage has occurred to the interval timer or to the word at real storage locations 80-83.

Interval-timer damage is a repressible condition. It is masked by the external-damage subclass-mask bit, which is in bit position 6 of control register 14.

Timing-Facility Damage

Bit 4 (CD), when one, indicates that damage has occurred to the TOD clock, the CPU timer, the clock comparator, or to the CPU-timer or clock-comparator external-interruption conditions. The timing-facility-damage machine-check condition is set whenever any of the following occurs:

1. The TOD clock accessed by this CPU enters the error or not-operational state.

2. The CPU timer is damaged, and the CPU is enabled for CPU-timer external interruptions. On some models, this condition may be recognized even when the CPU is not enabled for CPU-timer interruptions. Depending on the model, the machine-check condition may be generated only as the CPU timer enters an error state. Or, the machine-check condition may be continuously generated whenever the CPU is enabled for CPU-timer interruptions, until the CPU timer is validated.

3. The clock comparator is damaged, and the CPU is enabled for clock-comparator external interruptions. On some models, this condition may be recognized even when the CPU is not enabled for clock-comparator interruptions.

Timing-facility damage may also be set along with instruction-processing damage when an instruction which accesses the TOD clock, CPU timer, or clock comparator produces incorrect results. Depending on the model, the CPU timer or clock comparator may be validated by the interruption which reports the CPU timer or clock comparator as invalid.

Timing-facility damage is a repressible condition. It is masked by the timing-facility subclass-mask bit, which is in bit position 6 of control register 14.

Programming Note

Timing-facility-damage conditions for the CPU timer and the clock comparator are not recognized on most models when these facilities are not in use. The facilities are considered not in use when the CPU is disabled for the corresponding external interruptions (PSW bit 7, or the subclass-mask bits, bits 20 and 21 of control register 0, are zeros), and when the corresponding set and store instructions are not executed. Timing-facility-damage conditions that are already pending remain pending, however, when the CPU is disabled for the corresponding external interruption.

Timing-facility-damage conditions due to damage to the TOD clock are always recognized.

External Damage

Bit 5 (ED), when one, indicates that damage has occurred to a channel or to storage during operations not directly associated with processing the current instruction. Channel malfunctions are reported as external damage only when the channel is unable to report the malfunctions by an I/O-error condition. 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.

When bit 5, external damage, is one and bit 26, external-damage-code validity, is also one, the external-damage code has been stored to indicate, in more detail, the cause of the external-damage machine-check interruption. When the external damage cannot be isolated to one or more of the conditions as defined in the external-damage code, or when the detailed indication for the condition is not implemented by the model, external damage is indicated with bit 26 set to zero. The presence and extent of reporting external damage depend on the model.

External damage is a repressible condition. It is masked by the external-damage subclass-mask bit, which is in bit position 6 of control register 14.

Vector-Facility Failure

Bit 6 (VF) of the machine-check-interruption code, when one, indicates that the vector facility has failed to such an extent that the service processor has made the facility not available.

This bit may be set to one regardless of whether the vector-control bit, bit 14 of control register 0, is one or zero.

Vector-facility failure is a repressible condition and has no subclass-mask bit.

Degradation

Bit 7 (DG), when one, indicates that continuous degradation of system performance, more serious than that indicated by system recovery, has occurred. Degradation may be reported when system-recovery conditions exceed a machine-preestablished threshold or when unit deletion has occurred. The presence and extent of the degradation-report capability depend on the model.

Degradation is a repressible condition. It is masked by the degradation subclass-mask bit, which is in bit position 5 of control register 14.

Warning

Bit 8 (W), when one, indicates that damage is imminent in some part of the system (for example, that power is about to fail, or that a loss of cooling is occurring). Whether warning conditions are recognized depends on the model.

If the condition responsible for the imminent damage is removed before the interruption request is honored (for example, if power is restored), the request does not remain pending, and no interruption occurs. Conversely, the request is not cleared by the interruption, and, if the condition persists, more than one interruption may result from the same condition.

Warning is a repressible condition. It is masked by the warning subclass-mask bit, which is in bit position 7 of control register 14.

Service-Processor Damage

Bit 10 (SP), when one, indicates that damage has occurred to the service processor. Service-processor damage may be made pending at all CPUs in the configuration, or it may be detected independently by each CPU. The presence and extent of reporting service-processor damage depend on the model.

Service-processor damage is a repressible condition and has no subclass-mask bit.

SUBCLASS MODIFIERS

Bits 13 (VS), 14 (B), 15 (D), and 34 (DA) of the machine-check-interruption code act as modifiers to the subclass bits.

Vector-Facility Source

Bit 13 (VS) of the machine-check-interruption code, when one, indicates that the vector facility is the source of the reported machine-check condition. Vector-facility source is reported together with instruction-processing damage. When this bit is one, the contents of vector-facility registers may have been damaged.

This bit may be set to one regardless of whether the vector-control bit, bit 14 of control register 0, is one or zero.

Bit 13 is not meaningful when vector-facility failure is reported.

Backed Up

Bit 14 (B), when one, indicates that the point of interruption is at a checkpoint before the point of error. This bit is meaningful only when the instruction-processing-damage bit, bit 1, is also set to one. The presence and extent of the capability to indicate a backed-up condition depend on the model.

Delayed

Bit 15 (D), 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 condition occurred. The bit may or may not apply to floating machine-check interruptions. The presence and extent of the capability to indicate a delayed condition depend on the model.

Delayed Access Exception

Bit 34 (DA), when one, indicates that an access exception was detected during a storage access using DAT when no such exception was detected by an earlier test for access exceptions.

Bit 34 is a modifier to instruction-processing damage (bit 1) and is meaningful only when bit 1 of the machine-check-interruption code is one. When bit 1 is zero, bit 34 has no meaning. The presence and extent of reporting delayed access exception depend on the model.

Programming Note

The occurrence of a delayed access exception normally indicates that the program is using an improper procedure to update the DAT tables.

SYNCHRONOUS MACHINE-CHECK-INTERRUPTION CONDITIONS

The instruction-processing damage and backed-up bits, bits 1 and 14 of the machine-check-interruption code, identify, in combination, two conditions.

Bit 1	Bit 14	Name of Condition
1	0	Processing damage
1	1	Processing backup

Processing Backup

The processing-backup condition indicates that the point of interruption is prior to the point, or points, of error. This is a nullifying exigent condition. When all of the other CPU-related-damage subclasses and modifiers of the machine-check-interruption code are zero and all of the validity bits associated with CPU status are indicated as valid, the machine has successfully returned to a checkpoint prior to the malfunction, and no damage has yet occurred to the CPU.

The subclass bits which must be zero for this to be the case are as follows:

MCIC Bit	Name
0	System damage
3	Interval-timer damage
4	Timing-facility damage
6	Vector-facility failure

The subclass-modifier bits which must be zero for this to be the case are as follows:

MCIC Bit	Name
13	Vector-facility source
34	Delayed-access exception

The validity bits in the machine-check-interruption code which must be one for this to be the case are as follows:

MCIC Bit	Fields Covered by Bit
20	PSW EMWP bits
21	PSW mask and key
22	PSW program mask and condition code
23	PSW instruction address
27	Floating-point registers
28	General registers
29	Control registers
31	Storage logical validity (result fields within current checkpoint interval)
46	CPU timer
47	Clock comparator

Programming Note

The processing-backup condition is reported rather than system recovery to indicate that a malfunction or failure stands in the way of continued operation of the CPU. The malfunction has not been circumvented, and damage would have occurred if instruction processing had continued.

Processing Damage

The processing-damage condition indicates that damage has occurred to the instruction processing of the CPU. The point of interruption is a point beyond some or all of the points of damage. Processing damage is a terminating exigent condition; therefore, the contents of result fields may be unpredictable and still indicated as valid.

Processing damage may include malfunctions in program-event recording, monitor call, and dynamic address translation. Processing damage causes any supervisor-call-interruption condition and program-interruption condition to be discarded. However, the contents of the old PSW and interruption-code locations for these interruptions may be set to unpredictable values.

STORAGE ERRORS

Bits 16-18 of the machine-check-interruption code are used to indicate an invalid CBC or a near-valid CBC detected in main storage or an invalid CBC in a storage key. Bit 19, storage degradation, may be indicated concurrently with bit 17. The failing-storage-address field, when indicated as valid, identifies a location within the storage checking block containing the error, or, for storage-key error uncorrected, within the block associated with the storage key. Bit 32, indirect storage error, may be set to one to indicate that the location designated by the failing-storage address is not the original source of the error.

The storage-error-uncorrected and storage-key-error-uncorrected bits do not in themselves indicate the occurrence of damage because the error detected may not have affected a result. The portion of the configuration affected by an invalid CBC is indicated in the subclass field of the machine-check-interruption code.

Storage errors detected for a channel, when indicated as I/O-error conditions, may also be reported as (1) system recovery, (2) external damage with the external-damage code valid or invalid, or (3) external secondary report. CBC errors that occur in storage or in the storage key and that are detected on prefetched or unused data for a CPU program may or may not be reported, depending on the model.

Storage Error Uncorrected

Bit 16 (SE), when one, indicates that a checking block in main storage contained invalid CBC and that the information could not be corrected. The contents of the checking block in main storage have not been changed. The location reported may have been accessed or prefetched for this CPU or another CPU or a channel, or it may have been accessed as the result of a model-dependent storage access.

Storage Error Corrected

Bit 17 (SC), when one, indicates that a checking block in main storage contained near-valid CBC and that the information has been corrected before being used. Depending on the model, the contents of the checking block in main storage may or may not have been restored to valid CBC. The location reported may have been accessed or prefetched for this CPU or for another CPU or for a channel, or it may have been accessed as the result of a model-dependent storage access. The presence and extent of the storage-error-correction capability depend on the model. This indication may or may not be accompanied by an indication of storage degradation, bit 19 (DS).

Storage-Key Error Uncorrected

Bit 18 (KE), when one, indicates that a storage key contained invalid CBC and that the information could not be corrected. The contents of the checking block in the storage key have not been changed. The storage key may have been accessed or prefetched for this CPU or for another CPU or for a channel, or it may have been accessed as the result of a model-dependent storage access.

Storage Degradation

Bit 19 (DS), when one, indicates that performance degradation has occurred for the reported storage-error-corrected condition.

Storage degradation indicates that although the associated storage error has been corrected, the correction process involved a substantial amount of time. Thus, this bit indicates that use of the associated block of storage should be avoided, if possible.

The indication of storage degradation has meaning only when bit 17, storage error corrected, is also one. The presence and extent of reporting storage degradation depend on the model.

Programming Note

Because storage degradation is reported with storage error corrected and, furthermore, because storage error corrected is normally reported with system recovery, the recovery subclass mask, bit 4 of control register 14, should be set to one in order for storage degradation to be indicated.

Indirect Storage Error

Bit 32 (IE), when one, indicates that the physical main-storage location identified by the failing-storage address is not the original source of the error. Instead, the error originated in another level of the storage hierarchy and has been propagated to the current physical-storage portion of the storage hierarchy. Bit 32 is meaningful only when bit 16 or 18 (storage error uncorrected or storage-key error uncorrected) of the machine-check-interruption code is one. When bits 16 and 18 are both zeros, bit 32 has no meaning.

For errors originating outside the storage hierarchy, the attempt to store is rejected, and the appropriate error indication is presented. When an error is detected during implicit movement of information inside the storage hierarchy, the action is not rejected and reported in this manner because the movement may be asynchronous and may be initiated as the result of an attempt to access completely unrelated information. Instead, errors in the contents of the source during implicit moving of information from one portion of the storage hierarchy to another may be preserved in the target area by placing a special invalid CBC in the checking block associated with the target location. These propagated errors, when detected later, are reported as indirect storage errors. The original source of such an error may have been in a cache associated with an I/O processor or a CPU, or the error may have been the result of a data-path failure in transmitting data from one portion of the storage hierarchy to another. Additionally, a propagated error may be generated during the movement of data from one physical portion of storage to another as the result of a storage-reconfiguration action.

The presence and extent of reporting indirect storage error depend on the model.

Programming Note

See the programming notes under TEST BLOCK in Chapter 10, "Control Instructions," for the action which should be taken after storage errors are reported.

MACHINE-CHECK INTERRUPTION-CODE VALIDITY BITS

Bits 20-31, 46, and 47 of the machine-check-interruption code are validity bits. Each bit indicates the validity of a particular field in storage. A validity bit is meaningless if the associated facility is not installed. With the exception of the storage-logical-validity bit (bit 31), each bit is associated with a field stored during the machine-check interruption. When a validity bit is one, it indicates that the saved value placed in the corresponding storage field is valid with respect to the indicated point of interruption and that no error was detected when the data was stored.

When a validity bit is zero, one or more of the following conditions may have occurred: the original information was incorrect, the original information had invalid CBC, additional malfunctions were detected while storing the information, or none or only part of the information was stored. Even though the information is unpredictable, the machine attempts, when possible, to place valid CBC in the storage field and thus reduce the possibility of additional machine checks being caused.

The validity bits for the floating-point registers, general registers, control registers, CPU timer, and clock comparator indicate the validity of the saved value placed in the corresponding save area. The information in these registers after the machine-check interruption is not necessarily correct even when the correct value has been placed in the save area and the validity bit set to one. The use of the registers and the operation of the facility associated with the control registers, CPU timer, and clock comparator, are unpredictable until these registers are validated. (See the section "Invalid CBC in Registers" earlier in this chapter.)

PSW-EMWP Validity

Bit 20 (WP), when one, indicates that the EMWP bits (bits 12-15) of the machine-check old PSW are correct.

PSW Mask and Key Validity

Bit 21 (MS), when one, indicates that the system mask, PSW key, and miscellaneous bits of the machine-check old PSW are correct. Specifically, this bit covers bits 0-11 of both the EC-mode and the BC-mode PSWs, and also bits 16, 17, and 24-39 of the EC-mode PSW.

PSW Program-Mask and Condition-Code Validity

Bit 22 (PM), when one, indicates that the program mask and condition code of the machine-check old PSW are correct.

PSW-Instruction-Address Validity

Bit 23 (IA), when one, indicates that the instruction address (bits 40-63) of the machine-check old PSW is correct.

Programming Note

When a machine check occurs which stores a BC-mode PSW, the contents of the interruption code and ILC in the machine-check old PSW are unpredictable, and no PSW-validity bit covers these bits. The four PSW-validity bits cover all 64 bits of the EC-mode PSW.

Failing-Storage-Address Validity

Bit 24 (FA), when one, indicates that a correct failing-storage address has been placed at real location 248 after a storage-error-uncorrected, storage-key-error-uncorrected, or storage-error-corrected condition has occurred. The presence and extent of the capability to identify the failing-storage location depend on the model. When no such errors are reported, that is, bits 16-18 of the machine-check-interruption code are zeros, the failing-storage address is meaningless, even though it may be indicated as valid.

Region-Code Validity

Bit 25 (RC), when one, indicates that a correct region code has been stored in the word at real location 252. The presence of the region code depends on the model. When a model does not provide a region code, bit 25 is set to zero.

External-Damage-Code Validity

Bit 26 (EC), when one, and provided that bit 5, external damage, is also one, indicates that a valid external-damage code has been stored in the word at real location 244. When bit 5 is zero, bit 26 has no meaning.

Floating-Point-Register Validity

Bit 27 (FP), when one, indicates that the contents of the floating-point-register save area at real locations 352-383 reflect the correct state of the floating-point registers at the point of interruption. When the floating-point facility is not installed, this bit is set to zero.

General-Register Validity

Bit 28 (GR), when one, indicates that the contents of the general-register save area at real locations 384-447 reflect the correct state of the general registers at the point of interruption.

Control-Register Validity

Bit 29 (CR), when one, indicates that the contents of the control-register save area at real locations 448-511 reflect the correct state of the control registers at the point of interruption.

Logout Validity

Bit 30 (LG), when one, indicates that the machine-check extended-logout information was correctly stored. When a model does not provide extended-logout information, bit 30 is set to zero.

Storage Logical Validity

Bit 31 (ST), when one, indicates that the storage locations, the contents of which are modified by the instructions being executed, contain the correct information relative to the point of interruption. That is, all stores before the point of interruption are completed, and all stores, if any, after the point of interruption are suppressed. When a store before the point of interruption is suppressed because of an invalid CBC, the storage-logical-validity bit may be indicated as one, provided that the invalid CBC has been preserved as invalid.

When instruction-processing damage is indicated but processing backup is not indicated, the storage-logical-validity bit has no meaning.

Storage logical validity reflects only the instruction-processing activity and does not reflect errors in the state of storage as the result of interval-timer update or I/O operations, or of the storing of the old PSW and other interruption information.

CPU-Timer Validity

Bit 46 (CT), when one, indicates that the CPU timer is not in error and that the contents of the CPU-timer save area at real location 216 reflect the correct state of the CPU timer at the time the interruption occurred. When the CPU-timer and clock-comparator facility is not installed, bit 46 is set to zero.

Clock-Comparator Validity

Bit 47 (CC), when one, indicates that the clock comparator is not in error and that the contents of the clock-comparator save area at real location 224 reflect the correct state of the clock comparator. When the CPU-timer and clock-comparator facility is not installed, bit 47 is set to zero.

Programming Note

The validity bits must be used in conjunction with the subclass bits and the backed-up bit in order to determine the extent of the damage caused by a machine-check condition. No damage has occurred to the system when all of the following are true:

- The four PSW-validity bits, the three register-validity bits, the two timing-facility-validity bits, and the storage-logical-validity bit are all ones if the facility with which they are associated is installed.
- Subclass bits 0, 3, 4, 5, 6, and 10 are zeros.
- The instruction-processing-damage bit is zero or, if one, the backed-up bit is also one.
- The vector-facility-source bit and the delayed-access-exception bit are zeros.

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 real location designated by the machine-check extended-logout address in control register 15. 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 EXTENDED INTERRUPTION INFORMATION

As part of the machine-check interruption, in some cases, extended interruption information is placed in fixed areas assigned in storage. The contents of registers associated with the CPU are placed in register-save areas. For external damage, additional information is provided for some models by storing an external-damage code. When storage error uncorrected, storage error corrected, or storage-key error uncorrected is indicated, the failing-storage address is saved. Some models store a region code to show the location of the error.

Each of these fields has associated with it a validity bit in the machine-check-interruption code. If, for any reason, the machine cannot store the proper information in the field, the associated validity bit is set to zero.

REGISTER-SAVE AREAS

As part of the machine-check interruption, the current contents of the CPU registers, except for the prefix register and the TOD clock, are stored in five register-save areas assigned in storage. Each of these areas has associated with it a validity bit in the machine-check-interruption code. If, for any reason, the machine cannot store the proper information in the field, the associated validity bit is set to zero.

The following are the five sets of registers and the real locations in storage where their contents are saved during a machine-check interruption.

Locations	Registers
216-223	CPU timer
224-231	Clock comparator
352-383	Floating-point registers 0, 2, 4, 6
384-447	General registers 0-15
448-511	Control registers 0-15

When the CPU-timer and clock-comparator facility or the floating-point facility is not installed, the corresponding locations remain unchanged. The information stored for unassigned or uninstalled control-register positions is unpredictable.

EXTERNAL-DAMAGE CODE

The word at real location 244 is the external-damage code. This field, when implemented and indicated as valid, describes the cause of external damage. The field is valid only when the external-damage bit and the external-damage-validity bit (bits 5 and 26 in the machine-check-interruption code) are both ones. The presence and extent of reporting an external-damage code depend on the model.

The external-damage code has the following format:

0 0	ES	SN	CC	ST	TT	0	XN	XF	0 0
0	2					7	8		10 31

External Secondary Report (ES): Bit 2, when one, indicates that the machine-check interruption has been reported for an external error for which the primary indication has been or will be made by means of some other report. The primary indication may be an I/O-error condition, an indication to the operator, another machine-check interruption, or even another bit in the same machine-check interruption.

External secondary report has three main purposes. First, it is used to present the failing-storage address associated with storage errors detected during channel accesses to storage. In this case, the failing-storage address and storage-error-uncorrected, storage-error-corrected, or storage-key-error-uncorrected indication are used to identify the cause of failure and the associated location.

Second, external secondary report is used to present model-dependent logout information for an error associated with a channel that is physically integrated with the CPU. The machine-check indication in this case is provided so that channels integrated with the CPU can use the normal CPU logout mechanism for presenting the model-dependent logout information.

For these two purposes, the primary error indication is normally by means of an I/O-error condition. These errors include conditions presented as channel-control check, channel-data check, and interface-control check. External secondary reports due to I/O and channel errors (1) may be presented to any or all CPUs in the configuration, (2) are not necessarily presented to the CPU to which the channel is connected, and (3) when channel-set switching is installed, may be presented even when the channel set is disconnected. In some models, external secondary reports due to I/O and channel errors may be broadcast to all CPUs in the configuration.

The third use of external secondary report is to provide a mechanism for presenting logout information associated with errors detected by other external devices or during operator-initiated operations. The primary indication in this case is normally by means of the external device or by an indication to the operator.

Channel Not Operational (CN): Bit 3, when one, indicates that one or more channels in the configuration have entered the not-operational state without signaling system reset to their attached devices. This situation occurs when these channels have detected an error of such severity that channel operations cannot continue. In configurations with channel-set switching, channel-not-operational conditions are reported to all CPUs in the configuration even when the channel set is disconnected. Only those state changes in the channel which would be seen if the channel set were connected to a CPU are considered for purposes of this interruption. The channel-not-operational condition is reported only in configurations in which all channels have implemented the recovery-extension facility.

Channel-Control Failure (CC): Bit 4, when one, indicates that one or more channels in the configuration have entered the not-operational state and may or may not have signaled system reset to their attached devices. This situation occurs when the channels have lost power or detected an error of such severity that channel operations cannot continue. In configurations with channel-set switching, channel-control-failure conditions are reported to all CPUs in the configuration, even when the channel set is disconnected. The channel-control-failure condition is reported only in configurations in which all channels have implemented the recovery-extension facility.

When the machine can determine that all affected channels actually entered the not-operational state without signaling system reset to their attached devices, the channel-not-operational condition is indicated rather than channel-control failure.

I/O-Instruction Timeout (ST): Bit 5, when one, indicates that the execution time of an I/O instruction has exceeded the maximum allowed by the CPU. The I/O instruction has been completed by setting condition code 3. When the CPU is enabled for external-damage machine-check conditions at the time the timeout occurs, and, if a program interruption for a PER event does not intervene, the instruction address stored in the machine-check old PSW (if indicated as valid) points to the instruction following the last executed I/O instruction. In this case, the address of the failing I/O instruction (or of EXECUTE) can be obtained by subtracting 4 from the instruction address. Timeout of an I/O instruction is reported by means of bit 5 only when the CPU can ensure that the channel has not signaled system reset to its attached devices. Depending on the channel and the timeout condition, the channel may or may not be operational. The I/O-instruction-timeout condition is reported only in configurations in which all channels have implemented the recovery-extension facility.

I/O-Interruption Timeout (TT): Bit 6, when one, indicates that the channel portion of an I/O interruption has exceeded the time limit established by the CPU and that the CPU has canceled the interruption. The I/O-interruption condition may or may not have been lost, and information may or may not have been stored at the locations of the old PSW, CSW, and other areas associated with an I/O interruption. The I/O interruption was not taken; that is, sequential instruction processing continued without loading the I/O new PSW. Timeout of an I/O interruption is reported by means of bit 6 only when the CPU can ensure that the channel has not signaled system reset to its attached devices. Depending on the channel and the timeout condition, the channel may or may not be operational. The I/O-interruption-timeout condition is reported only in configurations in which all channels have implemented the recovery-extension facility.

Expanded Storage Not Operational (XN): Bit 8, when one, indicates that the controller associated with some or all of the expanded storage in the configuration has become not operational.

Expanded-storage-not-operational conditions are reported to all CPUs in the configuration.

Expanded-Storage Control Failure (XF): Bit 9, when one, indicates that a malfunction has been detected in a controller associated with some or all of the expanded storage in the configuration. When expanded-storage control failure is indicated, the blocks of the expanded storage contain either the proper contents or a preserved error.

Expanded-storage-control-failure conditions are reported to all CPUs in the configuration.

Reserved: Bits 0, 1, 7, and 10-31 are reserved for future expansion and are always set to zeros.

Programming Notes

1. Bit 0 is reserved for future expansion and possible redefinition of the remaining bits in the external-damage code. Thus, the program should test bit 0 for a zero value before interpreting the other bits in the external-damage code.

2. Bit 3 (channel not operational), bit 4 (channel-control failure), and external damage with the external-damage code invalid, form a set of three errors of increasing severity. When a channel-not-operational or channel-control-failure condition is reported, the affected channels enter the not-operational state. Thus, if the program is aware of the channel addresses of all channels which have been operational in the configuration, then, by repeatedly executing the TEST CHANNEL instruction designating each channel in the configuration, the program can determine which channels have entered the not-operational state. Since the channel-not-operational and channel-control-failure conditions are reported to all CPUs in the configuration, all channels on all CPUs must be tested. When channel-set switching is installed, then all channels, including those not currently connected to any CPU, must be tested.

   Channel not operational is the least severe indication of the three. The affected channels can be determined as indicated above, and it is known in this case that system reset has not been signaled to the attached devices.

   Channel-control failure is more severe than channel not operational in that system reset may have been signaled to the attached devices.

   External damage with the external-damage code invalid is the most severe indication of the three. All channels in the configuration may have been affected, and the affected channels may or may not appear to be not operational when a TEST CHANNEL instruction is executed. Damage which can be reported by means of this indication includes errors occurring during the execution of an I/O interruption. For example, this indication can be used to report that an I/O interruption occurred with incorrect I/O address, incorrect CSW, incorrect full-channel logout, incorrect limited-channel-logout information, or channel-control failure.

3. On some models, a channel which has become channel not operational may be restored by executing CLEAR CHANNEL. See the programming note under "CLEAR CHANNEL," in Chapter 13, Input/Output Operations."

FAILING-STORAGE ADDRESS

When storage error uncorrected, storage error corrected, or storage-key error uncorrected is indicated in the machine-check-interruption code, the associated address, called the failing-storage address, is stored in bit positions 8-31 of the word at real location 248. Bits 0-7 of that word are set to zeros. When the extended-real-address facility is installed, the failing-storage address is 31 bits, and a zero is stored in bit position 0 of the word at real location 248. The field is valid only if the failing-storage-address validity bit, bit 24 of the machine-check-interruption code, is one.

In the case of storage errors, the failing-storage address may designate any byte within the checking block. For storage-key error uncorrected, the failing-storage address may designate any address within the block of storage associated with the storage key that is in error. When an error is detected in more than one location before the interruption, the failing-storage address may designate any of the failing locations. The address stored is an absolute address; that is, the value stored is the address that is used to reference storage after dynamic address translation and prefixing have been applied.

REGION CODE

Depending on the model, a region code may be stored in the word at real location 252. The field is valid only if the region-code-validity bit, bit 25 in the machine-check-interruption code, is one. The region code may contain 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.

HANDLING OF MACHINE-CHECK CONDITIONS

FLOATING INTERRUPTION CONDITIONS

An interruption condition which is made available to any CPU in a multiprocessing configuration is called a floating interruption condition. The first CPU that accepts the interruption clears the interruption condition, and it is no longer available to any other CPU in the configuration.

The service-signal external-interruption condition is a floating interruption condition. Depending on the model, some machine-check-interruption conditions associated with system recovery, warning, and external secondary report may be floating interruption conditions.

A floating interruption is presented to the first CPU in the configuration which is enabled for the interruption condition and can accept the interruption. A CPU cannot accept the interruption when it is in the check-stop state, has an invalid prefix, is performing an unending string of interruptions due to a PSW-format error of the type that is recognized early, is executing a READ DIRECT instruction, or is in the stopped state. However, a CPU with the rate control set to instruction step can accept the interruption when the start key is activated.

Programming Note

When a CPU enters the check-stop state in a multiprocessing configuration, the program on another CPU can determine whether a floating interruption may have been reported to the failing CPU and then lost. This can be accomplished if the interruption program places zeros in the real storage locations containing old PSWs and interruption codes after the interruption has been handled (or has been moved into another area for later processing). After a CPU enters the check-stop state, the program in another CPU can inspect the old-PSW and interruption-code locations of the failing CPU. A nonzero value in an old PSW or interruption code indicates that the CPU has been interrupted but the program did not complete the handling of the interruption.

Floating Machine-Check-Interruption Conditions

Floating machine-check-interruption conditions are reset only by the manually initiated resets through the operator facilities. When a machine check occurs which prohibits completion of a floating machine-check interruption, the interruption condition is no longer considered a floating interruption condition, and system damage is indicated.

MACHINE-CHECK MASKING

All machine-check interruptions are under control of the machine-check mask, PSW bit 13. In addition, some machine-check conditions are controlled by subclass masks in control register 14.

The exigent machine-check conditions (system damage and instruction-processing damage) are controlled only by the machine-check mask, PSW bit 13. When PSW bit 13 is one, an exigent condition causes a machine-check interruption. When PSW bit 13 is zero and the check-stop-control bit, bit 0 of control register 14, is one, the occurrence of an exigent machine-check condition causes the CPU to enter the check-stop state. When PSW bit 13 is zero and 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.

The repressible machine-check conditions, except vector-facility failure and service-processor damage, are controlled both by the machine-check mask, PSW bit 13, and by four subclass-mask bits in control register 14. If PSW bit 13 is one and one of the subclass-mask bits is one, the associated condition initiates a machine-check interruption. If a subclass-mask bit is zero, the associated condition does not initiate an interruption but is held pending. However, when a machine-check interruption is initiated because of a condition for which the CPU is enabled, those conditions for which the CPU is not enabled may be presented along with the condition which initiates the interruption. All conditions presented are then cleared.

Control register 14 contains mask bits that specify whether certain conditions can cause machine-check interruptions; it has the following format:

C S		RDEW MMMM	
0	1	4	7

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.

Programming Note

The program should avoid, whenever possible, operating with PSW bit 13, the machine-check mask, set to zero, since any exigent machine-check condition which is recognized during this situation may cause the CPU to enter the check-stop state. In particular, the program should avoid executing I/O instructions or allowing I/O interruptions with PSW bit 13 zero.

Check-Stop Control

Bit 0 (CS) of control register 14, controls the system action taken when an exigent machine-check condition occurs under one of the following two conditions:

1. The CPU is disabled for machine-check interruptions (that is, PSW bit 13 is zero).

2. An 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.

Recovery Subclass Mask

Bit 4 (RM) of control register 14 controls system-recovery interruption conditions. This bit is initialized to zero.

Degradation Subclass Mask

Bit 5 (DM) of control register 14 controls degradation interruption conditions. This bit is initialized to zero.

External-Damage Subclass Mask

Bit 6 (EM) of control register 14 controls timing-facility-damage, interval-timer-damage, and external-damage interruption conditions. This bit is initialized to one.

Warning Subclass Mask

Bit 7 (WM) of control register 14 controls warning interruption conditions. This bit is initialized to zero.

MACHINE-CHECK LOGOUT

Some models place model-dependent information in main storage as a result of a machine check. This 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.

Machine-check-logout information may, depending on the model, be placed in the machine-check extended-logout (MCEL) area. The starting real location of the MCEL area is designated by the contents of control register 15. The existence and length of the MCEL are model-dependent.

Some models may place model-dependent information in the fixed-logout area. This area is 96 bytes in length and starts at real location 256. The fixed logout may be in addition to or instead of an extended logout.

When a machine-check logout occurs during the machine-check interruption, it is called a synchronous logout. 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 CPU retry, then the logout is called an asynchronous logout.

To preserve the initial machine-check conditions, some models perform an asynchronous logout before invoking CPU retry. 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.

LOGOUT CONTROLS

Control register 14 contains bits which control when a logout may occur; it has the following format:

	SL	IL		AL	FL	
	1	2		8	9	

Synchronous Machine-Check Extended-Logout Control

Bit 1 (SL) of control register 14 controls the logout action during a machine-check interruption. When this bit is one, the machine-check extended-logout area may be changed during the interruption; when this 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

Bit 2 (IL) of control register 14, when one, permits channel logout into the I/O extended-logout area. When this bit is zero, I/O extended logouts cannot occur. Bit 2 of control register 14 is initialized to zero.

Asynchronous Machine-Check Extended-Logout Control

Bit 8 (AL) 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; when this bit is zero, the area may be changed only under control of the synchronous machine-check extended-logout-control bit, bit 1 of control register 14. Bit 8 of control register 14 is initialized to zero.

Asynchronous Fixed-Logout Control

Bit 9 (FL) 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. Bit 9 of control register 14 is initialized to zero.

MACHINE-CHECK EXTENDED-LOGOUT ADDRESS

Control register 15 contains the machine-check extended-logout address and has the following format:

	MCEL Address	
0	8	29 31

Bits 8-28 of control register 15, with three rightmost zeros appended, designate the starting real location of the machine-check extended-logout (MCEL) 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). When extended real addressing is installed, the MCEL address is still a 24-bit real address and is extended on the left with zeros. Thus, the machine-check extended logout can wrap from real location 2^{24} - 1 to real location 0.

When a model provides the machine-check extended logout (MCEL), control register 15 is implemented.

Programming Notes

1. The availability and extent of the machine-check extended-logout area differs among models and, for any particular model, may depend on the facilities 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 that results from executing STORE CPU ID indicates that no MCEL is provided.

2. 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.

3. Use of the machine-check extended-logout area while asynchronous machine-check extended logout is allowed may produce unpredictable results.

4. 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. CPU programs or channel programs storing 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*.

SUMMARY OF MACHINE-CHECK MASKING AND LOGOUT

A summary of machine-check masking and logout is given in the following three figures.

Machine-Check Condition		Sub-Class Mask	Action When CPU Disabled for Subclass and	
MCIC Bit	Subclass		Check-Stop Ctrl = 0	Check-Stop Ctrl = 1
0	System damage	-	P*	Check stop
1	Instruction-processing damage	-	P*	Check stop
2	System recovery	RM	Y	Y
3	Interval-timer damage	EM	P	P
4	Timing-facility damage	EM	P	P
5	External damage	EM	P	P
6	Vector-facility failure	-	P	P
7	Degradation	DM	P	P
8	Warning	WM	P	P
10	Service-processor damage	-	P	P

Explanation:

* System integrity may have been lost, and the system cannot be considered dependable.

- The condition does not have a subclass mask.

P Indication is held pending.

Y Indication may be held pending or may be discarded.

DM Degradation subclass mask (bit 5 of CR14).

EM External-damage subclass mask (bit 6 of CR14).

RM Recovery subclass mask (bit 4 of CR14).

WM Warning subclass mask (bit 7 of CR14).

Machine-Check-Condition Masking

PSW Bit 13	CR14 Bit 1 (SL)	CR14 Bit 8 (AL)	MCEL Action
0	X	X	MCEL does not occur.
1	0	0	MCEL does not occur.
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.

CR14 Bit 9 (FL)	Fixed-Logout Action
0	Fixed-logout area may be changed by the CPU only during machine-check interruption.[1]
1	Fixed-logout area may be changed at any time.

Explanation:

[1] Logout prior to instruction retry is not permissible in this state even though recovery reports are enabled.

[2] In some models, the asynchronous machine-check extended-logout control (AL) is ignored, and no logout occurs in this state.

AL Asynchronous machine-check extended-logout control.

FL Asynchronous fixed-logout control.

MCEL Machine-check extended logout.

SL Synchronous machine-check extended-logout control.

X Indicates that the same action occurs whether the bit is zero or one.

Machine-Check-Logout Control

Bit Description	Control Register 14 Bit Position	State of Bit on Initial CPU Reset
Check-stop control	0	1
Synchronous MCEL control	1	1
IOEL control	2	0
Recovery subclass mask	4	0
Degradation subclass mask	5	0
External-damage subclass mask	6	1
Warning subclass mask	7	0
Asynchronous MCEL control	8	0
Asynchronous fixed-logout control	9	0

Machine-Check Control-Register Bits

CHAPTER 12. OPERATOR FACILITIES

MANUAL OPERATION

The operator facilities provide functions for the manual operation and control of the machine. The functions include operator-to-machine communication, indication of machine status, control over the setting of the TOD clock, initial program loading, resets, and other manual controls for operator intervention in normal machine operation.

A model may provide additional operator facilities which are not described in this chapter. Examples are the means to indicate specific error conditions in the equipment, to change equipment configurations, and to facilitate maintenance. Furthermore, controls covered in this chapter may have additional settings which are not described here. Such additional facilities and settings may be described in the appropriate System Library publication.

Most models provide, in association with the operator facilities, a console device which may be used as an I/O device for operator communication with the program; this console device may also be used to implement some or all of the facilities described in this chapter.

The operator facilities may be implemented on different models in various technologies and configurations. On some models, more than one set of physical representations of some keys, controls, and indicators may be provided, such as on multiple local or remote operating stations, which may be effective concurrently.

A machine malfunction that prevents a manual operation from being performed correctly, as defined for that operation, may cause the CPU to enter the check-stop state or give some other indication to the operator that the operation has failed. Alternatively, a machine malfunction may cause a machine-check-interruption condition to be recognized.

BASIC OPERATOR FACILITIES

ADDRESS-COMPARE CONTROLS

The address-compare controls provide a way to stop the CPU when a preset address matches the address used in a specified type of main-storage reference.

One of the address-compare controls is used to set up the address to be compared with the storage address.

Another control provides at least two positions to specify the action, if any, to be taken when the address match occurs:

1. The normal position disables the address-compare operation.

2. The stop position causes the CPU to enter the stopped state on an address match. When the control is in this setting, the test indicator is on. Depending on the model and the type of reference, pending I/O, external, and machine-check interruptions may or may not be taken before entering the stopped state.

A third control may specify the type of storage reference for which the address comparison is to be made. A model may provide one or more of the following positions, as well as others:

1. The any position causes the address comparison to be performed on all storage references.

2. The data-store position causes address comparison to be performed when storage is addressed to store data.

3. The I/O position causes address comparison to be performed when storage is addressed by a channel to transfer data or to fetch a channel-command or indirect-data-address word. Whether references to the channel-address word or the channel-status word cause a match to be indicated depends on the model.

4. The instruction-address position causes address comparison to be performed when storage is addressed to fetch an instruction. The rightmost bit of the address setting may or may not be ignored. The match is indicated only when the first byte of the instruction is fetched from the selected location. It depends on the model whether a match is indicated when fetching the target instruction of EXECUTE.

Depending on the model and the type of reference, address comparison may be performed on virtual, real, or absolute addresses, and it may be possible to specify the type of address.

In a multiprocessing configuration, it depends on the model whether the address setting applies to one or all CPUs in the configuration and whether an address match causes one or all CPUs in the configuration to stop.

ALTER-AND-DISPLAY CONTROLS

The operator facilities provide controls and procedures to permit the operator to alter and display the contents of locations in storage, the storage keys, the general, floating-point, and control registers, the prefix, and the PSW.

Before alter-and-display operations may be performed, the CPU must first be placed in the stopped state. During alter-and-display operations, the manual indicator may be turned off temporarily, and the start and restart keys may be inoperative.

Addresses used to select storage locations for alter-and-display operations are real addresses. The capability of specifying logical, virtual, or absolute addresses may also be provided.

CHECK-STOP INDICATOR

The check-stop indicator is on when the CPU is in the check-stop state. Reset operations normally cause the CPU to leave the check-stop state and thus turn off the indicator. The manual indicator may also be on in the check-stop state.

IML CONTROLS

The IML controls provided with some models perform initial microprogram loading (IML).

The IML controls are effective while the power is on.

<u>Note</u>: The name "IMPL controls" was used in earlier descriptions.

INTERRUPT KEY

When the interrupt key is activated, an external-interruption condition indicating the interrupt key is generated. (See the section "Interrupt Key" in Chapter 6, "Interruptions.")

The interrupt key is effective when the CPU is in the operating or stopped state. It depends on the model whether the interrupt key is effective when the CPU is in the load state.

INTERVAL-TIMER CONTROL

The interval-timer control disables or enables operation of the interval timer. Disabling the interval timer does not affect any other facility.

When the control is set to the disable position, updating of real-storage locations 80-83 ceases. The contents of the interval timer remain at the last value to which they were updated, unless changed by a subsequent store operation. Depending on the model, any already-pending interval-timer-interruption condition is unaffected, is cleared, or is kept pending without regard to the state of the external mask, PSW bit 7, and the interval-timer mask, bit 24 of control register 0.

When the control is set to the enable position, updating of real-storage locations 80-83 is resumed by using the current contents. If an interval-timer-interruption request existed and was kept pending when the interval-timer control was last set to the disable position, that condition remains pending until the CPU is enabled for the interruption.

The enable position is considered the normal position. The test indicator may or may not be turned on when the interval-timer control is set to the disable position.

Programming Note

Disabling the interval timer allows execution of a program which uses real-storage locations 80-83 as ordinary storage. A program which does not use the interval timer will function correctly with the interval timer disabled, even when the interval timer fails.

LOAD INDICATOR

The load indicator is on during initial program loading, indicating that the CPU is in the load state. The indicator goes on for a particular CPU when the load-clear or load-normal key is activated for that CPU and the corresponding operation is started. It goes off after the new PSW is loaded successfully. For details, see the section "Initial Program Loading" in Chapter 4, "Control."

LOAD-CLEAR KEY

Activating the load-clear key causes a reset operation to be performed and initial program loading to be started by using the channel and I/O device designated by the load-unit-address controls. Clear reset is performed on the configuration. For details, see the sections "Resets" and "Initial Program Loading" in Chapter 4, "Control."

The load-clear key is effective when the CPU is in the operating, stopped, load, or check-stop state.

LOAD-NORMAL KEY

Activating the load-normal key causes a reset operation to be performed and initial program loading to be started by using the channel and I/O device designated by the load-unit-address controls. Initial CPU reset is performed on the CPU for which the load-normal key was activated, CPU reset is propagated to all other CPUs in the configuration, and a subsystem reset is performed on the remainder of the configuration. For details, see the sections "Resets" and "Initial Program Loading" in Chapter 4, "Control."

The load-normal key is effective when the CPU is in the operating, stopped, load, or check-stop state.

LOAD-UNIT-ADDRESS CONTROLS

The load-unit-address controls specify the I/O address of the channel and the device used for initial program loading. For details, see the section "Initial Program Loading" in Chapter 4, "Control."

MANUAL INDICATOR

The manual indicator is on when the CPU is in the stopped state. Some functions and several manual controls are effective only when the CPU is in the stopped state.

POWER CONTROLS

The power controls are used to turn the power on and off.

The CPUs, storage, channels, operator facilities, and I/O devices may all have their power turned on and off by common controls, or they may have separate power controls. When a particular unit has its power turned on, that unit is reset. The sequence is performed so that no instructions or I/O operations are performed until explicitly specified. The controls may also permit power to be turned on in stages, but the machine does not become operational until power on is complete.

When the power is completely turned on, an IML operation is performed on models which have an IML function. A power-on reset is then initiated (see the section "Resets" in Chapter 4, "Control").

RATE CONTROL

The setting of the rate control determines the effect of the start function and the manner in which instructions are executed.

The rate control has at least two positions. The normal position is the process position. Another position is the instruction-step position. When the rate control is set to the process position and the start function is performed, the CPU starts operating at normal speed. When the rate control is set to the instruction-step position and the wait-state bit is zero, one instruction or, for interruptible instructions, one unit of operation is executed, and all pending allowed interruptions are taken before the CPU returns to the stopped state. When the rate control is set to the instruction-step position and the wait-state bit is one, no instruction is executed, but all pending allowed interruptions are taken before the CPU returns to the stopped state. For details, see the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control."

The test indicator is on while the rate control is not set to the process position.

If the setting of the rate control is changed while the CPU is in the operating or load state, the results are unpredictable.

RESTART KEY

Activating the restart key initiates a restart interruption. (See the section "Restart Interruption" in Chapter 6, "Interruptions.")

The restart key is effective when the CPU is in the operating or stopped state. The key is not effective when the CPU is in the check-stop state. It depends on the model whether the restart key is effective when any CPU in the configuration is in the load state.

The effect is unpredictable when the restart key is activated while any CPU in the configuration is in the load state. In particular, if the CPU performs a restart interruption and enters the operating state while another CPU is in the load state, operations such as I/O instructions, the SIGNAL PROCESSOR instruction, and the INVALIDATE PAGE TABLE ENTRY instruction may not operate according to the definitions given in this publication.

START KEY

Activating the start key causes the CPU to perform the start function. (See the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control.")

The start key is effective only when the CPU is in the stopped state. The effect is unpredictable when the stopped state has been entered by a reset.

STOP KEY

Activating the stop key causes the CPU to perform the stop function. (See the section "Stopped, Operating, Load, and Check-Stop States" in Chapter 4, "Control.")

The stop key is effective only when the CPU is in the operating state.

Operation Note

Activating the stop key has no effect when:

- An unending string of certain program or external interruptions occurs.
- The prefix register contains an invalid address.
- The CPU is in the load or check-stop state.
- A READ DIRECT instruction cannot be completed.

STORE-STATUS KEY

Activating the store-status key initiates a store-status operation. (See the section "Store Status" in Chapter 4, "Control.")

The store-status key is effective only when the CPU is in the stopped state.

Operation Note

The store-status operation may be used in conjunction with a standalone dump program for the analysis of major program malfunctions. For such an operation, the following sequence would be called for:

1. Activation of the stop or system-reset-normal key

2. Activation of the store-status key

3. Activation of the load-normal key to enter a standalone dump program

The system-reset-normal key must be activated in step 1 when (1) the stop key is not effective because a continuous string of interruptions is occurring, (2) the prefix register contains an invalid address, (3) a READ DIRECT instruction cannot be completed, or (4) the CPU is in the check-stop state.

SYSTEM-RESET-CLEAR KEY

Activating the system-reset-clear key causes a clear-reset operation to be performed. Clear reset is propagated to all CPUs and storage units in the configuration, and a subsystem reset is performed on the remainder of the configuration. For details, see the section "Resets" in Chapter 4, "Control."

The system-reset-clear key is effective when the CPU is in the operating, stopped, load, or check-stop state.

SYSTEM-RESET-NORMAL KEY

When the store-status facility is not installed, activating the system-reset-normal key causes an initial-CPU-reset operation and a subsystem-reset operation to be performed. When the store-status facility is installed, activating the system-reset-normal key causes a CPU-reset operation and a subsystem-reset operation to be performed. In a multiprocessing configuration, a CPU reset is propagated to all CPUs in the configuration. For details, see the section "Resets" in Chapter 4, "Control."

The system-reset-normal key is effective when the CPU is in the operating, stopped, load, or check-stop state.

TEST INDICATOR

The test indicator is on when a manual control for operation or maintenance is in an abnormal position that can affect the normal operation of a program.

Setting the address-compare controls or the check control to the stop position or setting the rate control to the instruction-step position turns on the test indicator. Setting the interval-timer control to the disable position may or may not turn on the test indicator.

The test indicator may be on when one or more diagnostic functions under the control of DIAGNOSE are activated, or when other abnormal conditions occur.

Operation Note

If a manual control is left in a setting intended for maintenance purposes, such an abnormal setting may, among other things, result in false machine-check indications or cause actual machine malfunctions to be ignored. It may also alter other aspects of machine operation, including instruction execution, channel operation, and the functioning of operator controls and indicators, to the extent that operation of the machine does not comply with that described in this publication.

The abnormal setting of a manual control causes the test indicator of the affected CPU to be turned on; however, in a multiprocessing configuration, the operation of other CPUs may be affected even though their test indicators are not turned on.

TOD-CLOCK CONTROL

When the TOD-clock control is not activated, that is, the control is set to the secure position, the state and value of the TOD clock are protected against unauthorized or inadvertent change by not permitting the instructions SET CLOCK or DIAGNOSE to change the state or value.

When the TOD-clock control is activated, that is, the control is set to the enable-set position, alteration of the clock state or value by means of SET CLOCK or DIAGNOSE is permitted. This setting is momentary, and the control automatically returns to the secure position.

In a multiprocessing configuration, activating the TOD-clock control enables all TOD clocks in the configuration to be set. If there is more than one physical representation of the TOD-clock control, no TOD clock is secure unless all TOD-clock controls in the configuration are set to the secure position.

WAIT INDICATOR

The wait indicator is on when the wait-state bit in the current PSW is one.

MULTIPROCESSING CONFIGURATIONS

In a multiprocessing configuration, one of each of the following keys and controls is provided for each CPU: alter and display, interrupt, rate, restart, start, stop, and store status. The load-clear key, load-normal key, and load-unit-address controls are provided for each CPU capable of performing I/O operations. Alternatively, a single set of initial-program-loading keys and controls may be used together with a control to select the desired CPU.

There need not be more than one of each of the following keys and controls in a multiprocessing configuration: address compare, check, IML, interval timer, power, system reset clear, system reset normal, and TOD clock.

One check-stop, manual, test, and wait indicator is provided for each CPU. A load indicator is provided only on a CPU capable of performing I/O operations. Alternatively, a single set of indicators may be switched to more than one CPU.

In a system capable of reconfiguration, there must be a separate set of keys, controls, and indicators in each configuration.

CHAPTER 13. INPUT/OUTPUT OPERATIONS

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 chapter 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 applicable only to some I/O-device types, such as erasing a gap on a magnetic-tape unit. The way in which a device makes use of the format is defined in the System Library (SL) publication for the particular device.

Almost all storage references for I/O operations are references to absolute storage. Throughout this chapter, unless indicated otherwise, "storage" means absolute storage, and "address" means absolute address. The terms "I/O address," "channel address," and "device address" are never abbreviated to "address" in this publication.

ATTACHMENT OF INPUT/OUTPUT DEVICES

INPUT/OUTPUT DEVICES

Input/output devices provide external storage and a means of communication between data-processing systems or between a system and its environment. Input/output devices include such equipment as card readers, card punches, magnetic-tape units, direct-access-storage devices (disks and drums), display units, typewriter-keyboard devices, printers, teleprocessing devices, and sensor-based equipment.

Most types of I/O devices, such as printers, card equipment, or tape devices, deal directly with external media, and these devices are physically distinguishable and identifiable. Other types consist only of electronic equipment and do not directly handle physical recording media. The channel-to-channel adapter, for example, provides a channel-to-channel data-transfer path, and the data never reaches a physical recording medium outside main storage. Similarly, a communications controller handles transmission of information between the data-processing system and a remote station, and its input and output are signals on a transmission line. An I/O device may be physically distinct equipment, or it may time-share equipment with other I/O devices.

An input/output device ordinarily is attached to one control unit and is accessible from one channel. Switching equipment is available to make some devices accessible to two or more channels by switching devices between control units and control units between channels. The time required for switching occurs during device-selection time and may be ignored.

CONTROL UNITS

A control unit provides the logical capabilities necessary to operate and control an I/O device and adapts the characteristics of each device to the standard form of control provided by the channel.

The control unit accepts control signals from the channel, controls the timing of data transfer, and provides indications concerning the status of the device.

The I/O device attached to the control unit may be designed to perform only certain limited operations, or it may perform many different operations. A typical operation is moving the recording medium and recording data. To accomplish these functions, the device needs detailed signal sequences peculiar to the type of device. The control unit decodes the commands received from the channel, interprets them for the particular type of device, and provides the signal sequence required for execution of the operation.

A control unit may be housed separately, or it may be physically and logically integral with the I/O device or the CPU. In 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 transfer data at a time (magnetic-tape units or disk-access mechanisms, for example), and the control unit is shared among a number of I/O devices. On the other hand, in some electronic I/O devices such as the channel-to-channel adapter, the control unit does not have an identity of its own.

From the programmer's point of view, most functions performed by the control unit can be merged with those performed by the I/O device. Therefore, this publication normally does not make specific mention of the control-unit function; the execution of I/O operations is described as if the I/O devices communicated directly with the channel. Reference is made to the control unit only when emphasizing a function performed by it or when describing how sharing of the control unit among a number of devices affects the execution of I/O operations.

CHANNELS

A channel directs the flow of information between I/O devices and main storage. It relieves the CPU of the task of communicating directly with the devices and permits data processing to proceed concurrently with I/O operations.

A channel provides a means for connecting various types of I/O devices to the CPU and to storage. The channel 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 and device. Similarly, when an I/O device provides signals that should be brought to the attention of the program, the channel transforms the signals to information that can be used in the CPU.

A channel contains facilities for the control of I/O operations. During execution of an I/O operation involving data transfer, the channel assembles or disassembles data and synchronizes the transfer of data bytes with storage cycles. To accomplish this, the channel maintains and updates an address and a count that describe the destination or source of data in storage. When the channel facilities are provided in the form of separate autonomous equipment designed specifically to control I/O devices, I/O operations are completely overlapped with the activity in the CPU. The only storage cycles required during I/O operations in such channels are those needed to transfer data and control information to or from the final locations in storage. These cycles do not delay the CPU program, except when both the CPU and the channel concurrently attempt to refer to the same storage area.

If separate equipment is not provided, facilities of the CPU are used for controlling I/O devices. When the CPU and channels, or the CPU, channels, and control units, share common facilities, I/O operations cause interference to the CPU, varying in intensity from occasional delay of a CPU cycle to a complete lockout of CPU activity. The intensity depends on the extent of sharing and on the I/O data rate. The sharing of the facilities, however, is accomplished automatically, and the program is not affected by CPU delays, except for an increase in execution time.

Modes of Operation

An I/O operation occurs in one of two modes: burst or byte-multiplex.

In burst mode, the I/O device monopolizes the channel and stays logically connected to the channel for the transfer of a burst of information. No other device can communicate with the channel during the time a burst is transferred. The burst can consist of a few bytes, a whole block of data, a sequence of blocks with associated control and status information (the block lengths may be zero), or status information | which monopolizes the channel.

Some channels can tolerate an absence of data transfer during a burst-mode operation, such as occurs when reading a long gap on magnetic tape, for not more than approximately 1/2 minute. Equipment malfunction may be indicated when an absence of data transfer exceeds this time.

In byte-multiplex mode, the I/O device stays logically connected to the channel only for a short interval of time. The facilities in a channel capable of operating in byte-multiplex mode may be shared by a number of concurrently operating I/O devices. In this mode, all I/O operations are split into short intervals of time during which only a segment of information is transferred. During such an interval, only one device is logically connected to the channel. The intervals associated with the concurrent operation of multiple I/O devices are sequenced in response to demands from the devices. The channel controls are occupied with any one operation only for the time required to transfer a segment of information. The segment can consist of a single byte of data, a few bytes of data, a status report from the device, or a control sequence used for initiation of a new operation.

Operation in burst and byte-multiplex modes is differentiated because of the way the channels respond to I/O instructions. A channel operating a device in the burst mode may appear busy to new I/O instructions, whereas a channel operating one or more devices in the byte-multiplex mode is capable of initiating an operation on another device. If a channel that can operate in either mode is communicating with an I/O device at the instant a new I/O instruction is issued, action on the instruction is delayed by the channel until the current mode of operation is established. Furthermore, the new I/O operation is initiated only after the channel has serviced all outstanding requests from devices previously placed in operation.

The distinction between a short burst of data occurring in the byte-multiplex mode and an operation in the burst mode is in the length of the bursts of data. A channel that can operate in either mode determines its mode of operation by timeout. Whenever the burst causes the device to be connected to the channel for more than approximately 100 microseconds, the channel is considered to be operating in the burst mode.

Ordinarily, devices with a high data-transfer rate operate with the channel in burst mode, and slower devices run in byte-multiplex mode. Some control units have a manual switch for setting the mode of operation.

Types of Channels

A system can be equipped with three types of channels: selector, byte multiplexer, and block multiplexer.

The channel facilities required for sustaining a single I/O operation are termed a subchannel. The subchannel consists of internal storage used for recording the addresses, count, and any status and control information associated with the I/O operation. The capability of a channel to permit multiplexing depends upon whether it has more than one subchannel.

A selector channel, which contains a minimum of facilities, has one subchannel and always forces the I/O device to transfer data in the burst mode. The burst extends over the whole block of data, or, when command chaining is specified, over the whole sequence of blocks. A selector channel cannot perform any multiplexing and therefore can be involved in only one I/O operation or chain of operations at a time. In the meantime, other I/O devices attached to the channel can be executing previously initiated operations that do not involve communication with the channel, such as backspacing tape. When the selector channel is not executing an operation or a chain of operations and is not processing an interruption, it monitors the attached devices for status information.

A byte-multiplexer channel contains multiple subchannels and can operate at any one time in either byte-multiplex or burst mode. A byte-multiplexer channel operates most efficiently with I/O devices that are designed to operate in byte-multiplex mode. The mode of operation is determined by the I/O device, and, during data transfer, the mode can change at any time. Unless data transfer is occurring, the mode of operation has no meaning. The data transfer associated with an operation can occur partially in the byte-multiplex mode and partially in the burst mode.

A block-multiplexer channel contains multiple subchannels and can only operate in burst mode. A block-multiplexer channel operates most efficiently with devices that are designed to operate in burst mode. When multiplexing is not inhibited, the channel permits multiplexing between bursts, between blocks when command chaining is specified, or when command retry is performed. On most models, the burst is forced to extend over the block of data, and multiplexing occurs between blocks of data when command chaining is specified. Whether or not multiplexing occurs depends on the design of the channel and I/O device and on the state of the block-multiplexing-control bit.

When the block-multiplexing-control bit, bit 0 of control register 0, is zero, multiplexing is inhibited; when it is one, multiplexing is allowed.

Whether a block-multiplexer channel executes an I/O operation with multiplexing inhibited or allowed is determined by the state of the block-multiplexing-control bit at the time the operation is initiated by START I/O or START I/O FAST RELEASE and applies to that operation until the involved subchannel becomes available.

For brevity, the term "multiplexer channel" is used hereafter when describing a function or facility that is common to both the byte-multiplexer and the block-multiplexer channel. 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 the attached devices for data and for status information.

Subchannels on a multiplexer channel may be either <u>nonshared</u> or <u>shared</u>.

A subchannel is referred to as nonshared if it is associated with and can be used only by a single I/O device. A nonshared subchannel is used with devices that do not have any restrictions on the concurrency of channel-program operations, such as a single drive of an IBM 3330 Disk Storage.

A subchannel is referred to as shared if data transfer to or from a set of devices implies the use of the same subchannel. Only one device associated with a shared subchannel may be involved in data transmission at a time. Shared subchannels are used with devices, such as magnetic-tape units or some display devices, 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, the IBM 3880 storage control recognizes 64 device addresses, each of which is assigned a nonshared subchannel.

<u>Programming Note</u>

A block-multiplexer channel can be made to operate as a selector channel by the appropriate setting of the block-multiplexing-control bit. However, since a block-multiplexer channel inherently can interleave the execution of multiple I/O operations and since the state of the block-multiplexing-control bit can be changed at any time, it is possible to have one or more operations that permit multiplexing and an operation that inhibits multiplexing being executed simultaneously by a channel.

Therefore, to ensure complete compatibility with selector channel operation, all operational subchannels on the block-multiplexer channel must be available or operating with multiplexing inhibited when the use of that channel as a selector channel is begun. All subsequent operations should then be initiated with the block-multiplexing-control bit inhibiting multiplexing.

I/O-SYSTEM OPERATION

Input/output operations are initiated and controlled by information with two types of formats: instructions and channel-command words (CCWs). <u>Instructions</u> are decoded by the CPU and are part of the CPU program. <u>CCWs</u> are decoded and executed by the channels and I/O devices and initiate I/O operations, such as reading and writing. One or more CCWs arranged for sequential execution form a channel program. Both instructions and CCWs are fetched from storage. The formats of CCWs are common for all types of I/O devices, although the modifier bits in the command code of a CCW may specify device-dependent operations.

The CPU program initiates I/O operations with the instruction START I/O or START I/O FAST RELEASE. These instructions identify the channel and the I/O device and cause the channel to fetch the channel-address word (CAW) from a fixed location in real storage. The CAW contains the subchannel key and suspend-control bit and designates the location in storage from which the channel subsequently fetches the first CCW. The CCW specifies the command to be executed and the storage area, if any, to be used.

When START I/O is executed and the addressed channel and subchannel are available and when the suspend flag is not specified in the CCW, the channel attempts to select the I/O device and sends the command-code part of the CCW to the control unit. The device responds indicating whether it can execute the command. If the suspend flag is specified, the command code is not sent to the device, and, depending on the circumstances, the operation is either suspended or terminated instead.

At this time, the execution of START I/O is completed. The results of the attempt to initiate the execution of the command are indicated by setting the condition code in the PSW and, in certain situations, by storing pertinent information in the channel-status word (CSW).

When START I/O FAST RELEASE is executed, the functions performed during the execution of the instruction depend on the design of the channel. Some channels perform the same functions as for START I/O; other channels release the CPU (that is, complete the execution of the instruction) before the I/O operation has been initiated at the addressed device. Channels are permitted to release the CPU as early as when the CAW has been fetched and validated. Channels designed to release the CPU before the I/O operation is initiated at the I/O device perform the functions associated with I/O operation initiation logically subsequent and asynchronous to the execution of START I/O FAST RELEASE. When the CPU is released, the results of the execution of the instruction to that point are indicated by setting the condition code in the PSW and, in certain situations, by storing pertinent information in the CSW.

If the I/O operation is initiated at the I/O 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 operations that do not require any data to be transferred to or from the device, the device may signal the end of the operation immediately on receipt of the command code.

An I/O operation may involve transfer of data to one storage area, designated by a single CCW, or to a number of noncontiguous storage areas. In the latter case, generally a list of CCWs is used for execution of the I/O operation, each CCW designating a contiguous storage area, and the CCWs are said to be coupled by data chaining. Data chaining is specified by a flag in the CCW and causes the channel to fetch another CCW upon the exhaustion or filling of the storage area designated by the current CCW. The storage area designated by a CCW fetched on data chaining pertains to the I/O operation already in progress at the I/O device, and the I/O device is not notified when a new CCW is fetched.

Provision is made in the CCW format for the programmer to specify that, when the CCW is decoded, the channel request an I/O interruption as soon as possible, thereby notifying the CPU program that chaining has progressed at least as far as that CCW.

To complement the dynamic-address-translation facility available in the CPU, 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 2K-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 conclusion of an I/O operation normally is indicated by channel end and device end. When channel end is presented, it means that the I/O device has received or provided all data associated with the operation and no longer needs channel facilities. When device end is presented, it usually means that the I/O device has concluded execution of the I/O operation. On some I/O devices, for reasons of performance, device end is presented before the I/O operation has been concluded. Device end can occur concurrently with channel end or later.

Operations that keep the control unit busy after releasing channel facilities may, in some situations, cause a third indication called control-unit end. Control-unit end may occur only concurrently with or after channel end.

Concurrent with channel end, both the channel and the I/O device can provide indications of unusual situations. Control-unit end and device end can be accompanied by error indications from the I/O device.

The indication of the conclusion of an I/O operation can be brought to the attention of the program by an I/O interruption or, when the CPU is disabled for I/O interruptions from the channel, by programmed interrogation of the I/O device. An indication that will result in a request for an I/O interruption is called an interruption condition. In either case, a CSW is stored, which contains additional information concerning the execution of the operation. When channel end is indicated in the CSW and no equipment malfunctions have been detected, the CSW identifies the last CCW used and provides its residual byte count, thus indicating the extent of storage used.

Facilities are provided for the program to initiate the execution of a chain of I/O operations with a single START I/O or START I/O FAST RELEASE instruction. 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, if the suspend flag is not specified in the new CCW, to initiate execution of a new command at the device. If the suspend flag is specified, execution of the new command is not initiated, and command chaining is terminated. Execution of the new command is initiated by the channel in the same way as the previous operation. Channel end and device end are not presented to the program when command chaining causes execution of another I/O operation to be initiated. However, unusual situations can cause premature termination of command chaining and generation of an I/O-interruption condition.

Activities that generate I/O-interruption conditions are asynchronous to activity in the CPU, and more than one I/O-interruption condition can exist at the same time. The channel and the CPU establish priority among the conditions so that only one condition is presented to the CPU at a time.

The execution of an I/O operation or chain of I/O operations 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 time condition code 0 is set for the START I/O or START I/O FAST RELEASE until the CPU has accepted the I/O interruption signaling that the I/O operation or, for chained operations, the last operation has been completed at the subchannel.

3. The control unit may remain busy after the execution has completed at the subchannel and may generate control-unit end when it becomes free.

4. The I/O device is busy from the initiation of the first operation at the I/O device until the interruption condition caused by the device end associated with the operation is cleared from the I/O device.

An interruption condition caused by device end blocks the initiation of an I/O operation with the I/O device, but normally does not affect the state of any other part of the system. An interruption condition caused by control-unit end may block communications through the control unit to any device attached to it, and an interruption condition caused by channel end normally blocks all communications through the subchannel.

In some system models, a suspend-and-resume facility may be provided on an individual subchannel basis for nonshared subchannels. The mechanism for suspending channel-program execution provides the program a controlling function over the execution of a channel program. The initiation of the suspend function is controlled by the setting of the suspend-control bit in the CAW. The suspend function is signaled to the channel during channel-program execution by a flag (that is, a bit set to one) in the CCW.

Suspension occurs when the channel fetches a CCW with a valid S flag. The command field of this CCW is not sent to the I/O device, and the device is signaled that the chain of commands is terminated. A subsequent RESUME I/O (RIO) instruction informs the channel that the suspend CCW may have been modified and that the channel must refetch the CCW and examine the current settings of the flags. If the suspend flag is zero in the CCW, the channel resumes execution of the chain of commands.

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 connected. Channels are provided with different data-transfer capabilities, and an I/O device designed to transfer data only at a specific rate (a magnetic-tape unit or a disk storage, for example) can operate only on a channel that can accommodate at least this data rate.

The data rate a channel can accommodate depends also on the way the I/O operation is programmed. The channel can sustain its highest data rate when no data chaining is specified. Data chaining reduces the maximum allowable rate, and the extent of the reduction depends on the frequency at which new CCWs are fetched and on the address resolution of the first byte in each new storage area. Furthermore, since a channel shares storage with the CPU and other channels, activity in the rest of the system affects the accessibility of storage and, hence, the instantaneous load the channel can sustain.

In view of the dependence of channel capacity on programming and on activity in the rest of the system, an evaluation of the ability of elements in a specific I/O configuration to function concurrently must be based on a consideration of both the data rate and the way the I/O operations are programmed. Two systems differing in performance but 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 10 I/O instructions: CLEAR CHANNEL, CLEAR I/O, HALT DEVICE, HALT I/O, RESUME I/O, START I/O, START I/O FAST RELEASE, STORE CHANNEL ID, TEST CHANNEL, and TEST I/O.

The instructions TEST CHANNEL, CLEAR CHANNEL, and STORE CHANNEL ID address a channel; they do not address an I/O device. The other seven I/O instructions address a channel and a device on that channel.

INPUT/OUTPUT DEVICE ADDRESSING

Within each channel set, an I/O device and the associated access path are designated by an I/O address. The 16-bit I/O address consists of two parts: a channel address in the leftmost eight bit positions and a device address in the rightmost eight bit positions.

The channel address provides for identifying up to 256 channels per channel set. Channels are numbered 0-255. Channel 0 is a byte-multiplexer channel, and each of channels 1-255 may be a byte-multiplexer, block-multiplexer, or selector channel.

The number and type of channels and subchannels available, as well as their address assignment, depend on the system model and the particular installation.

The device address identifies the particular I/O device and control unit on the designated channel. The device address identifies, for example, a particular magnetic-tape drive, disk-access mechanism, or transmission line. Any number in the range 0-255 can be used as a device address, providing facilities for addressing up to 256 devices per channel. An exception is some multiplexer channels that provide fewer than the maximum configuration of subchannels and hence do not permit use of 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 device 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 device address.

Devices sharing a control unit (for example, magnetic-tape drives or disk-access mechanisms) are assigned device 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 a device address in which the number of rightmost zeros is at least equal to the number of bit positions required for specifying the set size. The leftmost bit positions of a device address within such a set identify the control unit, and the rightmost bit positions designate the device on the control unit.

Control units designed to accommodate more than 16 devices may be assigned nonsequential sets of device addresses, each set consisting of 16, or the number required to bring the total number of assigned device addresses equal to the maximum number of devices attachable to the control unit, whichever is smaller. The device-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 device address outside its assigned set or sets. For example, if a control unit is designed to control devices having only the values 0000 to 1001 in the rightmost bit positions of the device address, it does not recognize device addresses containing 1010 to 1111 in these bit positions. On the other hand, a control unit responds to all device addresses in the assigned set for which the corresponding I/O devices are ready, or are not ready but can be made ready by means of an ordinary manual intervention. A control unit may or may not respond to an address within the assigned set when the corresponding device is not installed or has been logically removed from the control unit. If a control unit responds to a device address for which no I/O device is installed or the device has been logically removed from the control unit, the absent device appears in the not-ready state. If no control unit responds to the device address, the I/O device appears not operational.

Input/output devices accessible through more than one channel in a channel set have a distinct I/O address for each path of communications. This I/O address identifies the channel and the control unit. For sets of devices sharing a control unit or connected to two or more control units, the portion of the I/O address identifying the device on the control unit is fixed and does not depend on the path of communications.

The assignment of I/O addresses is arbitrary, subject to the rules described and any model-dependent restrictions. The assignment is made at the time of installation, and the addresses normally remain fixed thereafter.

STATES OF THE INPUT/OUTPUT SYSTEM

The state of the I/O system identified by an I/O address depends on the collective state of the channel, subchannel, and I/O device. Each of these components of the I/O system can have up to four states, as far as the response to an I/O instruction is concerned. These states are listed in the figure "Input/Output-System States." The name of the state is followed by its abbreviation and a brief definition.

A channel, subchannel, or I/O device that is available, interruption-pending, or working is called "operational." A channel, subchannel, or I/O device that is interruption-pending, working, or not-operational is called "not available."

In a multiplexer channel, the channel and subchannel are easily distinguishable and, if the channel is operational, any combination of channel and subchannel states is possible. Since the selector channel can have only one subchannel, the channel and subchannel are functionally coupled, and certain states of the channel are related to those of the subchannel. In particular, the working state can occur only concurrently in both the channel and subchannel and, whenever an interruption condition is pending in the subchannel, the channel also is in the same state. The channel and subchannel, however, are not synonymous, and an interruption condition not associated with data transfer, such as attention, may not affect the state of the subchannel. Thus, the subchannel may as a function of the I/O instruction, be available when the channel is interruption-pending or has an interruption condition pending at a device. A consistent distinction between the subchannel and channel permits selector and multiplexer channels to be covered uniformly by a single description.

Name	Abbreviation and Definition	
Channel		
Available	A	None of the following states
Interruption pending	I	Interruption condition immediately available from channel
Working	W	Channel operating in burst mode
Not operational	N	Channel not operational
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 in device
Working	W	Device executing an operation
Not operational	N	Device not operational

Input/Output-System States

The I/O device referred to in the figure "Input/Output-System States" includes both the I/O device proper and its control unit. For some types of I/O devices, such as magnetic-tape units, the working and the interruption-pending states can be caused by activity in the addressed I/O 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 I/O devices are not related to those of the channel and subchannel.

When the response to an I/O instruction is determined by the state of the channel or subchannel, the components further removed are not interrogated. Thus, 10 composite states may be distinguished as conditions for the execution of I/O instructions. Each composite state is identified by three letters. The first letter specifies the state of the channel, the second letter specifies the state of the subchannel, and the third letter specifies the state of the device. Each letter may be A, I, W, or N, denoting the state of the component. The letter X indicates that the state of the corresponding component is not significant for the execution of the instruction.

Available (AAA): The addressed channel, subchannel, control unit, and I/O device are operational, are not engaged in the execution of any previously initiated operations, and do not contain any pending interruption conditions.

Because of internal activity, some block-multiplexer channels may at times appear to be working even though they are not engaged in the execution of a previously initiated operation and do not contain any interruption condition. This will result in a WXX state instead of the AAA state.

If the addressed device is not installed or has been logically removed from the control unit, but the associated control unit is operational and the address has been assigned to the control unit, the device is said to be not ready. When an instruction is addressed to a device in the not-ready state, the control unit responds to the selection and indicates unit check whenever the not-ready state precludes a successful execution of the operation. When the control unit responds to the selection of a not-ready device, the device is said to be operational and therefore in the available state even though unit check is indicated. (See the section "Unit Check" in this chapter.)

Interruption Pending in Device (AAI) or Device Working (AAW): The addressed channel and subchannel are available. The addressed control unit or I/O device is executing a previously initiated operation or contains an interruption condition. These situations are possible:

1. The device is executing an operation, such as rewinding magnetic tape or seeking on a disk file, after signaling channel end.

2. The control unit associated with the device is executing an operation, such as backspacing file on a

magnetic-tape unit, after signaling channel end.

3. The device or control unit is executing an operation with 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 removed from the channel. The not-operational state is indicated also when the control unit is provided and is designed to attach the device, but the device has not been installed and the address has not been assigned to the control unit. (See also the section "Input/Output Device Addressing" in this chapter.)

Interruption Pending in Subchannel (AIX): The addressed channel is available. An interruption condition is pending in the addressed subchannel. The subchannel is able to provide information for a CSW. The interruption information indicates status associated with the addressed I/O device or another I/O device associated with the subchannel. The state of the addressed device is not significant, except when the address specified by TEST I/O is the same as the address of the I/O device for which the subchannel is interruption-pending, in which case the CSW contains status information that has been 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 START I/O (SIO) or START I/O FAST RELEASE (SIOF) function. The addressed subchannel enters the working state when condition code 0 is set for SIO or SIOF. The addressed subchannel remains in the working state until the SIO or SIOF function is concluded at the subchannel. Usually the conclusion of the SIO or SIOF function occurs when the I/O operation or chain of operations receives 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 the execution of 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.

HALT DEVICE issued to a subchannel that has a pending or suspended I/O operation considers the channel to be busy. In this case, the I/O system appears to be in the channel-working state (WXX) rather than the subchannel-working state (AWX).

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 channel. 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 in which the channel contains an interruption condition is distinguished only by the instruction TEST CHANNEL. This instruction does not cause the subchannel and I/O device to be interrogated. The other I/O instructions, with the exception of STORE CHANNEL ID, consider the channel available when it contains an interruption condition. A channel with an interruption condition may be considered to be working by the instruction STORE CHANNEL ID. When the channel assigns priority for interruptions among devices, the interruption condition is preserved in the I/O device or subchannel. (See the section "Interruption Conditions" in this chapter.)

Channel Working (WXX): The addressed channel is operating in the burst mode. In the multiplexer channel, a burst of bytes is currently being handled. In the selector channel, an operation or a chain of operations is currently being executed, and the channel end for the last operation has not yet been signaled. The states of the addressed device and, in the multiplexer channel, of the subchannel are not significant. In addition, because of internal activity, some block-multiplexer channels may at times appear to be working even though they are not operating in burst mode. Depending on the model and the channel type, TEST I/O, CLEAR I/O, START I/O FAST RELEASE, 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 available in the configuration, when power is off in the channel, when it is not connected to the CPU, or when it detects a channel-check-stop condition. As long as a channel-check-stop condition persists, the channel performs no I/O instructions, with the exception of CLEAR CHANNEL (which may be executed, depending on the system model); performs no I/O interruptions; executes no channel programs; and suspends all I/O-interface activity. When a channel is not operational, 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 types of reset is specified in the SL publication for the device.

I/O-System Reset

I/O-system reset is performed in the channel and on the associated I/O interface when the CPU to which the channel is connected executes the instruction CLEAR CHANNEL or a program reset, initial-program reset, clear reset, or power-on reset is performed, when a power-on sequence is performed by the channel, and, under certain conditions on some models, when a channel detects equipment malfunctions and the recovery-extension facility is not installed.

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

I/O selective reset is performed by some channels when they detect certain equipment malfunctions.

I/O selective reset causes the channel to signal selective reset to the device that is connected to the channel at the time the malfunction is detected. No subchannels are reset.

Effect of Reset on a Working Device

With either type of reset, if the device is currently communicating with a channel, the device immediately disconnects from the channel. Data transfer and any operation using the facilities of the control unit are immediately concluded, and the I/O device is not necessarily positioned at the beginning of a block. Mechanical motion not involving the use of the control unit, such as rewinding magnetic tape or positioning a disk-access mechanism, proceeds to the normal stopping point, if possible. The device appears in the working state until the termination of mechanical motion or the inherent cycle of operation, if any, whereupon it becomes available. Status information in the device and control unit is reset, but an interruption condition may be generated when any mechanical operation is completed.

Reset Upon Malfunction

When a malfunction occurs and the program is alerted by an I/O interruption, or when a malfunction occurs during the execution of an I/O instruction and the program is alerted by the setting of a condition code, then an I/O selective reset may have been performed. A CSW is stored identifying the cause of the malfunction.

The device addressed by the I/O instruction is not necessarily the device that is reset.

When a malfunction occurs and the program is alerted by a machine-check interruption, then an I/O selective reset or, on some models, I/O-system reset may have been performed. This may or may not be accompanied by an I/O interruption.

CONDITION CODE

The results of certain tests by the channel and device, and the original state of the addressed part of the I/O system are used during the execution of an I/O instruction to set one of four condition codes in the PSW. The condition code is set at the time the execution of the instruction is concluded, that is, the time the CPU is released to proceed with the next instruction. The condition code ordinarily indicates whether or not the function specified by the instruction has been performed and, if not, the reason for the rejection. In the case of START I/O FAST RELEASE executed independent of the device, a condition code 0 may be set that is later superseded by a deferred condition code stored in the CSW.

The figure "Condition-Code Settings for I/O States and Functions" lists the I/O-system states and the corresponding condition codes for each I/O function. The I/O-system states and associated abbreviations are defined in the section "States of the Input/Output System" earlier in this chapter. The digits in the figure represent the decimal value of the condition code.

		Condition-Code Settings								
Conditions	I/O State	SIO SIOF	TIO	CLRIO	HIO	HDV	RIO	TCH	STIDC	CLRCH
Available	AAA	0,1*@	0	0	1*	1*	0	0	0	0
Interruption pending in device	AAI	1*@	1*	0	1*	1*	0	0	0	0
Device working	AAW	1*@	1*	0	1*	1*	0	0	0	0
Device not operational	AAN	3@	3	0	3	3	0	0	0	0
Interruption pending in subch.	AIX									
For the addressed device		¤¤	1*#	1*	0	0	0	0	0	0
For another device		2	2	0	0	0	0	0	0	0
Subchannel working	AWX									
With the addressed device		2	2	1*	1*#	1*#	0	0	0	0
With another device		2	2	0	1*#	0	0	0	0	0
Subchannel not operational	ANX	3	3	3	3	3	0	0	0	0
Interruption pending in channel	IXX	@@	@@	@@	@@	@@	0	1	##	0
Channel working	WXX									
With the addressed device		2	2	***	2	+	0	2	##	0&
With another device		2¤	2•	**	2	≠	0	2	##	0&
Internal activity		2¤	2•	**	2	≠	0	2	##	0&
Channel not operational	NXX	3	3	3	3	3	3	3	3	3&&

Explanation:

*   Whenever condition code 1 is set, the CSW or its status portion is stored at real location 64 during execution of the instruction.

**   When CLEAR I/O encounters the WXX state, either condition code 2 is set, or the channel is treated as available and the condition code is set according to the state of the subchannel. When the channel is treated as available, the condition codes for the WXX states are the same as for the AXX states.

***   Condition code 1 (with the CSW stored) or 2 may be set, depending on the channel.

≠   The condition code depends on the state of the subchannel, the channel type, and the system model. If the subchannel is not operational, condition code 2 or 3 is set. If the subchannel is available or working with the addressed device, condition code 2 is set. Otherwise, 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.

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.

+   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.

Condition-Code Settings for I/O States and Functions (Part 1 of 2)

<u>Explanation</u> (Continued):

•	If the subchannel is interruption-pending for the addressed device, condition code 1 may be set depending on the channel type.
&	On certain channels, when the working state precludes performing the I/O-system reset, condition code 2 is set.
&&	On certain channels, when the not-operational state is due to a channel-check-stop condition, the instruction is executed, and condition code 0 is set.
¤	Depending on the facilities provided for START I/O FAST RELEASE, some channels may set condition code 0.
¤¤	If the subchannel is interruption-pending because of the concluding of the portion of the operation involving the use of channel facilities, condition code 2 is set. If the interruption-pending condition exists for other reasons, condition code 1 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.
@@	For the purpose of executing START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT DEVICE, and HALT I/O, a channel containing an interruption condition appears the same as an available channel, and the condition code setting depends on the states of the subchannel and device. The condition codes for the IYY states are the same as for the AYY states, where the Ys 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 Functions (Part 2 of 2)

The channel-available state results in condition code 0 only when no errors are detected during the execution of the I/O instruction.

When a subchannel on a multiplexer channel contains an 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 is detected 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. However, when the nature of the error causes a machine-check interruption, but no I/O interruption, to occur, the CSW is not stored. Three types of errors can occur:

<u>Channel-Equipment Error</u>: 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 channel received an address from the device during initial selection that either had a parity error or was 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 during initial selection had 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, RESUME I/O, and TEST CHANNEL, but RESUME I/O and TEST CHANNEL do not cause a CSW to be stored.)

The channel may perform an I/O selective reset or, on some models, may perform an I/O-system reset or 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.

<u>Channel-Programming Error</u>: The channel can detect the following programming errors during execution of START I/O or START I/O FAST RELEASE. All of the

errors 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 errors are indicated as for START I/O, and the remaining errors may be indicated as for SIO or may be indicated in a subsequent I/O interruption.

Depending on the model, conditions 9, 10, 11 and 12 may (a) cause an error condition to be recognized and prevent operation initiation or (b) may cause an error condition to be recognized only if the operation causes the device to attempt to transfer data. In case (b), a command that specifies an immediate operation does not cause an error indication for an SIO or SIOF function.

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 specifying 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

13. If suspend control was specified, invalid suspend flag in first CCW.

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 as part of the execution of TEST I/O, START I/O, or START I/O FAST RELEASE are indicated by unit check or unit exception in the CSW.

The causes of unit check and unit exception for each type of I/O device are detailed in the SL publication for the device.

INSTRUCTION FORMATS

All I/O instructions use the following S format:

Op Code	B_2	D_2
0	16	20 31

Except for STORE CHANNEL ID, bit positions 8-14 of these instructions are ignored unless the system model provides the suspend-and-resume facility. When the facility is provided, bits 8-14 are ignored, except for RESUME I/O, STORE CHANNEL ID, and the operation codes 9C03 through 9CFF, which are invalid.

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 effective address has the following format:

////////	Chn Addr	Dev Addr
8	16	24 31

Bit positions 16-31 contain the 16-bit I/O address. Bit positions 8-15 are ignored.

INSTRUCTIONS

All I/O instructions cause a serialization and checkpoint-synchronization function to be performed. See the section "Serialization" in Chapter 5, "Program Execution."

The names, mnemonics, and operation codes of the I/O instructions are listed in the figure "Summary of Input/Output Instructions." The figure also indicates that all I/O instructions cause a program interruption when they are encountered in the problem state, that all I/O instructions set the condition code, and that all I/O instructions are in the S instruction format.

Note: In the detailed descriptions of the individual instructions, the mnemonic and the symbolic operand designation for the assembler language are shown with each instruction. In the case of START I/O, for example, SIO is the mnemonic and $D_2(B_2)$ the operand designation.

Name	Mne- monic	Characteristics				Op* Code
CLEAR CHANNEL	CLRCH	S C RE	P	¢		9F01
CLEAR I/O	CLRIO	S C	P	¢		9D01
HALT DEVICE	HDV	S C	P	¢		9E01
HALT I/O	HIO	S C	P	¢		9E00
RESUME I/O	RIO	S C SR	P	¢		9C02
START I/O	SIO	S C	P	¢		9C00
START I/O FAST RELEASE	SIOF	S C	P	¢		9C01
STORE CHANNEL ID	STIDC	S C	P	¢		B203
TEST CHANNEL	TCH	S C	P	¢		9F00
TEST I/O	TIO	S C	P	¢		9D00

Explanation:

¢ Causes serialization and checkpoint synchronization.
* The handling of bits 8-15 of the operation code depends on the instruction and the facilities installed. See the description of the instruction for details.
C Condition code is set.
P Privileged-operation exception.
RE Recovery-extension facility.
S S instruction format.
SR Suspend-and-resume facility.

Summary of Input/Output Instructions

Programming Note

The instructions CLEAR I/O, HALT DEVICE, HALT I/O, START I/O, START I/O FAST RELEASE, STORE CHANNEL ID, and TEST I/O may 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 CLEAR I/O, HALT DEVICE, HALT I/O, START I/O, START I/O FAST RELEASE, STORE CHANNEL ID, or TEST I/O is issued and must remain disabled until the information in the CSW provided by any of these instructions has been acted upon or stored elsewhere for later use.

CLEAR CHANNEL

CLRCH $D_2(B_2)$ [S]

'9F01'	B_2	D_2
0	16	20 31

With the recovery-extension facility installed, the CLRCH function is performed. Otherwise, the TCH function, which is described in the definition of TEST CHANNEL, is performed.

I/O-system reset is performed in the addressed channel, with system reset signaled to all I/O devices attached to the addressed channel.

Bits 8-14 of the instruction are ignored. Bits 16-23 of the second-operand address identify the channel to which the instruction applies. Bits 24-31 of the address are ignored.

The CLRCH function inspects only the state of the addressed channel. When the channel is available or interruption-pending, I/O-system reset is performed.

When the channel is working, some channels may indicate busy and cause no I/O-interface action, while other channels cause I/O-system reset to be performed.

When the channel is not operational because of a channel-check-stop condition, some channels cause an I/O-system reset to be performed on the I/O interface. In all other not-operational-state cases, the reset function is inhibited.

Program Exceptions:

Privileged operation

Resulting Condition Code:

0 I/O-system reset was performed on the I/O interface associated with the addressed channel
1 --
2 Channel busy

3 Not operational

The condition code set when CLEAR CHANNEL causes the CLRCH function to be performed is shown for all possible states of the I/O system in the figure "Condition Codes Set by CLEAR CHANNEL." The condition code set when CLEAR CHANNEL causes the TCH function to be performed is shown for all possible states of the I/O system in the figure "Condition Codes Set by TEST CHANNEL" in the definition of the instruction TEST CHANNEL. See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

	A	I	W	N
Channel	0	0	0+	3++

A	Available
I	Interruption Pending
W	Working
N	Not Operational

+ On certain channels, when the working state precludes performing the I/O-system reset on the I/O interface, condition code 2 is set.

++ On certain channels, when the not-operational state is due to a channel-check-stop condition, the instruction is executed, and condition code 0 is set.

Condition Codes Set by CLEAR CHANNEL

Programming Note

CLEAR CHANNEL should be used to reset an I/O-device association with an I/O interface when I/O devices are shared with other systems or have multiple paths to the same system. In those cases when I/O devices are shared, before using CLEAR CHANNEL, steps should be taken to protect against compromising data integrity until the desired I/O-device association can be reestablished.

CLEAR CHANNEL may cause a channel that is not operational because of a channel-check-stop condition to be restored. Before a not-operational channel can be restored or system reset signaled on an I/O interface, on some models CLEAR CHANNEL must be issued to all channels. On other models, CLEAR CHANNEL, when issued to a subset of the channels, can cause a not-operational channel to be restored and system reset to be signaled on an I/O interface. Refer to the SL publication for the model to determine the appropriate recovery action.

CLEAR I/O

CLRIO $D_2(B_2)$ [S]

'9D01'	B_2	D_2
0	16	20 31

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.

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.

Either a TIO or CLRIO function is performed, depending on the channel and the block-multiplexing-control bit, bit 0 of control register 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 TEST I/O instruction.

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 set condition code 2.

When the subchannel is either working with the addressed device or interruption-pending with the addressed device, the CLRIO function causes condition code 1 to be set and causes the channel to discontinue the operation with the addressed device by storing the status of the operation in the CSW and making the subchannel available. When the channel is working with the addressed device, the device is signaled to terminate the current operation. Some channels may, instead, indicate busy and cause no channel action.

When any of the following conditions occurs, the CLRIO function causes the CSW to be stored at real storage locations 64-71. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

1. The channel is available or interruption-pending, and the subchannel (1) contains an interruption condition for the addressed device because of the ending of an I/O operation at the subchannel or (2) is working with the addressed device. The subchannel-key,

command-address, and count fields describe the state of the operation at the time of the execution of the instruction. If the subchannel is interruption-pending for reasons other than the completion of an I/O operation at the subchannel, the fields in the CSW other than the unit-status field are all set to zeros. If the operation has not yet been initiated at the device, the deferred condition code is 1.

2. The channel is working with the addressed device. The subchannel-key, command-address, and count fields describe the state of the operation at the time the instruction is executed. (Some channels alternatively indicate busy under this condition.)

3. The channel is working with a device other than the one addressed, and the subchannel (1) contains a pending interruption condition for the addressed device because of the ending of an I/O operation at the subchannel or (2) is working with the addressed device.

   In the former case, the subchannel-key, command-address, and count fields describe the state of the operation at the time CLEAR I/O is executed. If the operation has not yet been initiated at the device, the deferred condition code is 1.

   In the latter case, if the subchannel is interruption-pending for reasons other than the completion of an I/O operation at the subchannel, the fields in the CSW other than the unit-status field are all set to zeros.

   Some channels alternatively indicate busy under the above conditions (channel working).

4. The channel recognizes 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 the CLRIO function cannot be executed because of a pending logout that affects the operational capability of the channel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending and channel-control-check bits, which are set to ones. No channel logout is associated with this status.

<u>Program</u> <u>Exceptions</u>:

Privileged operation

<u>Resulting</u> <u>Condition</u> <u>Code</u>:

0 No operation in progress at the subchannel for the addressed device
1 CSW stored
2 Channel busy
3 Not operational

The condition code set when CLEAR I/O causes the CLRIO function to be performed is shown for all possible states of the I/O system in the figure "Condition Codes Set by CLEAR I/O." The condition code set when CLEAR I/O causes the TIO function to be performed is shown for all possible state of the I/O system in the figure "Condition Codes Set by TEST I/O" in the definition of the TEST I/O instruction. See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

Channel	A						I						W≠						W#	N
Subchannel	A	I≠	I#	W≠	W#	N	A	I≠	I#	W≠	W#	N	A	I≠	I#	W≠	W#	N	++	3
	0	$\underline{0}$	$\underline{1}$*	$\underline{0}$	$\underline{1}$*	$\underline{3}$	0	0	1*	$\underline{0}$	$\underline{1}$*	$\underline{3}$	+	+	++	+	++	+++		

A   Available
I   Interruption pending
    I≠ = Interruption pending for a device other than the one addressed
    I# = Interruption pending for the addressed device
W   Working
    W≠ = Working with a device other than the one addressed
    W# = Working with the addressed device
N   Not operational
*   CSW stored

+   In the W≠AX, W≠I≠X, and W≠W≠X states, a condition code 0 or 2 may be set, depending on the channel.

++ In the W≠I#X, W≠W#X, and W#XX states, a condition code 1 (with the CSW stored) or 2 may be set, depending on the channel.

+++ In the W≠NX state, a condition code 2 or 3 may be set, depending on the channel.

Note: Underscored codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by CLEAR I/O

Programming Notes

1. Since some channels cause condition code 2 to be set when the instruction is received and the channel is working, it may be useful to issue a halt instruction and then CLEAR I/O to the desired address. Using HALT DEVICE will ensure that condition code 2 is received on the CLEAR I/O only when the channel is working with a device other than the one addressed. Using HALT I/O will ensure that the current working state, if any, is terminated without regard for the address.

2. Because of the inability of CLEAR I/O to terminate operations on some channels when in the working state, the instruction is not a suitable substitute for HALT I/O or HALT DEVICE.

3. The combination of HALT DEVICE followed by CLEAR I/O can be used to clear out all activity on a channel by executing the two instructions for all device addresses on the channel.

4. The subchannel is said to be working with a device from the time condition code 0 is set for SIO or SIOF addressed to the device until the subchannel becomes interruption-pending because of the ending at the subchannel of the I/O operation or chain of operations. Suspension of the channel-program execution does not cause the ending at the subchannel of an I/O operation or chain of operations. Therefore, the subchannel is said to be working even while the channel-program execution is suspended.

HALT DEVICE

HDV $D_2(B_2)$ [S]

'9E01'	B_2	D_2
0	16	20 31

The current I/O operation at the addressed I/O device is terminated. The subsequent state of the subchannel depends on the type of channel.

Bits 8-14 of the instruction are ignored. Bits 16-31 of the second-operand address identify the channel, the subchannel, and the I/O device to which the instruction applies.

Either a HALT DEVICE (HDV) or a HALT I/O (HIO) function is performed, depending on the channel. The HIO function is performed when the HDV function is not implemented by the channel.

The HIO function is described in the definition of the HALT I/O instruction. The HDV function is described below.

If the subchannel is in the working state and an I/O operation is pending or suspended at the subchannel for the addressed device, the channel appears busy and condition code 2 is set. Subsequently, when conditions allow, the device is selected and issued the halt signal.

If condition code 2 is set for HALT DEVICE as described above and the I/O operation has not been initiated at the device by the time the halt signal is issued, the I/O operation is terminated and an interruption condition is recognized. The unit-status field of CSW stored when the interruption condition is cleared contains either the last status received from the device when the channel attempted to initiate the pending operation at the device, or zeros if the channel has not attempted to initiate the operation. The command-address field contains the address of the first or suspended CCW plus 8, and the deferred condition code is 1.

If condition code 2 is set when HALT DEVICE is executed as described above but the pending or suspended operation is terminated by the device before the halt signal is issued, the channel recognizes an interruption condition because of the termination. Deferred condition code 1 is indicated in the CSW stored when the interruption condition is cleared. The halt signal may or may not be issued in this case.

If condition code 2 is set when HALT DEVICE is executed as described above and the pending I/O operation has been initiated at the device by the time the halt signal is issued, the subchannel remains working with the device and termination of the operation occurs as a function of status received from the device.

If condition code 2 is set when HALT DEVICE is executed as described above and the I/O operation has not been initiated and the device is detected to be not operational either prior to or during the attempt to issue the halt signal, the I/O operation is terminated, and an interruption condition is recognized. Deferred condition code 3 is indicated in the CSW stored when the interruption condition is cleared, and the unit-status field contains zeros.

If condition code 2 is set when HALT DEVICE is executed as described above and the channel accepts status from the device before the pending I/O operation is initiated at the device and before the halt signal is issued, the operation is terminated, and an interruption condition is recognized. Deferred condition code 1 is indicated in the CSW stored when the interruption condition is cleared. The status that caused the channel to terminate the operation is indicated in the unit-status field of the CSW, with the busy bit included. The halt signal may or may not be issued in this case.

When the channel is either available or interruption-pending, with the subchannel available or working with an I/O operation in progress at 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 available, the subchannel is not affected. If, on a byte-multiplexer channel, the subchannel is working with an I/O operation in progress at the addressed device, data transfer is immediately terminated, but the subchannel remains in the working state until the device with which it is working provides the next status byte, whereupon the subchannel is placed in the interruption-pending state.

When the channel is either available or interruption-pending with the subchannel either working with a device other than the one addressed or interruption-pending, no action is taken.

When the channel is working in burst mode with the addressed device, data transfer for the operation is immediately terminated, and the device immediately disconnects from the channel. If command chaining or command retry is in progress for the I/O operation using the subchannel, it is suppressed.

When the channel is working in burst mode with a device other than the one addressed, and the subchannel is available, interruption-pending, or working with a device other than the one addressed, no action is taken. If the subchannel is working with an I/O operation in progress at 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. If command chaining or command retry is indicated for the I/O operation using the subchannel, it is suppressed.

When the channel is working in burst mode with a device other than the one addressed and the subchannel is not operational, is interruption-pending, or is working with a device other than the one addressed, the resulting condition code may, in some channels, be determined by the subchannel state.

Termination of a burst operation by HALT DEVICE on a selector channel causes the channel and subchannel to be placed in the interruption-pending state. Generation of the interruption condition is not contingent on the receipt of status information from the device. When HALT DEVICE causes a burst operation on a byte-multiplexer channel to be terminated, the subchannel associated with the burst operation remains in the working state until the device next provides 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 design of the subchannel, take place as for a selector channel or may allow the subchannel to remain in the working state until the device next provides status.

On the byte-multiplexer channel operating in the byte-multiplex mode, the I/O 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 signal to terminate the operation because it is busy or in the not-operational 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 or command retry is indicated for the I/O operation using the subchannel, it is suppressed.

When either of the two situations numbered below occurs, HALT DEVICE causes the 16-bit unit-status and channel-status portion of the CSW to be replaced by a new set of status bits. The contents of the other fields of the CSW are not changed. The CSW stored pertains only to the execution of HALT DEVICE and does not describe the I/O operation, at the addressed subchannel, that is terminated. The extent of data transfer and the status at the termination of the operation at the subchannel are provided in the CSW associated with the interruption condition caused by the termination. The two situations are:

1. The addressed device is selected and signaled to halt 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 I/O operation. The CSW unit-status field contains ones in the busy and status-modifier bit positions. The channel-status field contains zeros unless a machine malfunction is detected.

When a channel recognizes an equipment malfunction during the execution of HALT DEVICE, a CSW may or may not be immediately stored, depending on the state of the subchannel or the channel model. When the subchannel is interruption-pending and a malfunction occurs during the execution of HALT DEVICE, condition code 0 may be set, and the subsequently stored CSW may or may not indicate the malfunction, depending on whether or not the malfunction affected the I/O operation. When the channel recognizes a malfunction and the subchannel is working with the addressed device, condition code 0 or 1 may be set, depending on the channel model. If the channel sets condition code 1, the contents of the immediately stored CSW identify the type of malfunction. If the channel sets condition code 0, the contents of the subsequently stored CSW identify the type of malfunction. In either case, the state of the channel and the progress of the I/O operation are unpredictable. Refer to the SL publication for the system model to determine its particular implementation.

When HALT DEVICE cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

When HALT DEVICE causes data transfer to be terminated, the subchannel associated with the operation either (1) remains in the working state until the channel-end condition is received and the subchannel enters the interruption-pending state or (2) immediately enters the interruption-pending state, depending on the type of channel. If the subchannel is shared by other devices attached to the control unit, I/O instructions addressed to those devices set the condition code appropriate to the subchannel states described.

When HALT DEVICE causes data transfer to be terminated, the control unit associated with the operation may not become available until the data-handling portion of the operation in the control unit is concluded. Conclusion of this portion of the operation is signaled by the generation of channel end. This may occur at the normal time for the operation, or earlier, or later, depending on the operation and type of device.

When HALT DEVICE causes data transfer to be terminated, 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 signal depends partially on the type of device and its state. In all cases, the halt signal has no effect on devices that are not in the working state or are executing a mechanical operation in which data is not transferred, such as rewinding tape or positioning a disk-access mechanism. If the device is executing a type of operation that is unpredictable in duration, or in which data is transferred, the device interprets the signal as one to terminate the operation. Pending interruption 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 when HALT DEVICE causes the HDV function to be performed is shown for all possible states of the I/O system in the figure "Condition Codes Set by HALT DEVICE." The condition code set when HALT DEVICE causes the HIO function to be performed is shown for all possible states of the I/O system in the figure "Condition Codes Set by HALT I/O" in the description of the HALT I/O instruction. See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

Channel	A											I											W≠					W#	N
Subchannel	A				I	W≠	W#				N	A				I	W≠	W#				N	A	I	W≠	W#	N		
					$\underline{0}$	$\underline{0}$					$\underline{3}$					0	$\underline{0}$					$\underline{3}$	$\underline{2}$	$\underline{+}$	+	$\underline{2}$	$\underline{\bullet}$	@	3
CU/ Device	A	I	W	N			A	I	W	N		A	I	W	N			A	I	W	N								
	1*	1*	1*	3			$\underline{1\$}$	$\underline{1\$}$	$\underline{1\$}$	$\underline{3}$		1*	1*	1*	3			$\underline{1\$}$	$\underline{1\$}$	$\underline{1\$}$	$\underline{3}$								

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

$ CSW stored. Condition code 0 (with no CSW stored) instead of condition code 1 may be set when a malfunction is detected.

@ In the W#XX state, either condition code 1 (with CSW stored) or condition code 2 may be set, depending on the channel. However, condition code 1 (with CSW stored) can be set only if the control unit has received the signal to terminate.

+ In the W≠IX and W≠W≠X states, either condition code 0 or 2 may be set.

• In the W≠NX state, either condition code 2 or 3 may be set, depending on the model and the channel type.

Note: Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by HALT DEVICE

Programming Notes

1. A program can ensure complete compatibility between HALT DEVICE and HALT I/O on channels that execute HALT DEVICE as HALT I/O by observing the following conventions:

   a. On a byte-multiplexer channel, do not issue HALT DEVICE to a multiplexing device when a burst operation could be in progress on the channel.

   b. 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.

   c. 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.

2. A block-multiplexer channel may execute HALT DEVICE as a block-multiplexer or selector channel. However, when a block-multiplexer channel is operating with multiplexing inhibited, HALT DEVICE causes the HDV function to be performed rather than the HIO function.

3. The execution of HALT DEVICE always causes data transfer between the addressed device and the channel to be terminated. The condition code and the CSW (when stored) indicate whether the control unit was signaled to terminate its operation during the execution of the instruction. If the control unit was not signaled to terminate its operation, the condition code and the CSW (when stored) imply the situations under which the execution of a HALT DEVICE for the same address will cause the control unit to be signaled to terminate.

   Condition code 0 indicates that HALT DEVICE cannot signal the control unit until an interruption condition on the same subchannel is cleared.

   Condition code 1 with control-unit-busy status in the CSW indicates that HALT DEVICE cannot signal the control unit until the control-unit-end status is received from that control unit.

   Condition code 1 with zeros in the status field of the CSW indicates that the addressed device was selected and signaled to terminate the current operation, if any.

   Condition code 2 indicates that the control unit cannot be signaled until the channel is not working. The end of the working state can be detected by noting an interruption from the channel or by noting the results of repeatedly executing HALT DEVICE.

   Condition code 3 indicates that manual intervention is required in order to allow HALT DEVICE to signal the control unit to terminate.

HALT I/O

HIO $D_2(B_2)$ [S]

```
 _________________________________________
|      '9E00'       | B2 |       D2       |
|___________________|____|________________|
0                   16   20              31
```

Execution of the current I/O operation at the addressed I/O device, subchannel, or channel is terminated. The subsequent state of the subchannel depends on the type of channel.

Bits 8-14 of the instruction are ignored. Bits 16-31 of the second-operand address identify the channel and, when the channel is not working, identify the subchannel and the I/O device to which the instruction applies.

The HIO function is performed by the HALT I/O instruction and, on some channels and under certain circumstances, by HALT DEVICE.

When the channel is either available or interruption-pending, with the subchannel either available or working, HALT I/O causes the addressed I/O 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 the channel is either available or interruption-pending with the subchannel working but the I/O operation is either not yet initiated at the device or is suspended, HALT I/O causes the suspended or pending I/O operation to be terminated and an interruption condition to be recognized. The CSW stored when the interruption occurs contains zeros in the unit-status and channel-status fields. The command-address field contains the address of the first or the suspended CCW, plus 8, and the deferred condition code is 1.

When HALT I/O is issued to a channel operating in the burst mode, data transfer for the burst operation is terminated, and the I/O device performing the burst operation is immediately disconnected from the channel. The subchannel and I/O-device address in the instruction is ignored in this case.

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 I/O 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 I/O device next provides status, 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 design of the subchannel, take place as for a selector channel or may allow the subchannel to remain in the working state until the device next provides status.

On the byte-multiplexer channel operating in the byte-multiplex mode, the I/O 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 signal to halt the operation because it is in the not-operational or busy state, the subchannel, if working with a device, is set up to signal termination of device operation the next time the device requests or offers a byte of data. If command chaining or command retry is indicated in the subchannel, it is suppressed if the device presents status.

When the addressed subchannel is interruption-pending, with the channel available or interruption-pending, HALT I/O does not cause any action.

When any of the following conditions occurs, HALT I/O causes the status portion, bits 32-47, of the CSW to be replaced by a new set of status bits. The contents of the other fields of the CSW are not changed. The CSW stored by HALT I/O pertains only to the execution of HALT I/O and does not describe the I/O operation that is terminated at the addressed subchannel. The extent of data transfer, and the status at the termination of the operation at the subchannel, are provided in the CSW associated with the interruption condition due to the termination.

1. The addressed device was selected and signaled to halt 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 signal because it was executing a previously initiated operation or had an interruption condition associated with a device other than the one addressed. The signal to terminate the operation has not been transmitted to the device, and the subchannel, if in the working state with an I/O operation in progress at the device, will signal termination the next time the device identifies itself. The CSW unit-status field contains ones in the busy and status-modifier bit positions. The channel-status field contains zeros unless an equipment error is detected.

When a channel detects an equipment malfunction during the execution of HALT I/O, a CSW may or may not be immediately stored, depending on the state of the subchannel or the channel model. When the subchannel is interruption-pending and a malfunction occurs during the execution of HALT I/O, condition code 0 is set, and the channel-status field of the subsequently stored CSW may or may not indicate channel-control check, along with the other ending-status information, depending on whether the malfunction affected the I/O operation. When the channel recognizes a malfunction during the execution of HALT I/O and the subchannel is working, condition code 0 or 1 may be set, depending on the channel model. If the channel sets condition code 1, the contents of the immediately stored CSW identify the malfunction. If the channel sets condition code 0, the contents of the subsequently stored CSW identify the malfunction and may also indicate other status information describing the terminated operation. Consult the SL publication for each system model to determine implementation.

When HALT I/O cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

When HALT I/O causes data transfer to be terminated, the control unit associated with the operation may not become available 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.

When HALT I/O causes data transfer to be terminated, the subchannel associated with the operation either (1) remains in the working state until the channel-end condition is received and the subchannel enters the interruption-pending state or (2) immediately enters the interruption-pending state, depending on the type of channel. If the subchannel is shared by other devices attached to the control unit, I/O instructions addressed to those devices set the condition code appropriate to the subchannel states described.

When HALT I/O causes data transfer to be terminated, 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 I/O 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 signal depends on the type of I/O device and its state and is specified in the SL publication for the I/O device. The halt signal has no effect on I/O devices that are not in the working state or are executing a mechanical operation in which data is not transferred, such as rewinding tape or positioning a disk-access mechanism. If the I/O device is executing a type of operation that is variable in duration, the I/O device interprets the signal as one to terminate the operation. Attention or device-end signals at the device are not reset.

Program Exceptions:

Privileged operation

Resulting Condition Code:

0	Interruption pending in subchannel
1	CSW stored
2	Burst operation terminated
3	Not operational

The condition code set by HALT I/O for all possible states of the I/O system is shown in the figure "Condition Codes Set by HALT I/O." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

<table>
<tr><td>Channel</td><td colspan="7">A</td><td colspan="7">I</td><td>W</td><td>N</td></tr>
<tr><td>Subchannel</td><td colspan="4">A</td><td>I</td><td>W</td><td>N</td><td colspan="4">A</td><td>I</td><td>W</td><td>N</td><td>2</td><td>3</td></tr>
<tr><td></td><td colspan="4"></td><td>$\underline{0}$</td><td>$\underline{1}$*#</td><td>$\underline{3}$</td><td colspan="4"></td><td>0</td><td>$\underline{1}$*#</td><td>$\underline{3}$</td><td></td><td></td></tr>
<tr><td>CU/</td><td>A</td><td>I</td><td>W</td><td>N</td><td></td><td></td><td></td><td>A</td><td>I</td><td>W</td><td>N</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Device</td><td>1*</td><td>1*</td><td>1*</td><td>3</td><td></td><td></td><td></td><td>1*</td><td>1*</td><td>1*</td><td>3</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

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. Condition code 0 may be set if a malfunction is detected.

Note: Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by HALT I/O

Programming Note

The instruction HALT I/O provides the program with a means of terminating an I/O operation before all data specified in the operation has been transferred or before the operation at the device has reached its normal ending point. It permits the program to immediately free the selector channel for an operation of higher priority. On the byte-multiplexer channel, HALT I/O provides a means of controlling real-time operations and permits the program to terminate data transmission on a communication line.

RESUME I/O

RIO $D_2(B_2)$ [S]

'9C02'	B_2	D_2
0	16	20 31

Depending on whether the suspend-and-resume facility is provided by the system model, an RIO or SIO function is performed. The RIO function is performed when the suspend-and-resume facility is provided by the system model; otherwise, the SIO function is performed.

The SIO function is described in the definition of the instruction START I/O.

Execution of the RIO function causes a currently suspended channel-program execution to be resumed with the device if the suspend flag in the CCW causing suspension has been set to zero. If the suspend flag remains set to one or if the I/O operation is not currently suspended, the instruction has no effect on the channel-program execution. The instruction is executed only when the CPU is in the supervisor state; otherwise, a privileged-operation exception is recognized, and the operation is suppressed.

Bits 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies.

The RIO function is performed independent of the state of the channel, subchannel, and device so long as the channel is operational.

If the channel is not operational, condition code 3 is set, and no action takes place. If the channel is operational, condition code 0 is set. If the suspend-and-resume facility is not provided for the addressed subchannel or if the addressed subchannel is not operational, no further action takes place. If the suspend-and-resume facility is provided for the addressed subchannel and a channel-program execution is currently suspended or in the process of being suspended at the subchannel, the channel is signaled to perform the resume function. A channel-program execution is in the process of being suspended if a channel-command word (CCW) has been fetched which contains a valid S flag but the suspend function has not yet been completed.

The RIO function is performed by the channel logically subsequent to and asynchronous to the execution of the RESUME I/O that provided the stimulus. The RIO function causes the channel to perform a modified SIOF function by using the CCW that previously caused the channel to perform the suspend function as the first CCW of the resumed channel-program execution. Resumption of channel-program execution appears to the device to be the initiation of a new I/O operation not chained to the previous operation.

Program Exceptions:

Privileged operation

Resulting Condition Code:

0 RIO function performed
1 --
2 --
3 Channel not operational

The condition code set by RESUME I/O for all possible states of the I/O system is shown in the figure "Condition Codes Set by RESUME I/O." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

	A	I	W	N
Channel	0	0	0	3

A Available
I Interruption pending
W Working
N Not operational

Condition Codes Set by RESUME I/O

Programming Notes

1. Programs designed to be executed in models that do not provide the suspend-and-resume facility may not be executed properly in models that provide the facility because bits 8-14 of the RESUME I/O operation

code (9C02) are defined to be significant.

2. Programs that issue RESUME I/O are not executed correctly in models that do not provide the suspend-and-resume facility. In these models, RESUME I/O is executed as START I/O. This means that programs that use RESUME I/O must be designed to issue RESUME I/O only in models that provide the suspend-and-resume facility. The program can determine whether the suspend-and-resume facility is provided by issuing a mock START I/O with the suspend-control bit set to one in the CAW. If the mock I/O operation is not terminated with a channel-program-check indication because CAW bit 4 is not equal to zero, the suspend-and-resume facility is provided, and the program may safely issue RESUME I/O.

3. Unlike a channel program initiated by START I/O or START I/O FAST RELEASE, a suspended channel program being resumed may specify a CCW containing the transfer-in-channel command as the first CCW executed when channel-program execution is resumed.

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.

Bits 8-14 of the instruction are ignored unless the suspend-and-resume facility is provided by the system model. When the facility is provided, bits 0-15 of the instruction are interpreted as follows:

Operation Code	Interpretation
9C00	START I/O
9C01	START I/O FAST RELEASE
9C02	RESUME I/O
9C03-9CFF	Invalid operation

Bits 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies. The CAW, at real location 72, contains the subchannel key, the suspend-control bit, and the address of the first CCW. This CCW specifies the operation to be performed and the storage area to be used, if any.

Either an SIO or SIOF function is performed, depending on the instruction, the channel, and the block-multiplexing-control bit, bit 0 of control register 0. The instruction START I/O always causes the SIO function to be performed, as does START I/O FAST RELEASE when the block-multiplexing-control bit is zero. When the bit is one, START I/O FAST RELEASE may, depending on the channel, cause either the SIO or the SIOF function to be performed.

For the SIO function, the I/O operation is initiated at the device if the suspend flag is not 1 in the first CCW, the addressed I/O device and subchannel are available, the channel is available or interruption-pending, and errors or exceptional situations have not been detected. The I/O operation is not initiated when the addressed part of the I/O system is in any other state or when the channel or device detects any error or exceptional situations during execution of the instruction.

For the SIOF function, the I/O operation is made pending at the subchannel if the subchannel is available, the channel is available or interruption-pending, and no errors or exceptional conditions are recognized during the execution of START I/O FAST RELEASE. Selection of the I/O device may be performed during the execution of the instruction or may be performed later. When an SIOF function is performed, initiation of the I/O operation at the I/O device occurs logically subsequent and asynchronous to the execution of the instruction. When the I/O operation is not initiated at the I/O device during the execution of the instruction, the I/O operation is said to be pending at the subchannel until channel and subchannel facilities are available for initiation. When an I/O operation is made pending at the subchannel, the subchannel enters the working state and condition code 0 is set for the instruction.

Status, other than control-unit end or device end signaling the end of a previously signaled control-unit-busy or device-busy condition, that is presented by the device while the I/O operation is pending at the subchannel causes the pending I/O operation to be canceled. An interruption condition is recognized, and the status, with the busy bit appended, is stored in the unit-status field when the CSW is stored that clears the interruption condition. The deferred condition code is stored as 1 in the CSW in this case, and the CCW-address field contains the address of the first CCW plus 8.

When the channel attempts to initiate the pending I/O operation at the I/O device, the detection of any error condition by the channel or the I/O device causes the channel to terminate the operation. The detection of any exceptional condition by the channel or the I/O device during the attempt to initiate the I/O operation at the I/O device also causes the channel to terminate the operation, except for certain busy conditions when start-I/O-fast queuing is provided for the subchannel.

When start-I/O-fast queuing is provided, busy conditions detected during the selection of the I/O device cause the currently pending I/O operation at the subchannel not to be initiated. Whether the I/O operation remains pending or is terminated when these conditions are detected depends on the degree of start-I/O-fast queuing provided and conditions existing at the channel and subchannel when detected.

Control-unit or device-busy conditions detected during the attempt to initiate a pending I/O operation at the device may cause the operation to remain pending.

If conditions are such that the I/O operation is not terminated but remains pending at the subchannel, the operation will remain pending until terminated for some other reason or until the no-longer-busy indication is received from the control unit or device. When the latter occurs, the channel again attempts to initiate the pending I/O operation at the device.

When the channel is available or interruption-pending, when the subchannel is available before the execution of SIO or SIOF, and when the suspend-and-resume facility is provided and the first CCW contains a valid suspend (S) flag, condition code 0 is set, but the command in the first CCW is not transferred to the device. Instead, the subchannel enters the subchannel-working state with channel-program execution suspended. When the SIOF function is performed in this case, detection of a valid S flag in the first CCW may occur either before or after condition code 0 is set, depending on the system model.

When the channel is available or interruption-pending, and the subchannel is available before the execution of the instruction, the following situations cause a CSW to be stored. How the CSW is stored depends on whether an SIO or SIOF function is performed. The SIO function causes the status portion of the CSW to be replaced by a new set of status bits. The status bits pertain to the device addressed by the instruction. The contents of the other fields of the CSW are not changed. When the SIOF function is performed, situation 1 causes the same action as for the SIO function; also, the control-unit and device state may be tested, with the result that situation 5 may cause the same action as for the SIO function. Or, situation 5 may be indicated in a subsequent I/O interruption during which the entire CSW is stored, or, when start-I/O-fast queuing is provided, situation 5 may not be indicated at all. The remaining situations for the SIOF function are indicated in a subsequent I/O interruption, during which the entire CSW is 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. If selection of the device occurred prior to detection of the error or if the error condition was detected during the selection of the device, the device status is indicated in the CSW.

2. The channel detects a programming error associated with the first CCW or, if channel indirect data addressing 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. If selection of the device occurred prior to detection of the error, or if the error condition was detected during the selection of the device, the device status is indicated in the CSW.

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 situations detected during the operation. In the CSW, the channel-end bit is one, the busy bit is zero, and other status may be indicated. The I/O operation is initiated, but no information has been transferred to or from the storage area designated by the CCW. No interruption conditions are generated at the subchannel, and the subchannel is available for a new I/O operation.

If device end is not indicated, the device remains busy, and a subsequent device-end condition is generated by the device.

4. The I/O device is interruption-pending, or the control unit is interruption-pending for the addressed device. The CSW unit-status field contains one in the busy-bit position, identifies the interruption condition, and may contain other bits provided by the device or control unit. The interruption condition is cleared. The I/O operation is not initiated. The channel-status field indicates any errors detected by the channel.

5. The I/O device or the control unit is executing a previously initiated operation, or the control unit is interruption-pending for a device other than the one addressed. The CSW unit-status field contains one in the busy-bit position or, if the control unit is busy, the busy and status-modifier bits are ones. The I/O operation is not initiated. The channel-status field indicates any errors detected by the channel, and the PCI bit is one if specified in the first CCW.

6. The I/O device or control unit detected an equipment or programming error during the initiation, or the addressed device is not ready. The CSW identifies the error. The channel-end and busy bits are zeros, unless the device was busy, in which case the busy bit, as well as any bits causing interruption conditions, are ones. The interruption conditions indicated in the CSW have been cleared at the device. The I/O operation is not initiated. No interruption conditions are generated at the I/O device or subchannel.

When the SIO or SIOF function cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout occurs in this case.

Certain situations encountered during the execution of SIO cause condition code 1 to be set. When SIOF is executed, these same situations may be encountered after condition code 0 is set. When the latter occurs, a deferred-condition-code-1 I/O-interruption condition is generated to report these situations to the program. An exception to this may occur when start-I/O-fast queuing is provided for the subchannel. With start-I/O-fast queuing, control-unit-busy or device-busy conditions encountered while attempting to initiate the I/O operation may be handled by the channel instead of a deferred-condition-code-1 I/O interruption generated.

When the SIOF function causes condition code 0 to be set and, subsequently, it is determined that the device is not operational, a deferred-condition-code-3 I/O-interruption condition is generated. In both of the above cases, in the resulting I/O interruption, a full CSW is stored, and the deferred condition code appears in the CSW.

When start-I/O-fast queuing is provided, I/O operations may remain pending at the subchannel while the control unit or device is busy. The control unit or device signals the end of the busy period by presenting a status byte containing control-unit end or device end, respectively.

When device-end status signals the end of a previously signaled device-busy period, and an I/O operation is pending at the subchannel for the device, the channel attempts to initiate the pending operation without causing an I/O interruption. When the status is control-unit end, and one or more devices attached to the control unit have I/O operations pending, the channel attempts to initiate one of the pending operations.

If a control unit presents a status byte and the channel is unable to accept that status byte because of an I/O operation that is pending at the associated subchannel for a different device to which a busy indication had previously been presented, then the I/O operation that is queued at the subchannel is terminated and the subchannel becomes interruption-pending. When the associated interruption occurs, the CSW that is stored contains the busy indication in the unit-status byte, and the deferred condition code is 1.

If the busy indication received by the channel when the device or control unit was interrogated while busy was not presented to the program, the no-longer-busy indication is not presented to the program. If the device-busy indication was presented to the program and no I/O operation is pending for that device when the device-end indication is received, an interruption condition is recognized, and the device-end indication is presented to the program.

If the control-unit busy indication was presented to the program, receipt of the corresponding control-unit-end (CUE) indication causes the channel to recognize an interruption condition. If the subchannel corresponding to the unit address with which the CUE indication is associated is available, the subchannel is made interruption-pending. The CUE status is stored in the unit-status field of the CSW stored when the interruption condition is cleared. If the subchannel corresponding to the unit address with which the CUE indication is associated is working, that is, the subchannel contains a pending I/O operation or a suspended channel-program execution, the channel generates the channel-available-interruption (CAI) condition instead, and the control-unit-end status is not made available to the program. When the CAI condition replaces the CUE condition, the state of the associated subchannel is not affected. (See the section, "Channel-Available Interruption," for the detailed description of CAI.)

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	SIO or SIOF function has been accepted
1	CSW stored
2	Channel or subchannel busy
3	Not operational

The condition code set by START I/O and START I/O FAST RELEASE for all possible states of the I/O system is shown in the figure "Condition Codes Set by START I/O and START I/O FAST RELEASE." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

<table>
<tr><td>Channel</td><td colspan="9">A or I</td><td>W</td><td>N</td></tr>
<tr><td></td><td colspan="9"></td><td>2+</td><td>3</td></tr>
<tr><td>Subchannel</td><td colspan="5">A</td><td>I≠</td><td>I#</td><td>W</td><td>N</td><td></td><td></td></tr>
<tr><td></td><td colspan="5"></td><td>2$</td><td>2$&</td><td><u>2</u></td><td><u>3</u></td><td></td><td></td></tr>
<tr><td>CU/</td><td>A</td><td>I≠</td><td>I#</td><td>W</td><td>N</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Device</td><td>¢</td><td>1*@¤</td><td>1*@</td><td>1*@¤</td><td>3@</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

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

N   Not operational

*   CSW stored

¢
- When a nonimmediate I/O operation has been initiated and the channel is proceeding with its execution, when an immediate operation has been initiated and command chaining takes place, or when command retry is signaled in response to the first command and is honored by the channel, condition code 0 is set.
- When an immediate operation has been initiated, and no command chaining or command retry is taking place, or the device is not ready, or an error has been detected by the control unit or device, for the SIO function condition code 1 is set, and the CSW is stored. Under the same circumstances, for the SIOF function, condition code 0 is set, and subsequently an I/O-interruption condition is generated. The CSW stored when the I/O-interruption condition is cleared contains the same information as the CSW stored during the SIO function under the same conditions, plus the deferred-condition-code-1 indication.

@   The SIOF function may cause condition code 0 to be set, in which case the other condition code shown will be specified as a deferred condition code.

&   When the subchannel is interruption-pending because an I/O operation is concluded at the subchannel, condition code 2 is set. When the subchannel is interruption-pending for any other reason, condition code 1 is set, and the status portion of the CSW is stored with a one included in the busy-bit position of the unit-status field.

$ The AIX state only occurs on the multiplexer channel.

+   With start-I/O-fast queuing, the channel-working state (WXX) is normally treated the same as the available channel state for the purpose of performing the SIOF function and the condition-code setting depends on the state of the subchannel. When the working state of the channel precludes the acceptance of the SIOF function, however, condition code 2 is set. When the block-multiplexing-control bit is zero, it causes the SIO function to be performed instead of the SIOF function, so condition code 2 is set.

¤   When the SIOF function causes condition code 0 to be set and the subchannel is provided with start-I/O-fast queuing, the I/O operation may remain pending at the subchannel instead of being terminated.

Note: Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by START I/O and START I/O FAST RELEASE

Programming Notes

1. The instruction START I/O FAST RELEASE has the advantage over START I/O that the CPU can be released after the CAW is fetched, rather than after completion of a possibly lengthy device-selection procedure. 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.

2. When the channel detects a programming error during execution of the SIO function, when the addressed device contains an interruption condition, and when the channel and subchannel are available, the instruction may or may not clear the interruption condition, depending on the type of error and the system 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.

3. Two major differences exist between the SIO and SIOF functions:

   a. Unchained immediate commands on certain channels (that is, those which execute SIOF independent of the device) result in a condition code 0 for the SIOF function, whereas condition code 1 is set for the SIO function. See also programming note 2 in the section "Command Retry" of this chapter.

   b. Condition code 0 is set by these certain channels for the SIOF function, 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 condition.

4. Subsequent to an I/O interruption signaling the conclusion of an I/O operation at the subchannel but not at the device (as, for example, when the device status contains channel end without device end), the subchannel is available while the device remains working. With start-I/O-fast queuing, START I/O FAST RELEASE addressed to the device in this case causes the new I/O operation to be made pending at the subchannel. The new I/O operation remains pending at the subchannel until the device signals the conclusion of the previous I/O operation by presenting status containing the device-end indication. When this occurs, the device end, either alone or with control-unit end, may be interpreted by the channel as a no-longer-busy indication. If the status is interpreted as a no-longer-busy indication, the channel attempts to initiate the new pending I/O operation at the device. In this case, the device-end status or device-end and control-unit-end status for the previous operation is discarded by the channel and is not made available to the program. Otherwise, the device-end indication is interpreted by the channel as unsolicited status, and an interruption condition is recognized.

STORE CHANNEL ID

STIDC $D_2(B_2)$ [S]

'B203'	B_2	D_2

0 16 20 31

Information identifying the designated channel is stored in the four-byte field at real storage location 168.

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 locations 168-171 is:

Type	Channel Model	Max IOEL Length

0 4 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:

Bits 0	 1	 2*	 3	Channel Type
0	0	0	0	Selector
0	0	0	1	Byte multiplexer
0	0	1	0	Block multiplexer

* When STORE CHANNEL ID is executed, the setting of bit 2 is unpredictable when bit 3 of the channel type code is stored as zero and bit 0 of control register 0 is (1) currently set to zero or (2) was set to zero when a previous START I/O or START I/O FAST RELEASE was executed and at least one subchannel is currently in the working or interruption-pending state because of executing the function of that previous instruction.

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. If the channel never stores logout information using the IOEL address, 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 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 occurs in this case.

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 in the figure "Condition Codes Set by STORE CHANNEL ID." 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, condition code 2 is set. If the working or interruption-pending state does not preclude the storing of the channel ID, 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 8-14 of the instruction are ignored.

Bits 16-23 of the second-operand address identify the channel to which the instruction applies. Bit positions 24-31 of the address are ignored.

The TCH function is performed by the TEST CHANNEL instruction and, on some channels and under certain circumstances, by CLEAR CHANNEL.

The TCH function inspects only the state of the addressed channel. It tests whether the channel is operating in the burst mode, is interruption-pending, or is not operational. When the channel is operating in the burst mode and contains an interruption condition, the condition code is set as for operation in the burst mode. When none of these situations exist, the available state is indicated. No device is selected, and, on the multiplexer channel, the subchannels are not interrogated.

Program Exceptions:

Privileged operation

Resulting Condition Code:

0 Channel available
1 Interruption or logout condition in channel
2 Channel operating in burst mode
3 Channel not operational

The condition code set by TEST CHANNEL for all possible states of the addressed channel is shown in the figure "Condition Codes Set by TEST CHANNEL." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

	A	I	W	N
Channel	0	1	2	3

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]

'9D00'	B_2	D_2

0 16 20 31

The state of the addressed channel, subchannel, and device is indicated by setting the condition code in the PSW and, in certain situations, by storing the CSW. Interruption conditions may be cleared. Bits 8-14 of the instruction are ignored.

Bits 16-31 of the second-operand address identify the channel, subchannel, and I/O device to which the instruction applies.

The TIO function is performed by the instruction TEST I/O and, on some channels and under certain circumstances, by CLEAR I/O.

When the channel is operating in burst mode and the addressed subchannel contains an interruption condition for the addressed device, the TIO function causes condition code 1 or 2 to be set, depending on the model and channel type. If condition code 1 is set, the CSW is stored to identify the interruption condition, and the interruption condition is cleared. The interruption condition in the subchannel is not cleared, and the CSW is not stored if the channel is working and has not yet accepted the status causing the interruption condition from the device. Condition code 2 is set in this case.

When the channel is either available or interruption-pending and the addressed subchannel is either interruption-pending for a different device or working, the TIO function causes condition code 2 to be set.

When either of the situations described in the following two paragraphs occurs with the channel either available or interruption-pending or, on some channels, working, the TIO function causes the CSW to be stored. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

1. The subchannel is interruption-pending for the addressed device, and the interruption condition is due to the termination of an I/O operation at the subchannel. When the CSW is stored, the interruption condition is cleared. The CSW fields contain the final values for the I/O operation. The unit-status and/or channel-status fields contain indications provided by the device or channel respectively, which identify the interruption condition and any other conditions detected by the channel or device.

2. The subchannel is interruption-pending for the addressed device, and the interruption condition is not due to the termination of an I/O operation at the subchannel. When the CSW is stored, the interruption condition is cleared. The subchannel key, CCW address, and count fields are stored as zeros. The unit-status field contains indications provided by the device which identify the interruption condition. The channel-status field contains zeros unless a channel equipment error is detected.

When any of the following situations occurs with the channel either available or interruption-pending, the TIO function causes the CSW to be stored. The contents of the entire CSW pertain to the I/O device addressed by the instruction.

1. The subchannel is available, and the I/O device contains an interruption condition or the control unit contains control-unit end for the addressed device. The CSW unit-status field identifies the interruption condition and may contain other bits provided by the device or control unit. The interruption condition is cleared. The busy bit in the CSW is zero. The other fields of the CSW contain zeros unless an equipment error is detected.

2. The subchannel is available, and the I/O device or the control unit is executing a previously initiated operation or the control unit has an interruption condition associated with a device other than the one addressed. The CSW unit-status field contains one in the busy-bit position or, if the control unit is busy, the busy and status-modifier bits are ones. Other fields of the CSW contain zeros unless an equipment error is detected.

3. The subchannel is available, and the I/O device or channel detected an equipment error during execution of the instruction or the addressed device is not ready and does not have any interruption condition. The CSW identifies the error. If the device is not ready, unit check is indicated. No interruption conditions are generated at the I/O device or the subchannel.

When the TIO function cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones. No channel logout is associated with this status.

When the TIO function is used to clear an interruption condition signaling conclusion of an I/O operation at the subchannel and the channel has not yet accepted the condition from the device, the function causes the device to be selected and the interruption condition in the device to be cleared. During certain I/O operations, some types of devices cannot provide their current status in response to TEST I/O. Some magnetic-tape control units, for example, are in such a state when they have provided channel end and are executing the backspace-file operation. When TEST I/O is issued to a control unit in such a state, the unit-status field of the CSW has the busy and status-modifier bits set to ones, with zeros in the other CSW fields. The interruption condition in the device and in the subchannel is not cleared.

On some types of devices, the device never provides its current status in response to TEST I/O, and an interruption condition can be cleared only by permitting an I/O interruption, by I/O-system reset, or by I/O selective reset. When TEST I/O is issued to such a device, the unit-status field has the status-modifier bit set to one, with zeros in the other CSW fields. The interruption condition in the device and in the subchannel, if any, is not cleared.

However, by the time the channel assigns the highest priority for interruptions to a condition associated with an operation at the subchannel, the channel has accepted the status from the device and cleared the corresponding condition at the device. Some channels accept and clear an interruption condition signaling the conclusion of an I/O operation at the subchannel from the device before it is assigned the highest priority for interruptions. Other channels may accept and clear any type of interruption condition from the device prior to assigning it the highest priority for interruptions. The acceptance of an interruption condition from a device causes the associated subchannel to enter the interruption-pending state. When the channel recognizes an interruption condition signaling the conclusion of an I/O operation at the subchannel, the associated subchannel enters the interruption-pending state even when the interruption condition has not yet been accepted from the device.

When the TIO function is addressed to a device for which the channel has already accepted the interruption condition, the device is not selected, and the condition in the subchannel is cleared regardless of the type of device and its present state. The CSW contains unit status and other information associated with the interruption condition.

On the byte-multiplexer channel, the TIO function causes the addressed device to be selected only after the channel has serviced all outstanding requests for data transfer for previously initiated operations.

Program Exceptions:

Privileged operation

Resulting Condition Code:

0 Available
1 CSW stored
2 Channel or subchannel busy
3 Not operational

The condition code set by the TIO function for all possible states of the I/O system is shown in the figure "Condition Codes Set by TEST I/O." See the section "States of the Input/Output System" in this chapter for a detailed definition of the A, I, W, and N states.

<table>
<tr><td>Channel</td><td colspan="8">A</td><td colspan="8">I</td><td colspan="5">W≠</td><td>W#</td><td>N</td></tr>
<tr><td>Subchannel</td><td colspan="4">A</td><td>I≠</td><td>I#</td><td>W</td><td>N</td><td colspan="4">A</td><td>I≠</td><td>I#</td><td>W</td><td>N</td><td>A</td><td>I≠</td><td>I#</td><td>W</td><td>N</td><td>2</td><td>3</td></tr>
<tr><td></td><td colspan="4"></td><td><u>2</u></td><td><u>1</u>*</td><td><u>2</u></td><td><u>3</u></td><td colspan="4"></td><td>2</td><td>1*</td><td><u>2</u></td><td><u>3</u></td><td><u>2</u></td><td><u>2</u></td><td><u>@</u></td><td>2</td><td><u>2</u></td><td></td><td></td></tr>
<tr><td></td><td>A</td><td>I</td><td>W</td><td>N</td><td></td><td></td><td></td><td></td><td>A</td><td>I</td><td>W</td><td>N</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>CU/
Device</td><td>0</td><td>1*</td><td>1*</td><td>3</td><td></td><td></td><td></td><td></td><td>0</td><td>1*</td><td>1*</td><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

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 activity in the channel.

Note: Underscored condition codes pertain to situations that can occur only on the multiplexer channel.

Condition Codes Set by TEST I/O

Programming Notes

1. Disabling the CPU for I/O interruptions provides the program with a means of controlling the priority of I/O interruptions selectively by channels. The priority of devices attached on a channel cannot be controlled by the program. The instruction TEST I/O in some cases permits the program to clear interruption conditions selectively by I/O device.

2. When a CSW is stored by the TIO function, the interface-control-check and channel-control-check indications may be due to an interruption condition already existing in the channel or may be due to an interruption condition created by the TIO function. Similarly, the unit-check bit set to one with the channel-end, control-unit-end, or device-end bits set to zeros may be due to a situation created by the preceding operation, the I/O device being not ready, or an equipment error detected during the execution of TEST I/O. The instruction TEST I/O cannot be used to clear an interruption condition due to the PCI flag while the subchannel is working.

3. The use of a TEST I/O loop on a multiplexer channel to retrieve ending status for a channel program should, in general, be avoided. TEST I/O loops may be used to return ending status to a sense command when that command was initiated by a START I/O that received condition code 0. TEST I/O loops under other conditions may result in hang conditions.

4. In some models, the use of a disabled-TIO-loop procedure to detect the completion of an I/O operation initiated by SIOF may cause a deadlock condition. The deadlock occurs if SIOF is issued to a subchannel for which start-I/O-fast queuing is provided, and conditions are such that a pending I/O interruption must be cleared before the pending I/O operation can be initiated by the channel. This is another example where a disabled TIO loop does not work reliably.

INPUT/OUTPUT-INSTRUCTION-EXCEPTION HANDLING

Before the channel is signaled to execute an I/O instruction, the instruction is tested for validity by the CPU. Exceptional situations detected at this time cause a program interruption.

The following exception causes a program interruption:

Privileged Operation: An I/O instruction is encountered when the CPU is in the problem state. The instruction is suppressed before the channel has been signaled to execute it. The CSW, the condition code in the PSW, and the state of the addressed subchannel and I/O device are not affected by the attempt to execute an I/O instruction while in the problem state.

EXECUTION OF INPUT/OUTPUT OPERATIONS

The channel can execute six commands: write, read, read backward, control, sense, and transfer in channel. Each command except transfer in channel initiates a corresponding I/O operation. The term "I/O operation" refers to the activity initiated by a command in the I/O device and associated subchannel. The subchannel is involved with the execution of the operation from the initiation of the command until the channel-end signal is received or, in the case of command chaining, until the device-end signal is received. The operation in the device lasts until device end is signaled.

BLOCKING OF DATA

Data recorded by an I/O device may be divided into blocks. The length of a block depends on the device; for example, a block can be a card, a line of printing, or the information recorded between two consecutive gaps on magnetic tape.

The maximum amount of information that can be transferred in one I/O operation is one block. An I/O operation is terminated when the associated storage area is exhausted or the end of the block is reached, whichever occurs first. For some operations, such as writing on a magnetic-tape unit or at an inquiry station, blocks are not defined, and the amount of information transferred is controlled only by the program.

CHANNEL-ADDRESS WORD

The channel-address word (CAW) specifies the subchannel key, the suspend-control bit, 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 storage 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:

Key	S	000	CCW Address
0	4	8	31

The fields in the CAW are allocated for the following purposes:

<u>Subchannel</u> <u>Key</u>: Bits 0-3 form the access key for all fetching of CCWs, IDAWs, and output data and for the storing of input data associated with START I/O and START I/O FAST RELEASE. This key is matched with a storage key during these storage references. For details, see the section "Key-Controlled Protection" in Chapter 3, "Storage."

<u>Suspend</u> <u>Control</u> (<u>S</u>): Bit 4 of the CAW controls execution of the suspend function for the channel program identified by the CAW. The setting of the suspend-control bit applies to the channel program specified by the CAW. When bit 4 is set to one, suspend control is specified, and channel-program suspension occurs when a valid S flag is detected in a CCW. When bit 4 is set to zero, suspend control is not specified, and the presence of the S flag in any CCW of the channel program causes the program-check condition to be recognized.

If the suspend-and-resume facility is not provided for the model, the suspend-control bit must be zero; otherwise, a program-check condition is recognized, and the I/O operation is not initiated at the device. When the suspend-and-resume facility is provided for the model but the suspend function is not available for the addressed subchannel, bit 4 is ignored, but any occurrence of an S flag in the channel program causes a program-check condition to be recognized.

<u>CCW</u> <u>Address</u>: Bits 8-31 designate the location of the first CCW in absolute storage.

Bit positions 5-7 of the CAW must contain zeros. The three rightmost bits of the CCW address must be zeros to specify the CCW on integral boundaries for doublewords. If either of these restrictions is violated, an error condition is recognized during the execution of START I/O or START I/O FAST RELEASE. If the CCW address specifies a storage location which is not available or is protected against fetching, START I/O and, in some cases, START I/O FAST RELEASE, cause an error condition to be recognized. When a programming-error condition is recognized during the execution of START I/O or START I/O FAST RELEASE, the status portion of the CSW is stored, with the protection-check or program-check bit set to one. In this event, the I/O operation is not initiated at the device.

Programming Note

Bit positions 5-7 of the CAW, which presently must contain zeros, may in the future be assigned to the control of new functions. It is, therefore, recommended that these bit positions not be set to ones for the purpose of obtaining an intentional program-check indication.

CHANNEL-COMMAND WORD

The channel-command word (CCW) specifies the command to be executed and, for commands initiating I/O operations, it designates the storage area associated with the operation, the action to be taken whenever transfer to or from the area is completed, and other options. The CCWs can be located at any available location in the first 16M-byte block of storage, and more than one can be associated with a START I/O or START I/O FAST RELEASE.

The first CCW is fetched during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When START I/O FAST RELEASE is executed independent of the device, the first CCW may be fetched subsequent to the execution of START I/O FAST RELEASE. Each additional CCW in the sequence is obtained when the operation has progressed to the point where the additional CCW is needed. Fetching of the CCWs by the channel does not affect the contents of the location in storage.

Except for a CCW containing the transfer-in-channel command, the CCW has the following format:

Cmd Code	Data Address	
0	8	31

Flags	0	/////////	Count	
32	39		48	63

The fields in the CCW are allocated for the following purposes:

Command Code: Bits 0-7 specify the operation to be performed.

Data Address: Bits 8-31 specify a location in absolute 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 and S flag are zeros, specifies chaining of commands. It causes the operation specified by the command code in the next CCW to be initiated on normal completion of the current operation.

Suppress-Length-Indication (SLI) Flag: Bit 34 controls whether incorrect-length is to be indicated to the program. When this bit is one and the CD flag is zero, the incorrect-length indication is suppressed. When both the CC and SLI flags are one and the CD flag is zero, command chaining takes place regardless of any incorrect-length situation.

Skip (SKIP) Flag: Bit 35, when one, specifies suppression of the transfer of information to storage during a read, read backward, or sense operation.

Program-Controlled-Interruption (PCI) Flag: Bit 36, when one, causes the channel to generate an interruption condition when the CCW takes control of the channel. When bit 36 is zero, normal operation takes place.

Indirect-Data-Address (IDA) Flag: Bit 37, when one, specifies indirect data addressing.

Suspend (S) Flag: Bit 38, when set to one, specifies suspension of channel-program execution. When valid, it causes channel-program execution to be suspended prior to execution of the CCW containing the S flag.

Count: Bits 48-63 specify the number of bytes in the storage area designated by the CCW.

Bit position 39 of every CCW other than one specifying transfer in channel must contain zero. Otherwise, a program-check condition is generated. When the first CCW designated by the CAW does not contain zero in bit position 39, the I/O operation is not initiated, and the status portion of the CSW with the program-check indication is stored during execution of START I/O or START I/O FAST RELEASE being executed as START I/O. Detection of this condition during data chaining causes the I/O device to be signaled to conclude the operation. When the absence of these zeros is detected during command chaining or subsequent to the execution of START I/O FAST RELEASE, the new operation is not initiated, and an interruption condition is generated.

The contents of bit positions 40-47 of the CCW are ignored. If the command code specifies the transfer-in-channel command, bit positions 32-63 of the CCW are ignored. For the format of the CCW containing the transfer-in-channel command, see the section "Transfer in Channel" later in this chapter.

Programming Note

Bit position 39 of the CCW, which presently must be set to zero, may in the future be assigned to the control of new functions. It is recommended, therefore, that this bit position not be set to one for the purpose of obtaining an intentional program-check indication.

COMMAND CODE

The command code, bit positions 0-7 of the CCW, specifies to the channel and the I/O device the operation to be performed. A detailed description of each command appears under "Commands."

The two rightmost bits or, when these bits are 00, the four rightmost 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 leftmost bits of the command code.

Commands that initiate I/O operations (write, read, read backward, control, sense, and sense ID) cause all eight bits of the command code to be transferred to the I/O device. In these command codes, the leftmost bit positions contain modifier bits. The modifier bits specify to the device how the command is to be executed. They may, for example, cause the device to compare data received during a write operation with data previously recorded, and they may specify such information as recording density and parity. For the control command, the modifier bits may contain the order code specifying the control function to be performed. The meaning of the modifier bits depends on the type of I/O device and is specified in the SL publication for the device.

The command-code assignment is listed in the following table. The symbol X indicates that the bit position is ignored; M identifies a modifier bit.

Code	Command
XXXX 0000	Invalid
MMMM MM01	Write
MMMM MM10	Read
MMMM 1100	Read Backward
MMMM MM11	Control
MMMM 0100	Sense
1110 0100	Sense ID
XXXX 1000	Transfer in Channel

Whenever the channel detects an invalid command code during the initiation of a command, a program check is generated. When the first CCW designated by the CAW contains an invalid command code, the status portion of the CSW with the program-check indication is stored during execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When the invalid code is detected during command chaining or subsequent to the execution of START I/O FAST RELEASE, the new operation is not initiated, and an interruption condition is generated. The command code is ignored during data chaining, unless it specifies transfer in channel.

DESIGNATION OF STORAGE AREA

The storage area associated with an I/O operation is defined by one or more CCWs. A CCW defines an area by specifying the address of the first byte to be transferred and the number of consecutive bytes contained in the area. The address of the first byte appears in the data-address field of the CCW, except when channel indirect data addressing is specified. (See the section "Channel Indirect Data Addressing" later in this chapter.) The number of bytes contained in the storage area is specified in the count field.

In write, read, control, and sense operations, storage locations are used in ascending order of addresses. As information is transferred to or from storage, the address from the address field is incremented, and the count from the count field is decremented. The read-backward operation places data in storage in a descending order of addresses, and both the count and the address are decremented. When the count reaches zero, the storage area defined by the CCW is exhausted.

Some channels do not perform address wraparound. Depending on the model, a program check may be generated if an address generated by the channel to transfer data is incremented past 16,777,215 or is decremented past 0.

Any available storage location can be used in the transfer of data to or from an I/O device if the location is not protected against the type of reference. Similarly, a CCW can be located in any available storage location (in the first 16M-byte block of storage) if the location is not protected against a fetch-type reference.

When the first CCW designated by the CAW is in a storage location that is not available, the I/O operation is not initiated, and the status portion of the CSW with the program-check indication is stored during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When, subsequently, during the operation or chain of operations, the channel refers to a storage location that is not provided, an interruption condition indicating program check is generated, and the device is signaled to terminate the operation.

When the first CCW designated by the CAW is in a storage location that is protected against a fetch-type reference, the I/O operation is not initiated, and the status portion of the CSW with the protection-check indication is stored during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When, subsequently, during the I/O operation or chain of operations, the channel refers to a protected storage location, an interruption condition indicating protection check is generated, and the device is signaled to terminate the operation.

During an output operation, the channel may fetch data from storage before the time the I/O device requests the data. Any number of bytes specified by the current CCW may thus be prefetched. When data chaining during an output operation, the channel may prefetch the next CCW and the data and IDAWs associated with the prefetched CCW (as specified by the data-address and count field of the CCW or the data addresses from the IDAWs and the count field of the CCW) at any time during the execution of the current CCW.

Prefetching may cause the channel to refer to storage locations that are protected or not available. Such errors detected during prefetching of data, CCWs, or IDAWs, do not affect the execution of the operation and do not cause error indications until the I/O operation actually attempts to use the data or until the CCW or IDAW takes control. If the operation is concluded by the channel, by the I/O device, or by the HIO, HDV, CLRCH, or CLRIO function before the invalid information is needed, no program check or protection check is generated.

The count field in the CCW can specify any number of bytes from one to 65,535. Except for a CCW specifying transfer in channel, which has no count field, the count field may not contain the value zero. Whenever the count field in the CCW initially contains a zero, a program check is generated. When this occurs in the first CCW designated by the CAW, the operation is not initiated, and the status portion of the CSW with the program-check indication is stored during execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When a count of zero is detected during data chaining, the I/O device is signaled to terminate the operation. Detection of a count of zero during command chaining or subsequent to the execution of START I/O FAST RELEASE suppresses initiation of the new operation and generates an interruption condition.

CHAINING

When the channel has performed the transfer of information specified by a CCW, it can continue the activity initiated by START I/O or START I/O FAST RELEASE by fetching a new CCW. Such fetching of a new CCW is called chaining, and the CCWs belonging to such a sequence are said to be chained.

Chaining takes place between CCWs located in successive doubleword locations in storage. It proceeds in an ascending order of addresses; that is, the address of the new CCW is obtained by adding 8 to the address of the current CCW. Two chains of CCWs located in noncontiguous storage areas can be coupled for chaining purposes by a transfer-in-channel command. All CCWs in a chain apply to the I/O device specified in the original START I/O or START I/O FAST RELEASE. Depending on the model, the address used to fetch a CCW may wrap from 16,777,208 to 0, or a program check may be generated when that CCW takes control of the operation.

Two types of chaining are provided: chaining of data and chaining of commands. Chaining is controlled by the chain-data (CD) and chain-command (CC) flags in conjunction with the suppress-length-indication (SLI) flag in the CCW. These flags specify the action to be taken by the channel upon the exhaustion of the current CCW and upon receipt of ending status from the device, as shown in the figure "Channel-Chaining Action."

The specification of chaining is effectively propagated through a transfer-in-channel command. When in the process of chaining, a transfer-in-channel command is fetched, the CCW designated by the transfer in channel is used for the type of chaining specified in the CCW preceding the transfer in channel.

The CD and CC flags are ignored in the transfer-in-channel command.

Note: For a description of the storage area associated with a CCW when channel indirect data addressing is invoked, see the section "Channel Indirect Data Addressing" later in this chapter.

Flags in Current CCW			Action in Channel upon Exhaustion of Count or Receipt of Channel End			
				Nonimmediate Operation		
CD	CC	SLI	Immediate Operation	I	II	III
0	0	0	End, NIL	Stop, IL	End, NIL	End, IL
0	0	1	End, NIL	Stop, NIL	End, NIL	End, NIL
0	1	0	Chain Command	Stop, IL	Chain command	End, IL
0	1	1	Chain Command	Chain command	Chain command	Chain command
1	-	-	End, NIL	Chain Data	*	End, IL

Explanation:

- May be either zero or one.

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 START I/O (or START I/O FAST RELEASE executed as START I/O), condition code 1 is set, and the status portion of the CSW is stored as part of the execution of the instruction. 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.

NIL 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 residual count is zero but data chaining is indicated at the time the device provides channel end cannot validly occur. When data chaining is indicated, the channel fetches the new CCW after transferring the last byte of data designated by the current CCW but before the device provides the next request for data or status transfer. As a result, the channel recognizes the channel end from the device only after it has fetched the new CCW, which cannot contain a count of zero unless a programming error has been made.

Channel-Chaining Action

Data Chaining

During data chaining, the new CCW fetched by the channel defines a new storage area for the original I/O operation. Execution of the operation at the I/O device is not affected. When all data designated by the current CCW has been transferred to storage or to the device, data chaining causes the operation to continue, using the storage area designated by the new CCW. The contents of the command-code field of the new CCW are ignored, unless they specify transfer in channel.

Data chaining is considered to occur immediately after the last byte of data designated by the current CCW has been transferred to storage or to the device. When the last byte of the transfer has been placed in storage or accepted by the device, the new CCW takes over the control of the operation and replaces the pertinent information in the subchannel. If the device signals channel end after exhausting the count of the current CCW but before transferring any data to or from the storage area designated by the new CCW, the CSW associated with the concluded operation pertains to the new CCW.

If programming errors are detected in the new CCW or during its fetching, the error indication is generated, and the device is signaled to conclude the operation when it attempts to transfer data designated by the new CCW. If the device signals channel end after the new CCW takes control but before transferring any data designated by the new CCW, program check or protection check is indicated in the CSW associated with the termination. The contents of the CSW pertain to the new CCW unless a program check or protection check is generated while fetching the new CCW or while fetching or executing an intervening transfer-in-channel command. A data address which causes a program check or protection check gives an error indication only after the I/O device has attempted to transfer data to or from the addressed storage location.

If the chain-data flag is set to one in the current CCW, and the count has not been exhausted when the device signals channel end, the operation is terminated. An interruption condition is generated in the subchannel with incorrect length indicated. The incorrect-length condition is indicated regardless of the setting of the SLI bit in the current CCW.

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 storage. On an output operation, the channel may fetch the new CCW and the data and IDAWs associated with the prefetched CCW (as specified by the data-address field of the CCW or the data-address fields from the IDAWs and the count field of the CCW) from storage before data chaining occurs. Any programming errors in a prefetched CCW, however, do not affect the execution of the operation until all data designated by the current CCW has been transferred to the I/O device. If the device concludes the operation before all data designated by the current CCW has been transferred or if data chaining is suppressed for any other reason, the errors associated with the prefetched CCW are not indicated to the program.

During an output operation, the channel may prefetch only one CCW describing a data area; however, the data and IDAWs associated with the prefetched CCW may also be prefetched. If the prefetched CCW specifies transfer in channel, only one more CCW may be fetched before the exhaustion of the current CCW.

Programming Notes

1. Data chaining may be used to rearrange data as it is transferred between storage and an I/O device. Data chaining permits data to be transferred to or from noncontiguous areas of storage, and, when used in conjunction with the skipping function (see the section "Skipping" later in this chapter), data chaining enables the program to place in storage selected portions of a block of data.

   When, during an input operation, the program specifies data chaining to a location in which data has been placed under the control of the current CCW, the channel, in fetching the next CCW, fetches the new contents of the location. This is true even if the location contains the last byte transferred under the control of the current CCW. When, on input, a channel program data-chains to a CCW placed in storage by the CCW specifying data chaining, the block is said to be self-describing. A self-describing block contains one or more CCWs that specify storage locations and counts for subsequent data in the same block.

   The use of self-describing blocks is equivalent to the use of unchecked data. An I/O data-transfer malfunction that affects validity of a block is signaled only at the completion of data transfer. The error normally does not prematurely terminate or otherwise affect the execution of the operation. Thus, there is no assurance that a CCW read as data is valid until the operation is completed. If the CCW is in error, the use of the CCW in the current operation may cause subsequent data to be placed in wrong storage locations with resultant destruction of the contents of those locations.

2. When, during data chaining, an I/O device transfers data by using the data-streaming facility (see the section "Data-Streaming Feature" in Chapter 2 of the publication IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information, GA22-6974), an overrun or chaining-check condition may be recognized when a small count value is specified in the CCW. The minimum acceptable number of bytes that can be specified varies as a function of the system model and system activity. Refer to the appropriate channel SL publication to determine the most reasonable minimum byte count that can be handled by the channel.

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 (unless the new CCW contains a suspend flag) upon receipt of the device-end signal for the current operation. The presence of a suspend flag in the new CCW causes command chaining to be terminated. (See the section "Suspension of Channel-Program Execution" later in this chapter.) When command chaining takes place, the completion of the current operation does not generate an interruption condition, and the count indicating the amount of data transferred during the current operation is not made available to the program. For operations involving data transfer, the new command always applies to the next block at the device.

The new operation is initiated only if no unusual situations have been detected in the current operation. In particular, the channel initiates a new I/O operation by command chaining upon receipt of a status byte signaling one of the following status combinations: device end, device end and status modifier, device end and channel end, device end and channel end and status modifier. In the former two cases, channel end must have been signaled before device end, with all other status bits set to zeros. If status such as attention, unit check, unit exception, incorrect length, program check, or protection check has occurred, the sequence of operations is concluded, and the status associated with the current operation causes an interruption condition to be generated. The new CCW in this case is not fetched. Incorrect length does not suppress command chaining if the current CCW has the SLI flag set to one.

An exception to sequential chaining of CCWs occurs when the I/O device presents status modifier with device end. When no unusual conditions have been detected and command chaining is specified or when command retry has been previously signaled and an immediate retry could not be performed, the combination of status modifier and device end causes the channel to alter the sequential execution of CCWs. If command chaining was specified, status modifier and device end cause the channel to chain to the CCW whose storage address is 16 higher than that of the CCW that specified chaining. If command retry was previously signaled and immediate retry could not be performed, the status causes the channel to command-chain to the CCW whose storage address is 8 higher than that of the CCW for which retry was initially signaled.

When both command and data chaining are used, the first CCW associated with the operation specifies the operation to be executed, and the last CCW indicates whether another operation follows.

Programming Note

Command chaining makes it possible for the program to initiate transfer of multiple blocks by means of a single START I/O or START I/O FAST RELEASE. It also permits a subchannel to be set up for the execution of auxiliary functions, such as positioning the disk-access mechanism, and for data-transfer operations without interference by the program at the end of each operation. Command chaining, in conjunction with the status-modifier condition, permits the channel to modify the normal sequence of operations in response to signals provided by the I/O device.

SKIPPING

Skipping causes the suppression of storage references during an I/O operation. It is defined only for read, read backward, and sense operations and is controlled by the skip flag, which can be specified individually for each CCW. When the skip flag is one, skipping occurs; when zero, normal operation takes place. The setting of the skip flag is ignored in all other operations.

Skipping affects only the handling of information by the channel. The operation at the I/O device proceeds normally, and information is transferred to the channel. The channel keeps updating the count but does not place the information in storage. Chaining is not precluded by skipping. In the case of data chaining, normal operation is resumed if the skip flag in the new CCW is zero.

When the skip flag is set to one, the data address in the CCW is not checked.

Programming Note

Skipping, when combined with data chaining, permits the program to place in storage selected portions of a block from an I/O device.

PROGRAM-CONTROLLED INTERRUPTION

The program-controlled-interruption (PCI) function permits the program to cause an I/O interruption during the execution of an I/O operation. The function is controlled by the PCI flag in the CCW. The flag can be on either in the first CCW specified by START I/O or START I/O FAST RELEASE or in a CCW fetched during chaining or command retry. Neither the PCI flag nor the associated interruption affects the execution of the current operation.

Whenever the PCI flag in the CCW is one, an interruption condition is generated in the channel. When the first CCW associated with an operation contains the PCI flag, either initially or upon command chaining, the interruption may occur as early as immediately upon the initiation of the operation. The PCI flag in a CCW fetched on data chaining causes the interruption to occur after all data designated by the preceding CCW has been transferred. The time of the interruption, however, depends on the model and the current activity in the system and may be delayed even if I/O interruptions are allowed. No predictable relationship exists between the time the interruption due to the PCI flag occurs and the progress of data transfer to or from the area designated by the CCW, but the fields within the CSW pertain to the same instant of time.

If chaining occurs before the interruption due to the PCI flag has taken place, the PCI interruption condition is carried over to the new CCW. This carryover occurs both on data and command chaining and, in either case, the interruption condition is propagated through the transfer-in-channel command. The interruption conditions due to the PCI flags are not stacked; that is, if another CCW is fetched with a PCI flag before the interruption due to the PCI flag of the previous CCW has occurred, only one interruption takes place.

A CSW containing the PCI bit set to one may be stored by an interruption while the operation is still proceeding, while channel-program execution is suspended, or by an interruption, TEST I/O, or CLEAR I/O upon the termination of the operation. A CSW cannot be stored by TEST I/O while the subchannel is in the working state.

When the CSW is stored by an interruption before the operation or chain of operations has been concluded, the CCW address is 8 greater than the address of the CCW that contained the last recognized PCI flag or 8 greater than the address of a CCW which has subsequently become current, and the count is unpredictable. All unit-status bits in the CSW are zero. If the channel has detected any unusual situations, such as channel-data check, program check, or protection check by the time the interruption occurs, the corresponding channel-status bit is one, although the status in the subchannel is not reset and is indicated again upon the termination of the operation.

A unit-status bit set to one in the CSW indicates that the operation or chain of operations has been concluded. The CSW in this case has its regular format with the PCI bit set to one.

However, when the interruption due to the PCI flag is delayed until the operation at the subchannel is concluded, two interruptions from the subchannel may still take place. The first interruption indicates and clears the interruption condition due to the PCI flag, and the second provides the CSW associated with the ending status. Whether one or two interruptions occur depends on the model and on whether the interruption condition due to the PCI flag has been assigned the highest priority for interruption at the time of conclusion. TEST I/O or CLEAR I/O addressed to the device associated with an interruption condition in the subchannel clears the interruption condition due to the PCI flag, as well as the one associated with the conclusion.

The setting of the PCI flag is inspected in every CCW except those specifying transfer in channel, where it is ignored. The PCI flag is also ignored during initial program loading.

Programming Notes

1. Since no unit-status bits are set to ones in the CSW associated with the conclusion of an operation of a selector channel by HALT I/O or HALT DEVICE, unit-status bits and the PCI bit set to ones are not necessary for the operation to be concluded. When status in a selector channel includes PCI at the time the operation is concluded by HALT I/O or HALT DEVICE, the CSW associated with the concluded operation is indistinguishable from the CSW provided by an interruption during execution of the operation.

2. Program-controlled interruption provides a means of alerting the program to the progress of chaining during an I/O operation. It permits programmed dynamic storage allocation.

CHANNEL INDIRECT DATA ADDRESSING

Channel indirect data addressing permits a single channel-command word to control the transmission of data that spans noncontiguous pages in absolute storage.

Channel indirect data addressing is specified by a flag bit in the CCW which, when one, indicates that the data-address field is not used to directly address data. The contents of the data-address field specify the location of an indirect-data-address word (IDAW), which contains an absolute address designating a data area within storage. An IDAW is used for the transfer of up to 2K bytes. The IDAW specified by the CCW can designate any location. IDAWs can be located at any available location in the first 16M-byte block of storage.

Additional IDAWs, if needed for completing the data transfer for the CCW, are contained in successive storage locations. 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 4K bytes and the first IDAW specifies a location in the middle of a 2K-byte block, three IDAWs are required. 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 2K-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 2K-byte block. Thus, for read, write, control, and sense commands, these IDAWs have zeros in bit positions 21-31. For a read-backward command, these IDAWs 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 or to a prefetched CCW. An IDAW takes control of the data transfer when the last byte has been transferred for the previous IDAW. The same rules apply as with data chaining regarding when an IDAW takes control of data transfer during an I/O operation. That is, when the count in the CCW has not reached zero, a new IDAW takes control of the data transfer when the last byte has been transferred for the previous IDAW for that CCW, even in situations where (1) channel end, (2) channel end and device end, or (3) channel end, device end, and status modifier are received prior to transfer of any data bytes pertaining to the new IDAW. A prefetched IDAW does not take control of an I/O operation if the count in the CCW reached zero with the transfer of the last byte of data for the previous IDAW for that CCW. Errors detected in prefetched IDAWs are not indicated until the IDAW takes control of the data transfer. Depending on the model, addresses used to fetch an IDAW may wrap from 16,777,212 to 0, or a channel program check may be generated when that IDAW takes control of the operation.

Addressing Using the 24-Bit IDAW

The format of the IDAW and the significance of its fields when the 24-bit-IDAW facility is installed are as follows:

00000000	Data Address

0 8 31

Bit positions 0-7 are reserved for future use and must contain zeros; otherwise, a program-check condition is recognized.

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 2K-byte block located on a 2K-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 are zeros. For a read-backward command, the end of the block must be specified, and bits 21-31 of the IDAW are ones.

When the IDAW flag (bit 37) of the CCW is set to one and any of the following conditions occurs:

1. The address in the CCW does not designate the first IDAW on an integral word boundary,

2. The address in the CCW designates a storage location which is not available,

3. Access to the storage location specified by the address in the CCW is prohibited by protection, or

4. Bits 0-7 of the first IDAW are not zeros,

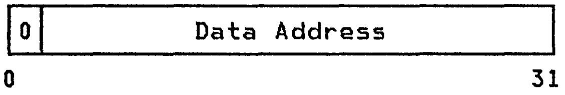
then, depending on the model, the above four conditions may be handled in one of two ways:

1. The channel checks for the above conditions before initiating the operation at the device. If any of these conditions is recognized, the channel does not initiate the operation with the device, and an interruption condition is generated.

2. The channel initiates the operation at the device prior to checking for these conditions. In this case, recognition of any of these conditions causes the channel to terminate execution of the I/O operation and generate an interruption condition only if the device attempts to transfer data.

Addressing Using the 31-Bit IDAW

The format of the IDAW and the significance of its fields when the 31-bit-IDAW facility is installed are as follows:



Bit position 0 is reserved for future use and must be zero. Otherwise, a program-check condition is recognized.

Bits 1-31 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 2K-byte block located on a 2K-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 are zeros. For a read-backward command, the end of the block must be specified, and bits 21-31 of the IDAW are ones.

When the IDAW flag (bit 37) of the CCW is set to one and any of the following conditions occurs:

1. The address in the CCW does not designate the first IDAW on an integral word boundary,

2. The address in the CCW does not designate a valid storage location,

3. Access to the storage location specified by the address in the CCW is prohibited by protection, or

4. Bit 0 of the first IDAW is not zero

then, depending on the model, the above four conditions may be handled in one of two ways:

1. The channel checks for the above conditions before initiating the operation at the device. If any of these conditions is recognized, the channel does not initiate the operation with the device, and an interruption condition is generated.

2. The channel initiates the operation at the device prior to checking for these conditions. In this case, recognition of any of these conditions causes the channel to terminate execution of the I/O operation and generate an interruption condition only if the device attempts to transfer data.

SUSPENSION OF CHANNEL-PROGRAM EXECUTION

The suspend function, when used in conjunction with the RIO function, provides the program with a means to stop and restart the execution of a channel program. The initiation of the suspend function is controlled by the setting of the suspend-control bit in the CAW. The suspend function is signaled to the channel during channel-program execution by the S flag in the CCW. The S flag in a CCW is not valid and causes a program-check condition to be recognized if (1) the CAW contains the suspend-control bit set to zero, (2) the CCW is fetched while data chaining (see the earlier section "Data Chaining" for the handling of programming errors detected during data chaining), or (3) the suspend function is not available for the subchannel.

The suspend-and-resume facility may be provided on an individual subchannel basis for nonshared subchannels. That is, if suspend-and-resume facilities are provided by the model, they are provided for one or more nonshared subchannels of one or more multiplexer channels. The suspend-and-resume facility is not provided for shared subchannels, including the subchannel of a selector channel.

When channel-program execution is initiated via SIO or SIOF executed while the block-multiplexing-control bit (bit 0 of control register 0) is zero, the suspend-and-resume facility, if provided for the subchannel, may or may not be operable. When the facility is not operable, detection of the S flag in a CCW causes the channel to recognize the program-check condition and terminate the operation.

Suspension occurs when a new CCW takes control that has a valid S flag. The command field of this CCW is not sent to the I/O device, and the device is signaled that the chain of commands is terminated. The CCW containing the S flag must be a valid CCW since all normal CCW checking is performed. A subsequent RESUME I/O instruction informs the channel that the suspend CCW may have been modified and that the channel must refetch the CCW and examine the current settings of the flags. The channel never executes a CCW with the S flag, regardless of the number of RIO instructions executed.

If the CCW containing the S flag also contains the PCI flag, an interruption condition is generated and made pending at the subchannel or device after channel-program execution is suspended. The PCI is presented to the program when it is allowed, regardless of whether the channel-program execution is still suspended or not. The suspend function, when used in conjunction with PCI, serves as a mechanism for alerting the program to the occurrence of a suspension at the subchannel.

When the first CCW of an I/O operation has the suspend flag validly set to one, the operation is suspended prior to initiating the operation at the device. When this occurs, condition code 0 is set for START I/O. Thus, when suspension occurs on the first CCW, a START I/O initiating an immediate operation for which command chaining is not specified in the CCW causes a condition code 0, rather than a condition code 1, to be set.

__Programming Notes__

1. In certain situations, normal resumption of a suspended channel program may not be desired. Normal termination of the suspended program may be accomplished by:

   a. Executing HALT DEVICE addressed to the device.

   b. Modifying the CCWs in storage such that when channel-program execution is resumed, the first command issued to the device is a control command with modifier bits of all zeros (no-operation) and with no chain-command flag specified, and then issuing RESUME I/O.

2. If the command code of a CCW that caused suspension of channel-program execution is replaced by the transfer-in-channel command code (X8 hex) prior to executing RIO, the S flag need not be removed from the CCW because bits 32-63 of the CCW are ignored when the command is transfer in channel (TIC).

3. In some models, the suspend-and-resume facility is operable for a channel-program execution that is initiated on a block-multiplexer channel while the block-multiplexing-control bit (bit 0 of control register 0) is zero. In these models, channel-program execution occurs with multiplexing inhibited until the channel-program execution is suspended. When suspension occurs, the effect on the channel is the same as if block multiplexing had occurred. That is, the device is disconnected from the channel at the end of a block, and the subchannel remains in the working state. When this happens, the channel becomes available for a new SIO function for some other device.

   When the suspended channel-program execution is subsequently resumed, it is executed as if a new channel-program execution were initiated via an SIOF function with the block-multiplexing-control bit set to one. That is, block multiplexing is no longer inhibited after the channel-program execution is resumed.

COMMANDS

The figure "Channel-Command Codes" lists the command codes for the seven valid 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	S
Read	MMMM MM10	CD	CC	SLI	SKIP	PCI	IDA	S
Read backward	MMMM 1100	CD	CC	SLI	SKIP	PCI	IDA	S
Control	MMMM MM11	CD	CC	SLI		PCI	IDA	S
Sense	MMMM 0100	CD	CC	SLI	SKIP	PCI	IDA	S
Sense ID	1110 0100	CD	CC	SLI	SKIP	PCI	IDA	S
Transfer in channel	XXXX 1000							

<u>Explanation</u>:

CD	Chain data
CC	Chain command
SLI	Suppress length indication
SKIP	Skip
PCI	Program-controlled interruption
IDA	Indirect data addressing
M	Modifier bit
S	Suspend
X	Ignored

Channel-Command Codes

All flags have individual significance, except that the CC and SLI flags are ignored when the CD flag is set to one. The SLI flag is ignored on immediate operations, in which case the incorrect-length indication is suppressed, regardless of the setting of the flag. The PCI flag is ignored during initial program loading.

Each command is described below, and the format is illustrated.

Programming Notes

1. A malfunction that affects the validity of data transferred in an I/O operation is signaled at the end of the operation by means of unit check or channel-data check, depending on whether the device (control unit) or the channel detected the error. In order to make use of the checking facilities provided in the system, data read in an input operation should not be used until the end of the operation has been reached and the validity of the data has been checked. Similarly, on writing, the copy of data in storage should not be destroyed until the program has verified that no malfunction affecting the transfer and recording of data was detected.

2. An error condition may be recognized by the channel and the I/O operation terminated when 256 or more chained commands are executed with an I/O device and none of the executed commands result in the transfer of any data. When this condition is recognized, program check is indicated.

Write

MMMMMM01	Data Address
0	8 31

CD	CC	SLI	/	PCI	IDA	S	0	////////	Count
32			35				40	48	63

A write operation is initiated at the I/O device, and the subchannel is set up to transfer data from storage to the I/O device. Data in storage is fetched in an ascending order of addresses, starting with the address specified in the CCW.

A CCW used in a write operation is inspected for the CD, CC, SLI, S, 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

MMMMMM10	Data Address
0	8 31

CD	CC	SLI	SKIP	PCI	IDA	S	0	////////	Count
32							40	48	63

A read operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to storage. For devices such as magnetic-tape units, disk storage, and card equipment, the bytes of data within a block are provided in the same sequence as written by means of a write command. Data is placed in storage in an ascending order of addresses, starting with the address specified in the CCW.

A CCW used in a read operation is inspected for every flag -- CD, CC, SLI, SKIP, S, PCI, and IDA. Bit positions 0-5 of the CCW contain modifier bits.

Read Backward

MMMM1100	Data Address

0 8 31

CD	CC	SLI	SKIP	PCI	IDA	S	0	////////	Count

32 40 48 63

A read-backward operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to storage. On magnetic-tape units, read backward causes reading to be performed with the tape moving backward. 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 addresses, starting with the address specified in the CCW. The bits within a byte are in the same order as sent to the device on writing.

A CCW used in a read-backward operation is inspected for every flag -- CD, CC, SLI, SKIP, S, PCI, and IDA. Bit positions 0-3 of the CCW contain modifier bits.

Control

MMMMMM11	Data Address

0 8 31

CD	CC	SLI	/	PCI	IDA	S	0	////////	Count

32 35 40 48 63

A control operation is initiated at the I/O device, and the subchannel is set up to transfer data from storage to the device. The device interprets the data as control information. The control information, if any, is fetched from storage in an ascending order of addresses, starting with the address specified in the CCW. A control command may be used to initiate at the I/O device an operation not involving transfer of data, such as backspacing or rewinding magnetic tape or positioning a disk-access mechanism.

For many control functions, the entire operation is specified by the modifier bits in the command code, and the function is performed as an immediate operation (see the section "Immediate Operations" later in this chapter). If the command code does not specify the entire control function, the data-address field of the CCW designates the location containing the required additional information. This control information may include a code further specifying the operation to be performed or an external address, such as the disk address for the seek function, and is transferred in response to requests by the device.

A control command code containing zeros for the six modifier bits is defined as a no-operation. The no-operation order causes the addressed device to respond with channel end and device end without causing any action at the device. The control command can be executed as an immediate operation, or the device can delay the status until after the initial selection sequence is completed. Other operations that can be initiated by means of the control command depend on the type of I/O device. These operations and their codes are specified in the SL publication for the device.

A CCW used in a control operation is inspected for the CD, CC, SLI, S, 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 (other than transfer in channel) with a count of zero is invalid, the program cannot use the CCW count field to specify that no data be transferred to the I/O device. Any operation terminated before data has been transferred causes the incorrect-length indication, provided the operation is not immediate and has not been rejected during the initiation sequence. The incorrect-length indication is suppressed when the SLI flag is on and the CD flag is off.

Sense

MMMM0100	Data Address

0 8 31

CD	CC	SLI	SKIP	PCI	IDA	S	0	////////	Count

32 40 48 63

A sense operation is initiated at the I/O device, and the subchannel is set up to transfer data from the device to storage. The data is placed in storage in an ascending order of addresses, starting with the address specified in the CCW.

The sense command is similar to a read command except that the data is obtained from sense indicators rather than from a record source.

The basic sense command (modifier bits set to zeros) initiates a sense operation on all I/O devices and causes the retrieval of up to 32 bytes of data. The basic sense command does not initiate any operation other than the reading of sense indicators. The basic sense command sent to an addressable control unit is accepted even though the addressed I/O device is in the not-ready state. If the control unit detects an error during the sense operation, unit check is sent with the channel-end status condition.

The purpose of the basic sense command is to provide data detailed enough to ascertain the actual state of the device and unusual conditions associated with the execution of the I/O operation during which the error was detected.

The first six bits of the first sense data byte (sense byte 0) retrieved by the basic sense command are common to all I/O devices. The six bits, when set to ones, designate the following:

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 indicated when the program issues an invalid sequence of commands, such as write to a direct-access-storage device without previous designation of the block. Command reject may also be indicated when invalid data is transferred and the data is treated as an extension of the command. For example, command reject is indicated when an invalid seek argument is transferred to a direct-access-storage device.

Intervention Required: The last operation could not be executed because of a situation requiring some type of intervention at the device. This bit indicates situations such as the hopper in a card punch being empty or the printer being out of paper. It is also turned on when the addressed device is not ready, is in test mode, or is not provided on the control unit.

Bus-Out Check: The device or the control unit has received a data byte or a command code with an invalid parity from the channel. During writing, bus-out check indicates that incorrect data may have been recorded at the device, but this does not cause the operation to be terminated prematurely. Parity errors on command codes and control information cause the operation to be immediately terminated and suppress checking for situations that would cause command reject and intervention required.

Equipment Check: During the last operation, the device or the control unit has detected equipment malfunctioning, such as an invalid card-hole count or a printer-buffer parity error.

Data Check: The device or the control unit has detected a data error other than those included in bus-out check. Data check identifies errors associated with the recording medium and includes errors such as reading an invalid card code or detecting invalid parity on data recorded on magnetic tape.

On an input operation, data check indicates that incorrect data may have been placed in storage. The control unit forces correct parity on data sent to the channel. On writing, data check indicates that incorrect data may have been recorded at the device. Unless the operation is of a type where the error precludes meaningful continuation, data errors on reading and writing do not cause the operation to be terminated prematurely.

Overrun: The overrun condition occurs when the channel fails to respond to the control unit in the anticipated time interval to a request for service from the I/O device. When the total activity initiated by the program exceeds the capability of the channel, an overrun may occur when data is transferred to or from a control unit that is either using the data-streaming facility or is not buffered. Data streaming is described in the publication *IBM System/360 and System/370 I/O Interface Channel to Control Unit Original Equipment Manufacturers' Information*, GA22-6974. An overrun condition also may occur when the I/O device receives the new command too late during command chaining.

When the channel fails to accept a byte on an input operation, the following data transferred to storage may be used to fill the gap. On an output operation, overrun indicates that data recorded at the device may be invalid.

All information significant to the use of the device normally is provided in the first byte. Any bit positions following those used for programming information may contain diagnostic information, and the total number of sense bytes for the basic sense command (command code 04) may extend up to 32 bytes as needed. The number and the meaning of the sense bytes extending beyond the first byte are peculiar to the type of I/O device and are specified in the SL publication for the device.

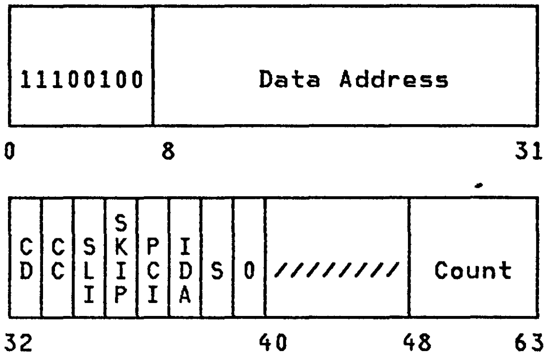
The basic sense command has zero modifier bits. This command initiates a sense operation on all devices and cannot cause the command-reject, intervention-required, data-check, or overrun bit to be set to one. If the control unit detects an equipment malfunction, or invalid parity of the sense command code, the equipment-check or bus-out-check bit is set to one, and unit check is indicated in the unit-status byte.

Devices that can provide special diagnostic sense information or can be instructed to perform other special functions by use of the sense command may define modifier bits for the control of these functions. The special sense operations may be initiated by a unique combination of modifier bits, or a group of codes may specify the same function. Any remaining sense command codes may be considered invalid, thus causing the unit-check indication, or may cause the same action as the basic sense command, depending upon the type of device.

The sense information that pertains to the last I/O operation or other action at a device may be reset any time after the completion of a sense command addressed to that device. Any command addressed to the control unit of a device, other than the no-operation command and the command which results from the TIO function, is allowed to reset the sense information, provided that the busy bit is not included in the initial status. The sense information may also be changed as a result of asynchronous actions, as when the device changes from the not-ready to ready state. (See "Device End" in this chapter.)

A CCW used in a sense operation is inspected for every flag -- CD, CC, SLI, SKIP, S, PCI, and IDA. Bit positions 0-3 of the CCW contain modifier bits.

Sense ID



Execution of the sense-ID command proceeds exactly as that of a read command, except that data is obtained from sensing indicators rather than from a record source. The data source is up to seven bytes in length.

The control unit and I/O device may properly execute the sense-ID command, may execute the command as the basic sense command, or may reject the sense-ID command with unit-check status. Refer to the SL publication for the control unit and I/O device.

The sense-ID command does not initiate any operations other than the sensing of the type/model number. If the addressed unit is available and not busy, then execution of the sense-ID command is accomplished. Basic sense data may be reset as a result of executing the sense-ID command.

Basic sense data may be reset as a result of executing the sense-ID command.

Bytes	Contents
0	FF hex
1,2	Control-unit type number
3	Control-unit model number
4,5	I/O-device type number
6	I/O-device model number

All unused sense bytes are set to zeros.

Bytes 1 and 2 contain the four-decimal-digit control-unit type number that corresponds directly with the control-unit type number attached to the control unit.

Byte 3 contains the control-unit model number, if applicable. If not applicable, byte 3 is a byte of all zeros.

Bytes 4 and 5 contain the four-decimal-digit I/O-device type number that corresponds directly with the I/O-device type number attached to the I/O device.

Byte 6 contains the I/O-device model number, if applicable. If not applicable, byte 6 is a byte of all zeros.

Whenever a control unit is not separately addressable from the attached I/O device or I/O devices, the response to the sense-ID command is a concatenation of the control-unit type number and the I/O-device type number.

If a control unit can be addressed separately from the attached I/O device or I/O devices, then the response to the sense-ID command depends on the unit addressed. If the control unit is addressed, the response to the sense-ID command is as follows:

<u>Bytes</u>	<u>Contents</u>
0	FF hex
1,2	Control-unit type number
3	Control-unit model number

The response consists of the control-unit type and model number, with normal ending status presented after byte 3.

If the I/O device is addressed, the response to the sense-ID command is as follows:

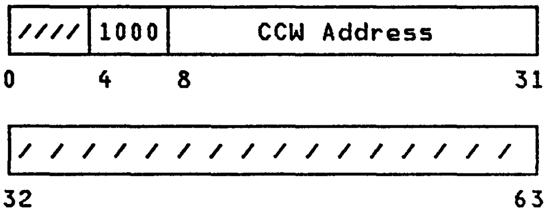
<u>Bytes</u>	<u>Contents</u>
0	FF hex
1,2	I/O-device type number
3	I/O-device model number

The response consists of the I/O-device type and model number, with normal ending status presented after byte 3.

For communication controllers utilizing indirect addressing to end devices, and for cases where the control unit and device are not distinct, the sense data source is the same as if a control unit were being addressed.

A CCW used in a sense ID operation is inspected for every flag -- CD, CC, SLI, SKIP, S, PCI, and IDA.

<u>Transfer in Channel</u>



The next CCW is fetched from the location in absolute storage designated by the data-address field of the CCW specifying transfer in channel. The transfer-in-channel command does not initiate any I/O operation at the channel, and the I/O device is not signaled. The purpose of the transfer-in-channel command is to provide chaining between CCWs not located in successive doubleword locations. The command can occur in both data and command chaining.

The first CCW designated by the CAW must not specify transfer in channel. When this restriction is violated, no I/O operation is initiated, and a program check is generated. The error causes the status portion of the CSW, with the program-check status bit set to one, to be stored during the execution of START I/O or START I/O FAST RELEASE being executed as START I/O. When START I/O FAST RELEASE is executed independent of the device, the error may cause, depending on the model, the same indication as for START I/O or may cause an interruption condition to be generated.

To address a CCW on integral boundaries for doublewords, a CCW specifying transfer in channel must contain zeros in bit positions 29-31. Furthermore, a CCW specifying a transfer in channel must not be fetched from a location designated by an immediately preceding transfer in channel. When either of these errors is detected, a program check is generated.

The contents of the second half of the CCW, bit positions 32-63, are ignored. Similarly, the contents of bit positions 0-3 of the CCW are ignored.

COMMAND RETRY

Some channels have the capability to perform command retry, a channel and control-unit procedure that causes a command to be retried without requiring an I/O interruption. This retry is initiated by the control unit presenting either of two status-bit combinations by means of a special communication sequence with the channel. When immediate retry can be performed, the control unit signals a channel-end, unit-check, and status-modifier status-bit combination, together with device end. When immediate retry cannot be performed, the presentation of device end is delayed until the control unit is prepared. If device end and no other status bits are signaled, command retry is performed. If device end is accompanied by status modifier, command retry is not performed, and the channel command-chains to the CCW following the one for which retry was signaled. When any other status bits accompany device end or device end and status modifier, command retry is suppressed, and the operation is terminated. The resulting CSW contains the status indications that caused command retry to be suppressed.

When the channel is not capable of performing command retry, the retry is suppressed. If command retry is suppressed during the execution of START I/O or START I/O FAST RELEASE executed as START I/O, the CSW is stored, and condition code 1 is set. If command retry is suppressed subsequently, the operation is terminated, and an interruption condition is recognized. The CSW will contain the channel-end, unit-check, and status-modifier status indications, along with any other appropriate status.

During command retry, the channel action is similar to that taken when command chaining. Thus, when command retry is performed, a START I/O initiating an immediate operation for which command chaining is not indicated in the CCW causes a condition code 0, rather than a condition code 1, to be set. The subsequent termination of the I/O operation causes an interruption condition to be generated. During command retry, the CCW may be refetched.

Programming Note

The following possible results of command retry must be anticipated by the program:

1. A CCW with the PCI flag set to one may, if retried because of command retry, cause multiple PCI interruptions to occur.

2. A channel program consisting of a single, unchained CCW specifying an immediate command may cause a condition code 0 rather than a condition code 1 to be set. This setting of the condition code occurs if the control unit signals command retry at the time initial status is signaled to the channel. An interruption condition is generated upon completion of the operation.

3. If a CCW used in an operation is changed before that operation has been successfully completed, the results are unpredictable.

4. A CSW stored after the initiation of a retry but before the presentation of device end, as when an interruption due to the PCI flag occurs, contains the address of the command to be retried plus 8.

5. If a HALT I/O, HALT DEVICE, or CLEAR I/O instruction is issued after the initiation of a retry but before the presentation of device end, the CSW contains the address of the command to be retried plus 8.

6. On a multiplexer channel, chained CCWs which might ordinarily have been executed in a burst may, upon the occurrence of command retry, cause multiplexing to occur, with the result that the channel becomes unexpectedly available.

7. Command chaining may occur even though the CCW does not indicate command chaining. This can occur if command retry is signaled, immediate retry is not requested, and the control unit or device presents status consisting solely of device end and status modifier.

CONCLUSION OF INPUT/OUTPUT OPERATIONS

When the operation or sequence of operations initiated by START I/O or START I/O FAST RELEASE is ended, the channel and the device generate status. Status can be brought to the attention of the program by means of an I/O interruption, by TEST I/O or CLEAR I/O, or, in certain cases, by START I/O or START I/O FAST RELEASE. This status, as well as an address and a count indicating the extent of the operation sequence, are presented to the program in the form of a channel-status word (CSW).

TYPES OF CONCLUSION

Normally an I/O operation at the subchannel lasts until the device signals channel end for a CCW for which command chaining or command retry is not indicated. Channel end can be signaled during the sequence initiating the operation, or later. When the channel detects equipment malfunctioning or an I/O-system reset is performed, the channel disconnects the device without receiving channel end. The program can force a device to be disconnected prematurely by issuing CLEAR CHANNEL, CLEAR I/O, HALT I/O, or HALT DEVICE.

Conclusion at Operation Initiation

After the addressed channel and subchannel have been verified to be in a state where START I/O or START I/O FAST RELEASE can be executed, certain tests are performed on the validity of the information specified by the program and on the availability of the addressed control unit and I/O device. This testing occurs during the execution of START I/O, either during or subsequent to the execution of START I/O FAST RELEASE, and during command chaining and command retry.

A data-transfer operation is initiated at the subchannel and device only when the CCW contains the S flag set to zero, when no programming or equipment errors are detected by the channel, and when the device responds with zero status or signals command retry during the initiation sequence. When the channel detects or the device signals any unusual situations during the initiation of an operation, the command is said to be rejected.

Rejection of the command during the execution of START I/O or START I/O FAST RELEASE is indicated by the setting of the condition code in the PSW. Unless the I/O device is not operational, the reasons for the rejection are detailed by the portion of the CSW stored by START I/O or START I/O FAST RELEASE. The I/O device is not started, no interruption conditions are generated, and the subchannel is available subsequent to the initiation sequence. The I/O device is immediately available for the initiation of another operation, provided the command was not rejected because the device or control unit was busy or not operational.

When an unusual situation causes a command to be rejected during initiation of an I/O operation by command chaining or command retry, an interruption condition is generated, and the subchannel is not available until the condition is cleared. The reasons for the rejection are indicated to the program by means of the corresponding status bits in the CSW. The not-operational state of the I/O device, which during the execution of START I/O and in some cases during the execution of START I/O FAST RELEASE causes condition code 3 to be set, instead causes the interface-control-check bit to be set to one when detected during command chaining or command retry. The new operation at the I/O device is not initiated.

When START I/O FAST RELEASE is executed by a channel independent of the addressed device, tests for most program-specified information, for control-unit and device availability, for control-unit and device status, and for most errors may be performed subsequent to the execution of START I/O FAST RELEASE. Some situations which would have caused a condition code 1 or 3 to be set had the instruction been START I/O instead cause an interruption condition to be generated. The CSW, when stored, indicates that the interruption condition is a deferred condition code 1 or 3.

When START I/O FAST RELEASE is executed and start-I/O-fast queuing is provided for the addressed subchannel, control-unit or device busy indications, when presented in the absence of other indications, may not result in the generation of an interruption condition indicating deferred condition code 1. Instead the I/O operation may remain pending at the subchannel with the subchannel in the working state until the corresponding no-longer-busy indication is presented to the channel by the control unit or device. Subsequently, when the no-longer-busy indication is presented to the channel, the channel again attempts to initiate the pending I/O operation at the device. (See also "START I/O FAST RELEASE" in this chapter.)

When the resume function is performed by the channel, tests for program-specified information, for control-unit and device availability, for control-unit and device status, and for errors are performed as for START I/O FAST RELEASE executed independent of the addressed device. Any unusual or error conditions (except control unit or device busy) detected while attempting to resume channel-program execution at the device causes an interruption condition to be generated. The CSW, when stored, indicates that the interruption condition is a deferred condition code 1 or 3.

Control-unit or device-busy conditions encountered when the resume function is performed by the channel are handled as for START I/O FAST RELEASE when start-I/O-fast queuing is provided. That is, control-unit or device-busy indications may not result in the generation of an interruption condition. Instead, the channel program may remain pending at the subchannel until the no-longer-busy indication is presented by the control unit or device.

Immediate Operations

Any command except that for the TIO function may cause the I/O device to signal channel end immediately upon receipt of the command code. An I/O operation causing channel end to be signaled during the initiation sequence is called an *immediate operation*.

When the first CCW designated by the CAW during a START I/O or START I/O FAST RELEASE executed as a START I/O initiates an immediate operation with command chaining not indicated and command retry not occurring, no interruption condition is generated. In this case, channel end is brought to the attention of the program by causing START I/O or START I/O FAST RELEASE to store the CSW status portion. The subchannel is immediately made available to the program. The I/O operation, however, is initiated, and, if channel end is not accompanied by device end, the device remains busy. Device end, when subsequently provided by the device, causes an interruption condition to be generated.

An immediate operation initiated by the first CCW designated by the CAW during a START I/O FAST RELEASE executed independent of the addressed device appears to the program as a nonimmediate command. That is, any status generated by the device for the immediate command, or for a subsequent command if command chaining occurs, causes an interruption condition to be generated.

When command chaining is specified after an immediate operation and no unusual situations have been detected during the execution, or when command retry occurs for an immediate operation, neither START I/O nor START I/O FAST RELEASE causes the immediate storing of CSW status. The subsequent commands in the chain are handled normally, and channel end for the last operation of the chain of CCWs generates an interruption condition even if the I/O 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 as a result of that operation.

Since a count of zero is not valid, any CCW specifying an immediate operation must contain a nonzero count. When an immediate operation is executed, however, incorrect length is not indicated to the program, and command chaining is performed when so specified.

Programming Note

Control operations for which the entire operation is specified in the command code may be executed as immediate operations. Whether the control function is executed as an immediate operation depends on the operation and type of device and is specified in the SL publication for the device.

Conclusion of Data Transfer

When the device accepts a command, the subchannel is set up for data transfer. The subchannel is in the working state during this period. Unless the channel detects equipment malfunctioning or the operation is concluded by CLEAR CHANNEL, CLEAR I/O, or, on the selector channel, the operation is concluded by CLEAR CHANNEL, CLEAR I/O, HALT I/O, or HALT DEVICE, the subchannel-working state lasts until the channel receives the channel-end signal from the I/O device. When no command chaining or command retry is specified or when chaining is suppressed because of unusual situations, channel end causes the operation at the subchannel to be terminated and an interruption condition to be generated. The status bits in the associated CSW indicate channel end and any unusual situations. The I/O 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 I/O device normally controls the timing of channel end. The duration of data-transfer operations may be variable and may be controlled by the I/O device or the channel.

Excluding I/O-system reset, equipment errors, CLEAR CHANNEL, CLEAR I/O, HALT DEVICE, and HALT I/O, the channel signals the device to conclude data transfer whenever any of the following events occurs:

1. The storage areas specified for the operation are exhausted or filled.

2. A program check is detected.

3. A protection check is detected.

4. A chaining check is detected.

The first event occurs when the channel has stepped the count to zero in the last CCW associated with the operation. A count of zero indicates that the channel has transferred all information specified by the program. The other three events are due to errors and cause premature conclusion of data transfer. In every case, the conclusion is signaled in response to a service request from the device and causes data transfer to cease. If the device has no blocks defined for the operation (such as writing to magnetic tape), it concludes the operation and generates channel end.

The device can control the duration of an operation and the timing of channel end. On certain operations for which blocks are defined (such as reading from magnetic tape), the device does not provide the channel-end signal until the end of the block is reached, regardless of whether or not the device has been previously signaled to conclude data transfer.

If the data address in the CCW is invalid, and the operation is a write or control operation, 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 data address is invalid, the operation is concluded before the recording medium has been advanced. However, since the operation has been initiated at the I/O device, the I/O device generates a channel-end interruption condition. Whether a block at the I/O device is advanced when no data is transferred depends on the type of I/O device and is specified in the SL publication for the I/O device.

When command chaining takes place, the subchannel is in the working state from the time condition code 0 is set for START I/O or START I/O FAST RELEASE until the device signals channel end for the last operation of the chain. On a selector channel or a block-multiplexer channel operating with multiplexing inhibited, the device executing the I/O operation stays connected to the channel and the channel is in the working state during the entire execution of the chain of I/O operations. When multiplexing occurs, an I/O operation in the burst mode causes the channel to be in the working state only while transferring a burst of data. If channel end and device end do not occur concurrently, the device disconnects from the channel after providing channel end, and the channel can in the meantime communicate with other devices.

Any unusual situations cause command chaining to be suppressed and an interruption condition to be generated. The unusual situations can be detected by either the channel or the device, and the device can provide the indications with channel end, control-unit end, or device end. When the channel is aware of the unusual situation by the time the channel-end signal for the operation is received, the chain is ended as if the operation during which the situation occurred were the last operation of the chain. The device-end signal subsequently is processed as an interruption condition. When the device signals unit check or unit exception with control-unit end or device end, the subchannel terminates the working state upon receipt of the signal from the device. The channel-end indication in this case is not made available to the program.

Termination by HALT I/O or HALT DEVICE

The instructions HALT I/O and HALT DEVICE cause the current operation at the addressed channel or subchannel to be immediately terminated. The method of termination differs from that used upon exhaustion of count or upon detection of programming errors to the extent that termination by HALT I/O or HALT DEVICE is not necessarily contingent on the receipt of a service request from the device.

When HALT I/O is issued to a channel operating in burst mode, the channel issues the halt signal to the device currently operating with the channel, regardless of the device address specified by the HALT I/O instruction. If the channel is involved in the data-transfer portion of an operation, data transfer is immediately terminated, and the device is disconnected from the channel. If the channel is 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 channel is 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 and suppresses any further data transfer or command chaining for that device.

When HALT DEVICE is issued to a device for which an I/O operation is pending or suspended at the subchannel, condition code 2 is set as if the channel is operating in burst mode with a different device. Subsequently, when conditions allow, the device is selected, and the halt signal is issued as the device responds. The pending or suspended operation is terminated at the subchannel and an interruption condition is recognized which is not contingent on the receipt of status from the device.

When HALT I/O or HALT DEVICE is issued to a channel not operating in burst mode, then, if the subchannel is not interruption-pending (or, for HALT DEVICE, working with another device), the channel attempts to select the device and issue the halt signal as the device responds. If the device presents status and command chaining is indicated in the subchannel, chaining is suppressed.

The termination of an operation by HALT I/O or HALT DEVICE on the selector channel results in up to four distinct interruption conditions. The first one is generated by the channel upon execution of the instruction and is not contingent on the receipt of status from the device. The channel-status bits reflect the unusual situations, if any, detected during the operation. The execution of HALT I/O or HALT DEVICE itself is not reflected in CSW status, and all status bits in a CSW due to this interruption condition can be zero. The channel is available for the initiation of a new I/O operation as soon as the interruption condition is cleared.

The second interruption condition on the selector channel occurs when the control unit signals channel end. The selector channel handles this condition as any other interruption condition from the device after the device has been disconnected from the channel, and provides zeros in the subchannel-key, CCW-address, count, and channel-status fields of the associated CSW. Channel end is not made available to the program when HALT I/O or HALT DEVICE is issued to a channel executing a chain of operations and the device has already provided channel end for the current operation.

Finally, the third and fourth interruption conditions occur when control-unit end, if any, and device end are signaled. These signals are handled as for any other I/O operation.

The termination of an operation by HALT I/O or HALT DEVICE on a multiplexer channel causes the normal interruption conditions to be generated. If the instruction is issued when the subchannel is in the data-transfer portion of an operation, the subchannel remains in the working state until channel end is signaled by the device, at which time the subchannel is placed in the interruption-pending state. If HALT I/O or HALT DEVICE is issued after the device has signaled channel end and the subchannel is executing a chain of operations, channel end is not made available to the program, and the subchannel remains in the working state until the next status byte from the device is received. Receipt of a status byte subsequently places the subchannel in the interruption-pending state. The CSW associated with the interruption condition in the subchannel contains the status bytes provided by the device and the channel, if any. 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 design of the 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 HALT I/O is issued and the subchannel is in the working state with either a pending or a suspended channel-program execution and the channel is either available or interruption-pending, the addressed device is selected and issued the halt signal. Condition code 1 is set, and the status portion, bits 32-47 of the CSW, are stored to indicate the results of HIO execution. If the addressed device is issued the halt signal, the CSW contains zeros in the status field unless an equipment error is detected. If the channel attempted to select the device but the control unit could not accept the halt signal because of a busy condition, the CSW unit-status field indicates the busy condition.

The termination of a pending or suspended channel-program execution by HALT I/O causes an interruption condition to be recognized. The CSW stored when the interruption occurs contains either zeros or the last status received from the device in the unit-status field and zeros in the channel-status field. The command-address field contains the address of the first CCW, plus 8, or the CCW having the S flag, plus 8, and the deferred condition code is 1.

When a pending or suspended I/O operation is terminated by HALT I/O or HALT DEVICE on any channel, the CSW stored when the interruption condition is cleared contains either the last status received from the device since the I/O operation was made pending or zeros. The command-address field contains the address of the first or suspended CCW plus 8, and the deferred condition code is 1 or 3, depending on whether the device is detected to be operational or not operational, respectively. If the unit status is not zeros, the busy bit is included.

Programming Note

The count field in the CSW associated with an operation terminated by HALT I/O or HALT DEVICE is unpredictable.

Termination by CLEAR I/O

The termination of an operation by CLEAR I/O causes the subchannel to be set to the available state and causes a CSW to be stored. The validity of the CSW fields is defined in the instruction CLEAR I/O earlier in this chapter.

When the CLRIO function terminates an operation at a subchannel in the interruption-pending state, up to three subsequent interruption conditions related to the operation can occur. Since the CLRIO function causes the subchannel to be made available, these interruption conditions will result in only the status portion of the CSW being indicated.

The first interruption condition arises when channel end is signaled to a selector or block-multiplexer channel. This occurs only when the interruption-pending state of the subchannel at the execution of CLEAR I/O is due to the previous execution of HALT I/O or HALT DEVICE.

The second and third interruption conditions arise when control-unit end, if any, and device end are signaled to the channel.

When the CLRIO function terminates an operation at a subchannel in the working state, up to four subsequent interruption conditions related to the operation can occur. For all of these conditions, only the status portion of the CSW is indicated.

The first interruption condition arises on certain channels when the terminated operation was in the midst of data transfer. Since the device is not signaled to terminate the operation during the execution of the CLRIO function unless the channel is working with the addressed device when the instruction is received, the device may, subsequent to execution of the CLRIO function, 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, and an all-zero unit status, is indicated in the CSW.

A second interruption condition may occur if channel-end status is received from the device. The third and fourth conditions may occur if control-unit end, and/or device end are presented to the channel. In these three cases, only unit status is indicated in the CSW, unless an error is detected by the channel.

When a pending I/O operation is terminated by the CLRIO function, the CSW stored contains the address of the first CCW, plus 8, in the command-address field; either zeros, if the channel has not attempted to initiate the operation at the device, or the last status received from the device in the unit-status field; zeros in the channel-status field; and the deferred condition code is 1. If the unit status is not zeros, the busy bit is included.

When CLEAR I/O is issued to a device having a suspended channel-program execution, the suspended channel-program execution is terminated, condition code 1 is set, and a CSW is stored with zeros in the unit-status field and channel-status field. The command-address field contains the address of the CCW having the S flag, plus 8.

Termination by CLEAR CHANNEL

When CLEAR CHANNEL is issued, I/O-system reset is performed in the addressed channel, and system reset is signaled to all I/O devices attached to that channel. I/O-system reset causes the channel to conclude operations in all subchannels. Status information and all interruption conditions in all subchannels are reset, and all operational subchannels are placed in the available state.

Termination Due to Equipment Malfunction

When channel-equipment malfunctioning is detected or invalid signals are received from a device, the recovery procedure and the subsequent states of the subchannels and devices on the channel depend on the type of error and on the model. Normally, the program is alerted to the termination by an I/O interruption condition, and the associated CSW indicates channel-control check or interface-control check. However, when the nature of the malfunction prevents generation of an I/O interruption condition, a machine-check interruption condition is created, and a CSW is not stored. A malfunction may cause the channel to perform I/O selective reset or generate the halt signal.

Signaling of the halt signal, I/O selective reset, or system reset causes channel-program execution, if any, to be terminated at the affected subchannels.

In any termination of a suspended channel-program execution that causes an interruption condition to be recognized, suspension is canceled at the subchannel and the command-address field of the CSW stored when the interruption condition is cleared contains the address of the current (suspended) CCW, plus 8.

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. The conditions are indicated in an associated CSW which is stored at the time of interruption. These conditions can be caused by the program or by an external event at the device.

Interruption Conditions

A request for an I/O interruption is called an I/O-interruption condition, or, in this chapter, simply an interruption condition.

An interruption condition can be brought to the attention of the program only once and is cleared when it causes an interruption. Alternatively, an interruption condition can be cleared by I/O-system reset, I/O selective reset, TEST I/O, or CLEAR I/O, and conditions generated by the I/O device following the termination of an operation at the subchannel can also be cleared by START I/O or START I/O FAST RELEASE. The latter include interruption conditions caused by attention, device end, and control-unit end, and channel end when provided by a device after conclusion of an operation at the subchannel.

The device attempts to initiate a request to the channel for an I/O interruption whenever it detects any of the following:

Channel end
Control-unit end
Device end
Attention

The channel combines the above status with information in the subchannel and either creates an interruption condition, attempts command retry, or continues command chaining as a function of the received status. When command chaining or command retry takes place, channel end and device end do not create an interruption condition and are not made available to the program.

The channel creates an interruption condition when any of the following conditions occurs during command chaining:

Unit check (except when command retry occurs)
Unit exception
Busy indication from device or control unit
Program check
Protection check

When an operation initiated by command chaining is terminated because of an unusual situation detected during the command initiation sequence, the interruption condition may remain pending within the channel, or the channel may create an interruption condition at the device. This interruption condition is created at the device only in response to presentation of status by the device and causes the device subsequently to present the same status for interruption purposes. The interruption condition at the device may or may not be associated with unit status. If the unusual situation is detected by the device (unit check or unit exception) the unit-status field of the associated CSW identifies the condition. If the unusual situation is detected by the channel, as in the case of program and protection check, the identification of the error is preserved in the subchannel and appears in the channel-status field of the associated CSW.

An interruption condition caused by the device may be accompanied by channel and other unit status. Furthermore, more than one condition associated with the same device can be cleared at the same time. As an example, when channel end is not cleared at the device by the time device end is generated, both may be indicated in the CSW and cleared at the device concurrently.

However, either prior to or at the time the channel assigns highest priority for interruptions to an interruption condition associated with an operation at the subchannel, the channel accepts the status from the device and clears the condition at the device. The interruption condition and the associated status indication are subsequently preserved in the subchannel. Any subsequent status generated by the device is not included when the CSW is stored, even if the status is generated before the interruption condition is cleared.

When the channel is not working, a device that is interruption-pending may attempt to initiate a request to the channel for an I/O interruption by presenting a nonzero status byte to the channel. Depending on the channel, some models may accept the status into the subchannel. Alternatively, some models may signal the device to hold the status until the channel is capable of causing an interruption. In this case, the channel selects the device to obtain the status when the interruption occurs. The status stored by the channel is the status presented by the device at interruption time and, because of changed conditions at the device, may not be the same status presented by the device initially. Specifically, a status of zero, busy, or busy and status modifier may be stored.

When the channel detects any of the following, it generates an interruption condition without necessarily communicating with or having received the status byte from the device:

- PCI flag in a CCW
- Termination of a burst operation by 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 device not operational after condition code 0 is set for an SIOF or RIO 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 in this case.

Channel-Available Interruption

The channel-available-interruption (CAI) condition is provided on all block-multiplexer channels and all channels that provide start-I/O-fast queuing for one or more subchannels. The CAI condition causes the entire CSW to be replaced by a new set of bits. All fields of the CSW are set to zero. The I/O address stored contains a zero device address and a channel address identifying the interrupting channel.

A channel which provides the channel-available-interruption condition generates the CAI condition if it previously had responded with a condition code 2 to an I/O instruction other than HALT I/O or HALT DEVICE and if the working state thus indicated no longer exists. When the working state which caused condition code 2 was due to a subchannel busy with a device other than the one addressed, the conclusion of the working state is not signaled by a CAI. Some channels may generate the CAI condition in the following situations:

1. The channel is unable to retrieve status from the I/O device because the I/O device appeared not operational when the channel was allowed to cause an interruption.

2. The channel had previously responded with a condition code 1 to a TEST CHANNEL instruction.

A channel that provides start-I/O-fast queuing also generates the CAI condition in the following situation. If a control-unit-busy condition has been signaled to the program by storing a CSW, either during the execution of START I/O or START I/O FAST RELEASE, or during an I/O interruption subsequent to setting condition code 0 for START I/O FAST RELEASE executed independent of the device, the control unit subsequently generates the control-unit-end condition to signal that the control unit is now available. The control unit may associate the control-unit-end status with any device address that the control unit is capable of recognizing to present the status to the channel. When the device address used by the control unit to present the control-unit-end status (in the absence of any other status indication) is associated with a subchannel that is working and has an I/O operation pending at the subchannel or has a suspended channel-program execution, the subchannel is not made interruption-pending with the control-unit-end status. Instead, the channel recognizes the CAI condition. The control-unit-end status is discarded in this case and the state of the subchannel associated with the device address remains unchanged.

Since any other interruption condition (except PCI) accomplishes the same function as CAI, a CAI condition is reset upon the occurrence of any interruption (except PCI) on that channel. Some channels also reset a CAI condition when another interruption condition (except PCI) is cleared by a TEST I/O or CLEAR I/O on the same channel. The occurrence of another channel-working state before the CAI causes the CAI condition to be suspended until the working state ends.

Programming Note

The CAI can be used as a tool for keeping I/O requests in sequence by using it in conjunction with TEST CHANNEL. 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. A channel which responded with condition code 1 or

2 because the channel was interruption-pending or busy does not subsequently respond with a condition code 0 to a TEST CHANNEL without clearing an interruption condition in the interim.

PRIORITY OF INTERRUPTIONS

Generation of interruption conditions is asynchronous to the activity in the CPU, and interruption conditions associated with more than one I/O device can exist at the same time. The priority among interruption conditions is controlled by two types of mechanisms -- one establishes the priority among interruption conditions within a channel, and another establishes priority among interruption conditions from different channels. A channel requests an I/O interruption only after it has established priority among interruption conditions. The status associated with interruption conditions is preserved in the devices or channels until accepted by the CPU.

Assignment of priority among requests for interruption associated with devices on any one channel is a function of the type of channel, the type of interruption condition, and the method of attaching the device to the channel. A device's priority is not related to its device address. Interruption conditions from different devices do not necessarily occur in the sequence in which they are generated. However, multiple interruption conditions for a single device are presented in the sequence in which they are generated.

The priorities among requests for I/O interruptions from different channels are unpredictable. The priority assignment need not be dependent on the channel address or type.

Interruption Action

An I/O interruption can occur only when the CPU is enabled for I/O interruptions. The interruption occurs at the completion of a unit of operation. If a channel has established the priority among interruption conditions, while the CPU is disabled for I/O interruptions, the interruption occurs immediately after the completion of the instruction enabling the CPU and before the next instruction is executed. This interruption is associated with the highest priority condition for the channel. If interruptions are allowed from more than one channel concurrently, the interruption occurs from the channel having the highest priority among those requesting interruption.

If the priority among interruption conditions has not yet been established in the channel by the time the interruption is allowed, the interruption does not necessarily occur immediately after the completion of the instruction enabling the CPU. This delay can occur regardless of how long the interruption condition has existed in the device or the subchannel.

The interruption causes the current program-status word (PSW) to be stored as the old PSW at real storage location 56 and causes the CSW associated with the interruption to be stored at real storage location 64. In EC mode, the measurement byte is stored at real storage location 185, and the channel and device causing the interruption are identified by the I/O address which is stored at real storage locations 186-187. In BC mode, the channel and device causing the interruption are identified by the I/O address in bit positions 16-31 of the I/O old PSW.

If a limited-channel logout is present, it is stored at real storage locations 176-179.

A new PSW is loaded from real storage location 120. Subsequently, processing resumes in the state indicated by this PSW. The CSW associated with the interruption identifies the interruption condition responsible for the interruption and provides further details about the progress of the operation and the status of the device.

Programming Note

When a control unit which is shared among a number of I/O devices which are concurrently executing operations such as rewinding tape or positioning a disk-access mechanism, the initial device-end signals generated on completion of the operations are provided in the order of generation, unless command chaining is specified for the operation last initiated. In the latter case, the control unit provides the device-end signal for the last initiated operation first, and the other signals are delayed until the subchannel is freed. Whenever interruptions due to the device-end signals are delayed because the CPU is disabled for I/O interruptions or the subchannel is busy, the original order of the signals is destroyed.

CHANNEL-STATUS WORD

The channel-status word (CSW) provides to the program the status of an I/O device or the indication of the reasons for which an I/O operation has been concluded. The CSW is formed, or parts of it are replaced, in the process of I/O interruptions and possibly during the execution of START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, HALT DEVICE, and STORE CHANNEL ID. The CSW is stored at real storage location 64 and is available to the program at this location until the time the next I/O interruption occurs or until another I/O instruction causes its contents to be replaced, whichever occurs first.

The information placed in the CSW by an I/O interruption pertains to the channel and device which are identified by the I/O address stored during the interruption. The information placed in the CSW by START I/O, START I/O FAST RELEASE, TEST I/O, CLEAR I/O, HALT I/O, HALT DEVICE, or STORE CHANNEL ID pertains to the channel and (except for STORE CHANNEL ID) the device addressed by the instruction.

The CSW has the following format:

Key	S	L	CC	CCW Address
0	4		6	8 31

Unit Status	Channel Status	Count
32	40	48 63

The fields in the CSW are allocated as follows:

Subchannel Key: Bits 0-3 form the access key used in the chain of operations at the subchannel.

Suspended (S): Bit 4, when stored as one, indicates that the subchannel associated with the information in the CSW has the execution of a channel program currently suspended. The S condition can only be indicated in the CSW stored as a result of an I/O interruption because of the program-controlled-interruption (PCI) condition.

Logout Pending (L): Bit 5, when one, indicates that an I/O instruction cannot be executed until a logout has been cleared. Bit 45, channel-control check, will always be one when bit 5 is one.

Deferred Condition Code (CC): Bits 6 and 7 indicate whether situations have been encountered subsequent to the setting of a condition code 0 for START I/O FAST RELEASE or RESUME I/O 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 8 higher than the address of the last CCW used.

Status: Bits 32-47 identify the status of the device and the channel that caused the storing of the CSW. Bits 32-39, the unit status, indicate situations detected by the device or control unit. Bits 40-47, the channel status, are provided by the channel and indicate situations associated with the subchannel. The 16 bits are designated as follows:

Bit	Designation
32	Attention
33	Status modifier
34	Control-unit end
35	Busy
36	Channel end
37	Device end
38	Unit check
39	Unit exception
40	Program-controlled interruption
41	Incorrect length
42	Program check
43	Protection check
44	Channel-data check
45	Channel-control check
46	Interface-control check
47	Chaining check

Count: Bits 48-63 form the residual count for the last CCW used.

UNIT STATUS

The following status indications are generated by the I/O device or control unit. The timing and causes of these status indications for each type of device are specified in the SL publication for the device.

When the I/O device is accessible from more than one channel, status due to channel-initiated operations is signaled to the channel that initiated the associated I/O operation. The handling of status not associated with I/O operations, such as attention, unit exception, and device end because of transition from the not-ready to the ready state, depends on the type of device and situation and is specified in the SL publication for the device. (See "Device End" in this chapter.)

Attention

Attention is signaled when the device detects an asynchronous condition that is significant to the program. The condition may also be described by other status indications that accompany attention. Attention is interpreted by the program and is not associated with the initiation, execution, or conclusion of an I/O operation.

The device can signal attention to the channel when no operation is in progress at the I/O device, control unit, or subchannel. Attention can be signaled with device end upon completion of an operation, and it can be signaled to the channel during the initiation of a new I/O operation. An I/O device may present attention accompanied by device end and unit exception when a not-ready-to-ready-state transition is signaled. (See "Device End" in this chapter.) The handling and presentation of attention to the channel depends on the type of device.

When the device signals attention during the initiation of an operation, the operation is not initiated. Attention causes command chaining and command retry to be suppressed.

Status Modifier

Status modifier is generated by the device when the device cannot provide its current status in response to the TIO function, when the control unit is busy, when the normal sequence of commands has to be modified, or when command retry is to be initiated.

When status modifier is signaled in response to the TIO function and status modifier is the only status bit that is set to one, this indicates that the device is unable to execute the TIO function 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 contains zeros in the subchannel-key, CCW-address, and count fields.

When the status-modifier bit in the CSW is set to one together with the busy bit, it indicates that the busy status pertains to the control unit associated with the addressed I/O device. The control unit appears busy when it is executing a type of operation that precludes the acceptance and execution of any command or the instructions TEST I/O, HALT I/O, and HALT DEVICE or, for some control units, 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 execution of the CLRIO function, or, on a selector channel or block-multiplexer channel operating with multiplexing inhibited, due to channel end following the execution of HALT I/O or HALT DEVICE. The busy state occurs for operations such as backspace file, in which case the control unit remains busy after providing channel end, for operations concluded by CLEAR I/O, and for operations concluded on the selector channel by HALT I/O or HALT DEVICE, and temporarily occurs on control units such as the IBM 3705 Communication Controller after initiation of an operation on a device accommodated by the control unit. A control unit accessible from two or more channels may appear busy when it is communicating with another channel.

Presence of status modifier and device end means that the normal sequence of commands must be modified. The handling of this status combination by the channel depends on the operation. If command chaining is specified in the current CCW and no unusual situations have been detected, presence of status modifier and device end causes the channel to fetch and chain to the CCW whose storage address is 16 higher than that of the current CCW. If the I/O device signals status modifier at a time when no command chaining is specified, or when any unusual situations have been detected, no action is taken in the channel, and the status-modifier bit and any other status bits presented by the device are set to ones in the CSW.

Status modifier is set to one in combination with unit check and channel end to initiate the command-retry procedure.

Control units that recognize special conditions that must be brought to the attention of the program present status modifier along with other status indications in order to modify the meaning of the status. The status presented is unrelated to the execution of an I/O operation.

Control-Unit End

Control-unit end indicates that the control unit has become available for use for another operation.

Control-unit end is provided only by control units shared by I/O devices or control units accessible by two or more channels, and only when one or both of the following have occurred:

1. The program had previously caused the control unit to be interrogated while the control unit was in the busy state. The control unit is considered to have been interrogated in the busy state when a command or the instructions START I/O, START I/O FAST RELEASE (when not executed independent of the device), TEST I/O, HALT I/O, or HALT DEVICE had been issued to a device on the control unit, and the control unit had responded with busy and status modifier in the unit-status byte. (See the section "Status Modifier" earlier in this chapter.)

2. The control unit detected an unusual condition during the portion of the operation after channel end had been signaled to the channel. The indication of the unusual situation accompanies control-unit end.

If the control unit remains busy with the execution of an operation after signaling channel end but has not detected any unusual situations and has not been interrogated by the program, control-unit end is not generated. Similarly, control-unit end is not provided when the control unit has been interrogated and could perform the indicated function. The latter case is indicated by the absence of busy and status modifier in the response to the instruction causing the interrogation.

When the busy state of the control unit is temporary, control-unit end is included with busy and status modifier in response to the interrogation even though the control unit has not yet been freed. The busy condition is considered to be temporary if its duration is commensurate with the program time required to handle an I/O interruption. The IBM 3705 Communications Controller is an example of a device in which the control unit may be busy temporarily and which includes control-unit end with busy and status modifier.

Control-unit end can be signaled with channel end, with device end, or between the two. Control-unit end may be signaled at other times and may be accompanied by other status bits. When control-unit end is signaled by means of an I/O interruption in the absence of any other status, the interruption may be identified by any device address assigned to the control unit which is associated with a device in the available state, even if the device is not ready or absent. A control-unit end may cause the control unit to appear busy for the initiation of new operations with any attached device. Alternatively, a control-unit end may be assigned by the control unit to a specific device address, and only that device would appear busy for the initiation of new operations.

When control-unit end is signaled to the channel in the absence of any other status to indicate that the control-unit busy period previously indicated to the program is ended, and the control unit is available, the control-unit-end status normally causes the channel to recognize an interruption condition to present the control-unit end to the program. However, when start-I/O-fast queuing or the suspend-and-resume facility is provided and the device address with which the control unit signals the control-unit end is associated with a working subchannel that has a pending I/O operation or has a suspended channel-program execution, the channel recognizes the channel-available-interruption (CAI) condition instead. The control-unit-end status is discarded by the channel and the state of the associated subchannel remains unchanged in this case. (See the section "Channel-Available Interruption," earlier in this chapter.)

Busy

Busy indicates that the I/O device or control unit cannot execute the command or instruction because (1) it is executing a previously initiated operation, (2) it contains an interruption condition, (3) it is shared by channels or I/O devices and the shared facility is not available, or (4) a self-initiated function is being performed. The status associated with the interruption condition for the addressed device, if any, accompanies the busy status. If busy applies to the control unit, busy is accompanied by status modifier.

The figure "Indications of Busy in CSW" lists the situations for devices connected to only one channel when the busy bit is set to one in the CSW and indicates when busy is accompanied by status modifier. For devices shared by more than one channel, operations related to one channel may cause the control unit or device to appear busy to the other channels.

Condition	CSW Status Stored by				
	SIO, SIOF≠, or RIO≠	TIO	CLRIO+	HIO or HDV	I/O IRPT#
Subchannel available					
DE or attention in device	B,cl	NB,cl%	*	*	NB,cl
Device working, CU available	B	B%	*	*	m
CU end or channel end in CU:					
For the addressed device	B,cl	NB,cl%	*	*	NB,cl
For another device	&	$%	*	*	NB,cl
CU working	B,SM	B,SM%	*	*	B,SM
Interruption condition in subchannel for the addressed device because of:					
Chaining terminated by busy device	*	B,cl	B,cl	*	B,cl
Chaining or retry terminated by busy CU	*	B,SM,cl	B,SM,cl	*	B,SM,cl
Other type of termination	*	NB,cl	NB,cl	*	NB,cl
Asynchronous status∂	B,cl	NB,cl	NB,cl	*	NB,cl=
Subchannel working					
CU available	*	*	NB	NB	*
CU working	*	*	NB	B,SM	*

Explanation:

B Busy bit in CSW is one.

cl Interruption condition cleared; status is placed in CSW.

CU Control unit.

DE Device end.

NB Busy bit in CSW is zero.

SM Status-modifier bit in CSW is one.

* CSW not stored, or I/O interruption cannot occur.

≠ 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 independent of the device and when RESUME I/O is executed, the CSW status is stored by an I/O interruption with the CSW also indicating deferred condition code 1, except when start-I/O-fast queuing is provided for the subchannel. When start-I/O-fast queuing is provided, a control-unit-busy or device-busy condition, in the absence of other status, may not cause an interruption and, instead, the I/O operation remains pending at the subchannel until the no-longer-busy indication is received by the channel.

= When the device presents asynchronous status other than control-unit end while a channel program is suspended at the subchannel, the channel program is terminated, and an interruption condition is generated; the status, with the busy bit included, is stored in the CSW when the interruption occurs, along with the deferred condition code equal to zero and the command address equal to the address of the suspended CCW + 8.

Indications of Busy in CSW (Part 1 of 2)

Explanation (Continued):

& Either a CSW is not stored or busy and status modifier are stored.

$ Unit status of either zeros or busy and status modifier is stored.

m Unit status of busy may be stored, or an I/O interruption may not occur.

@ Asynchronous status is any unit status that is not related to the termination of an I/O operation at the subchannel.

# Except when the I/O interruption is caused by a deferred condition code 1 for START I/O FAST RELEASE.

+ The entries in this column apply only when the CLRIO function is executed. When CLEAR I/O causes the TIO function to be executed, the entries in the TIO column apply.

% When the control unit is the type that never supplies status to the TIO function, unit status consisting solely of status modifier is stored, and no interruption conditions are cleared.

Indications of Busy in CSW (Part 2 of 2)

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 control unit no longer requires channel facilities to perform the operation.

Each I/O operation initiated at the device causes channel end to be signaled, and there is only one channel end for an operation. Channel end is not signaled when programming errors or equipment malfunctions are detected during initiation of the operation. When command chaining takes place, only the channel end of the last operation of the chain is made available to the program. Channel end is not made available to the program when a chain of commands is prematurely concluded because of an unusual situation indicated with control-unit end or device end, or during the initiation of a chained or retried command.

The instant within an I/O operation when channel end is signaled depends on the operation and the type of device. For operations such as writing on magnetic tape, channel end occurs when the block has been written. On devices that verify the writing, channel end may or may not be delayed until verification is performed, depending on the device. When magnetic tape is being read, channel end occurs when the next interblock gap on tape reaches the read-write head. On devices equipped with buffers, channel end occurs upon completion of data transfer between the channel and the buffer. During control operations, channel end is generated when the control information has been transferred to the devices, although for short operations channel end may be delayed until completion of the operation. Operations that do not cause any data to be transferred can provide channel end during the initiation sequence.

Channel end in the control unit may cause the control unit to appear busy for the initiation of new operations.

Channel end is presented in combination with status modifier and unit check to initiate the command-retry procedure.

Device End

Device end is indicated (1) when the completion of an I/O operation occurs at the device, (2) when the I/O device signals that a change from the not-ready to the ready state has occurred, (3) when the termination of an activity has occurred which previously caused a response of busy to the channel, and (4) when the I/O device signals that an asynchronous condition has been recognized. Device end normally indicates that the I/O device has become available for use in another operation.

Each I/O operation initiated at the device causes device end, and there is only one device end for an operation. Device end is not generated when any programming or equipment malfunction is detected during initiation of the operation. When command chaining takes place, only the device end of the last operation of the chain is made available to the program unless an unusual condition is detected during the initiation of a chained or retried command, in which case the chain is concluded without device end.

Device end associated with an I/O operation is generated either simultaneously with channel end or later. For data-transfer operations on some I/O devices, the operation is complete at the time channel end is generated, and both device end and channel end occur together. The time at which device end is presented depends upon the I/O-device type and the kind of command executed. For most I/O devices, device end is presented when the I/O operation is completed at the I/O device. In some cases, for reasons of performance, device end is presented before the I/O operation has actually been completed at the I/O device. However, in all cases, when device end is presented, the I/O device is available for execution of an immediately following CCW if command chaining was specified in the previous CCW. During execution of control commands, device end may be presented with channel end or later.

When command chaining is specified, receipt of the device-end signal, in the absence of any unusual situations, causes the channel to initiate a new I/O operation.

When the state of a device is changed from not ready to ready, either device end or device end, attention, and unit exception are indicated. Refer to the SL publication for the I/O device to determine which indication is given.

A device is considered to be not-ready when operator intervention is required in order to make the device ready. A not-ready condition can occur, for example, because of any of the following:

1. An unloaded condition for magnetic tape

2. Card equipment out of cards or with the stacker full

3. A printer out of paper

4. Error conditions that need operator intervention

5. The unit having changed from the enabled to the disabled state

Device end is also accompanied by other status where conditions are recognized that are unrelated to the execution of an I/O operation.

Unit Check

Unit check indicates that the I/O device or control unit has detected an unusual situation that is detailed by the information available to a sense command. Unit check may indicate that a programming or equipment error has been detected, that the not-ready state of the device has affected the execution of the command or instruction, or that an exceptional situation other than the one identified by unit exception has occurred. The unit-check bit provides a summary indication of the sense data.

An error causes the unit-check indication when it occurs during the execution of a command or the TIO function or during some activity associated with an I/O operation. Unless the error pertains to the activity initiated by a command or is of immediate significance to the program, the error does not cause the program to be alerted after device end has been cleared; a malfunction may, however, cause the device to become not ready.

Unit check is indicated when the existence of the not-ready state precludes a satisfactory execution of the command, or when the command, by its nature, tests the state of the device. When no interruption condition is pending for the addressed device at the control unit, the control unit signals unit check when the TIO function 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 the rewind-and-unload command for magnetic tape, unit check is not indicated if the command is properly executed even though the device has become not ready during or as a result of the operation. Similarly, unit check is not indicated if the command can be executed with the device not ready. Selection of a device that is not ready does not cause a unit check when the sense command is issued or when an interruption condition is pending for the addressed device at the control unit.

If the device detects during the initiation sequence that the command cannot be executed, unit check is signaled to the channel without channel end, control-unit end, or device end. Such unit status indicates that no action has been taken at the device in response to the command. If the situation precluding proper execution of the operation occurs after execution has been started, unit check is accompanied by channel end, control-unit end, or device end, depending on when the situation was detected. Any errors detected after device end has been cleared are indicated by signaling unit check with attention, unit check with control-unit end, or unit check with device end.

Errors, such as invalid command code or invalid command-code parity, do not cause unit check when the device is working or contains an interruption condition at the time of selection. Under these circumstances, the device responds by providing busy status and indicating the interruption condition, if any. The command-code invalidity is not indicated.

Concluding an operation with the unit-check indication causes command chaining to be suppressed.

Unit check is presented in combination with channel end and status modifier to initiate the command-retry procedure.

Programming Notes

1. If a device becomes not ready upon completion of a command, the ending interruption condition can be cleared by the TIO function without generation of unit check due to the not-ready state, but any subsequent TIO function issued to the device causes a unit-check indication.

2. In order that sense indications set in conjunction with unit check are preserved by the device until requested by a sense command, some devices inhibit certain functions until a command other than the TIO function or no-operation is received. Furthermore, any command other than sense, the TIO function, or no-operation may cause the device to reset any sense information. Similarly, when start-I/O-fast queuing is provided, initiation of I/O operations pending at the time the unit check is received may be inhibited for other devices attached to the same control unit. The initiation of the pending operations is inhibited until a subsequent I/O operation (usually a sense operation) is successfully initiated at the device that presented the unit check. 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.

3. Unit-check status presented either in the absence of or accompanied by other status indicates only that sense information is available to the basic sense command. Presentation of either channel end and unit check or channel end, device end, and unit check does not provide any indication as to the kind of conditions encountered by the control unit, the state of the I/O device, or whether execution of the I/O operation ever was initiated. Descriptions of these conditions or states are provided in the sense information.

Unit Exception

Unit exception is caused when the I/O device detects a situation that usually does not occur. Unit exception includes situations such as recognition of a tape mark and does not necessarily indicate an error. During execution of an I/O operation, unit exception has only one meaning for any particular command and type of device.

Unit exception may be generated 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 immediate significance to the program. If the device detects during the initiation sequence that the operation cannot be executed, unit exception is presented to the channel and appears without channel end, control-unit end, or device end. Such unit status indicates that no action has been taken at the device in response to the command. If the condition precluding normal execution of the operation occurs after the I/O operation has been initiated, unit exception is accompanied by channel end, control-unit end, or device end, depending on when the situation was detected. Any unusual condition associated with an I/O operation, but detected after device end has been cleared, is indicated by signaling unit exception with attention.

If the I/O device responds with busy status to a command, the generation of unit exception is suppressed even when execution of that command usually causes unit exception to be indicated.

Concluding an operation with the unit-exception indication causes command chaining and command retry to be suppressed.

Some devices present unit exception accompanied by device end and attention whenever a device changes from the not-ready state to the ready state. (See "Device End" in this chapter.)

CHANNEL STATUS

The following status bits are generated by the channel. Except for the status bits resulting from equipment malfunction, they can occur only while the subchannel is involved with the execution of an I/O operation.

Program-Controlled Interruption

A program-controlled interruption occurs when the channel fetches a CCW with the program-controlled-interruption (PCI) flag set to one. The I/O interruption due to the PCI flag takes place as soon as possible after the CCW takes control of the operation, unless the CCW also contains the S flag set to one, but may be delayed an unpredictable amount of time because I/O interruptions are disallowed or because of other activity in the system. When the CCW also contains a valid S flag, the PCI condition is not generated until after channel-program execution is suspended.

The interruption condition due to the PCI flag does not affect the progress of the I/O operation.

Incorrect Length

Incorrect length occurs when the number of bytes contained in the storage areas assigned for the I/O operation is not equal to the number of bytes requested or offered by the I/O device. Incorrect length is indicated for one of the following reasons:

Long Block on Input: During a read, read-backward, or sense operation, the device attempted to transfer one or more bytes to storage after the assigned storage areas were filled. The extra bytes have not been placed in storage. The count in the CSW is zero.

Long Block on Output: During a write or control operation, the device requested one or more bytes from the channel after the assigned storage areas were exhausted. The count in the CSW is zero.

Short Block on Input: The number of bytes transferred during a read, read-backward, or sense operation is insufficient to fill the storage areas assigned to the operation. The count in the CSW is not zero.

Short Block on Output: The device terminated a write or control operation before all information contained in the assigned storage areas was transferred to the device. The count in the CSW is not zero.

Incorrect length is not indicated when the current CCW has the SLI flag set to one and the CD flag set to zero. The indication does not occur for immediate operations and for operations rejected during the initiation sequence.

When incorrect length occurs, command chaining is suppressed, unless the SLI flag in the CCW is one or unless the operation is immediate. See the figure "Channel-Chaining Action" in this chapter for the effect of the CD, CC, and SLI flags on the indication of incorrect length.

Programming Note

The setting of incorrect length is unpredictable in the CSW stored during | CLEAR I/O, HALT I/O, or HALT DEVICE if | the subchannel was in the working state.

Program Check

Program check occurs when programming errors are detected by the channel. Program check can be due to the following causes:

Invalid CCW-Address Specification: The CAW or the transfer-in-channel command does not designate the CCW on a doubleword boundary.

Invalid CCW Address: The channel has attempted to fetch a CCW from a 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 has attempted 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 zeros in bit positions 4-7. 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 contents of the data-address field in the CCW do not designate the first IDAW on an integral word boundary.

Invalid IDAW Address: The channel has attempted to fetch an IDAW from a 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, has attempted to fetch from an unavailable location.

Invalid Data Address: The channel has attempted to transfer data to or from a 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, has attempted to access an unavailable location.

Invalid IDAW Specification: The 24-bit-IDAW facility is installed and bits 0-7 of the IDAW are not all zeros, or the second or subsequent IDAW does not specify the first or, for read-backward operations, the last byte of a 2K-byte storage block. The 31-bit IDAW facility is installed and bit 0 of the IDAW is not zero, or the second or subsequent IDAW does not specify the first or, for read-backward operations, the last byte of a 2K-byte storage block.

Invalid CAW Format: The CAW does not contain zeros in bit positions 4-7 when the suspend-and-resume facility is not provided by the system model or in bit positions 5-7 when the suspend-and-resume facility is provided.

Invalid CCW Format: A CCW other than a CCW specifying transfer in channel does not contain zeros in bit positions 38-39 when the suspend function is not provided for the subchannel or does not contain zero in bit position 39 when the suspend function is provided.

Invalid Suspend Flag: A CCW fetched during data chaining, other than a CCW specifying transfer in channel, does not contain a zero in bit position 38. A CCW other than a CCW specifying transfer in channel does not contain a zero in bit position 38 and either suspend control was not specified in the CAW, or the suspend function is not operable for the subchannel.

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, or a sequence of 256 or more CCWs with command chaining specified were executed by the channel and did not result in the transfer of any data with an I/O device.

Detection of program check during the initiation of an operation causes execution of the operation to be suppressed. When program check is detected after the operation has been initiated at the device, the device is signaled to conclude the operation the next time it requests or offers a byte of data. Program check causes command chaining and command retry to be suppressed.

Protection Check

Protection check occurs when the channel attempts a storage access that is prohibited by key-controlled storage protection. Protection applies to the fetching of CCWs, IDAWs, and output data, and to the storing of input data. Storage accesses associated with each channel program are performed using the subchannel key provided in the CAW associated with that channel program. For details, see the section "Key-Controlled Protection" in Chapter 3, "Storage."

When protection check 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 operation has been initiated at the device, the device is signaled to conclude the operation the next time it requests or offers a byte of data. Protection check causes command chaining and command retry to be suppressed.

Channel-Data Check

Channel-data check indicates that a machine error has been detected in the information transferred to or from storage during an I/O operation, or that an error has been detected on data transferred from the device during an input operation. This information includes the data read or written, as well as the information transferred as data during a sense or control operation. The error may have been detected in the channel, in storage, or on the path between the two. Channel-data check may be indicated for data with an invalid checking-block code in storage when the data is referred to by the channel but the data does not participate in the operation. This can happen, for example, on an input operation when less than a full checking block of data is to be placed in storage. In this case, called a partial store, the entire checking block is fetched from storage, is updated with the input data, and is replaced in storage. If a CBC error is detected when the checking block is fetched, it cannot be corrected because only part of the checking block is updated during a partial store. In this situation, a channel-data check condition is recognized because of a CBC error in data referred to (the original contents of the checking block) and not because of an error in the input data itself.

Whenever an error on input data is indicated by means of channel-data check, the channel forces correct parity on all data received from the I/O device, and all data placed in storage has valid checking-block code. When, on an input operation, the channel attempts to store less than a complete checking block, and when invalid checking-block code is detected on the checking block in storage, the contents of the location remain unchanged with invalid checking-block code. On an output operation, whenever a channel-data check is indicated, all bytes that came from a checking block with invalid checking-block code have been transmitted with parity errors.

Channel-data check causes command chaining and command retry to be suppressed but does not affect the execution of the current operation. Data transfer proceeds to normal completion, if possible, and an interruption condition is generated when the device presents channel end. A logout may be performed, depending on the channel. Accordingly, the detection of the error may affect the state of the channel and the device.

Channel-Control Check

Channel-control check is caused by machine malfunction affecting channel controls. It may be caused by invalid checking-block codes on CCW addresses, data addresses, and the contents of the CCW. Channel-control check may also include those channel-detected errors associated with data transfer that are not indicated as channel-data check, as well as those communication errors detected by the channel that are not indicated as interface-control check. Errors responsible for channel-control check may cause the contents of the CSW to be invalid and conflicting. The CSW as generated by the channel has valid checking-block code.

Detection of channel-control check causes the current operation, if any, to be immediately concluded.

Channel-control check is set whenever CSW bit 5, logout pending, is set to one.

In some situations, machine malfunctions affecting channel control may instead be reported as an external-damage or system-damage machine-check condition.

Interface-Control Check

Interface-control check indicates that an invalid signal has been received by the channel when communicating with a control unit or device. This check is detected by the channel and usually indicates malfunctioning of an I/O device. It can be due to the following:

1. The device address or status byte received from a device has invalid parity.

2. A device responded with a device address other than the device address specified by the channel during initiation of an operation.

3. During command chaining or command retry 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.

The interface-control-check condition may also include those channel-detected errors associated with data transferred from the device that are not indicated as channel-data check.

Detection of interface-control check causes the current operation, if any, to be immediately concluded.

Chaining Check

Chaining check is caused by channel overrun during data chaining on input operations. Chaining check occurs when the I/O data rate is too high to be handled by the channel and by storage under current conditions. Chaining check cannot occur on output operations.

Chaining check causes the I/O device to be signaled to conclude the operation. It causes command chaining and command retry to be suppressed.

CONTENTS OF CHANNEL-STATUS WORD

The contents of the CSW depend on the reason the CSW was stored and on the programming method by which the information is obtained. The deferred-condition-code field and the status portion identify the reason the CSW was stored. The subchannel-key, suspended-indication, logout-pending, deferred-condition-code, CCW-address, and count fields may contain information pertaining to the last operation or may be set to zero, or the original contents of these fields at real locations 64-67 and 70-71 may be left unchanged.

Information Provided by Channel-Status Word

Interruption conditions resulting from the execution or conclusion of an operation at the subchannel cause the whole CSW to be replaced. Such a CSW can be stored only by an I/O interruption or by TEST I/O or CLEAR I/O. Except for situations associated with command chaining and equipment malfunctioning, the storing can be caused by PCI or channel end and by the execution of HALT I/O or HALT DEVICE on the selector channel. The contents of the CSW are related to the current values of the corresponding quantities, although the count is unpredictable after program check, protection check, and chaining check, and after an interruption due to HALT I/O, HALT DEVICE, the CLRIO function, or the PCI flag.

A CSW stored upon the execution of a chain of operations pertains to the last operation which the channel executed or attempted to initiate. Information concerning the preceding operations is not preserved and is not made available to the program.

When an unusual situation causes command chaining to be suppressed, the premature conclusion of the chain is not explicitly indicated in the CSW. A CSW associated with a conclusion due to a situation occurring at channel-end time contains channel end and identifies the unusual situation. When the device signals the unusual situation with control-unit end or device end, the channel-end indication is not made available to the program, and the channel provides the current subchannel key, CCW address, and count, as well as the unusual indication, with control-unit end or device end in the CSW. The CCW-address and count fields pertain to the operation that was executed.

When the execution of a chain of commands is concluded by an unusual situation detected during initiation of a new operation, the CCW-address and count fields pertain to the rejected command. Except for situations resulting from equipment malfunctioning, conclusion at initiation time can occur because of attention, unit check, unit exception, busy, protection check, or program check, and causes both the channel-end and device-end bits in the CSW to be set to zeros.

A CSW associated with status signaled after the operation at the subchannel has been concluded contains zeros in the subchannel-key, CCW-address, and count fields, provided the status is not cleared during START I/O or START I/O FAST RELEASE. This status includes attention, control-unit end, and device end (and channel end when it occurs after the conclusion of an operation on the selector channel by HALT I/O or HALT DEVICE).

When the above status indications are cleared during START I/O or START I/O FAST RELEASE, only the status portion of the CSW is stored, and the original contents of the subchannel-key, CCW-address, deferred-condition-code, logout-pending, and count fields at locations 64-67 and 70-71 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.

The CSW stored when a channel-available interruption occurs contains zeros in all fields.

Errors detected during execution of the I/O operation do not affect the validity of the CSW unless channel-control check or interface-control check are indicated. Channel-control check indicates that equipment errors have been detected which can cause any part of the CSW, as well as the I/O address, to be invalid. Interface-control check indicates that the address identifying the device or the status bits received from the device may be invalid. The channel forces correct parity on invalid CSW fields. The validity of these fields can be ascertained by inspecting the limited channel logout.

When any I/O instruction cannot be executed because of a pending logout which affects the operational capability of the channel or subchannel, a full CSW is stored. The fields in the CSW are all set to zeros, with the exception of the logout-pending bit and the channel-control-check bit, which are set to ones.

Subchannel Key

A CSW stored to reflect the progress of an operation at the subchannel contains the subchannel key used in that operation. The contents of this field are not affected by programming errors detected by the channel or by the situations causing termination of the operation.

Suspended Indication

When the CSW is stored during an interruption because of the program-controlled-interruption (PCI) condition, bit 4 of the CSW indicates whether the channel-program execution is currently suspended. Suspension of channel-program execution is a function of the suspend-and-resume facility that may be provided for one or more subchannels of multiplexer channels, depending on the system model.

A channel-program execution is considered to be suspended from the time the channel performs the suspend function because of the presence of a valid S flag in a CCW until that channel-program execution is terminated at the subchannel or until the resume function is performed because of a successful (condition code 0) RIO issued to the subchannel. During the period of suspension, the storing of a CSW can only occur as a result of the PCI condition. The PCI condition may be generated because of a PCI flag in the CCW containing the S flag or because of a PCI flag in a CCW fetched earlier in the chain of commands being executed at the subchannel. When the PCI flag and a valid S flag are in the same CCW, the resulting CSW contains the suspended indication unless the CSW indicates that channel-program execution is terminated at the subchannel.

Logout Pending

The logout-pending bit can be stored as one only in a CSW stored during the execution of an I/O instruction. The I/O instructions that can result in storing the CSW with the logout-pending indication are CLEAR I/O, HALT DEVICE, HALT I/O, START I/O, START I/O FAST RELEASE, STORE CHANNEL ID, and TEST I/O. When the CSW is stored and indicates logout pending, channel-control check is also indicated in the channel-status field.

Deferred Condition Code

In the case of START I/O FAST RELEASE executed independent of the device or RESUME I/O issued to a suspended subchannel, initiation or resumption of the I/O operation is not completed during the execution of the instruction. If no conditions are encountered during the execution of the instruction that preclude the acceptance of the function of the instruction by the channel, condition code 0 is set, and conditions encountered subsequent to executing the instruction which preclude the completion of the specified function cause the deferred condition code to be set. The deferred condition code is set when a CSW is stored because of an interruption condition signaling the conclusion of the I/O operation at the subchannel.

Deferred condition code 1 is set either when the channel has detected a condition that would have caused condition code 1 to be set in response to the START I/O FAST RELEASE instruction if the SIO function had been performed, or when HALT I/O, HALT DEVICE, CLEAR I/O or equipment malfunction causes the channel to terminate the I/O operation while it is pending at the subchannel. When HALT I/O, HALT DEVICE, or equipment malfunction terminates a pending I/O operation, deferred condition code 1 is set in the CSW that is stored during the I/O interruption signaling the termination.

Deferred condition code 1 is also set when the channel detects a condition while attempting to resume a suspended channel-program execution that would have caused deferred condition code 1, had the SIOF function been executed independent of the device with the subchannel available, instead of RESUME I/O with the subchannel suspended.

Deferred condition code 1 is also set when, after HALT DEVICE is issued to a suspended subchannel, the device has been selected and an attempt made to issue the halt signal.

Deferred condition code 3 is set when the channel has detected that the addressed device is not operational even though condition code 0 was set in response to the START I/O FAST RELEASE or RESUME I/O instruction, or when, after HDV is issued to a suspended subchannel, the device is found to be not operational when the attempt is made to issue the halt signal. When the CSW contains deferred condition code 3, the unit-status field contains zeros and has no meaning with respect to the progress of the I/O operation.

The figure "Contents of the Deferred-Condition-Code Field" summarizes the handling of deferred condition codes. The figure lists the states and activities that can cause deferred-condition-code indications to be created and the methods by which these indications can be placed in the CSW.

Deferred Condition Code	When I/O Is Idle	When Subch Is Working	Upon Termination of Operation at			After SIOF@ or RIO or during Command Retry or Chaining	When SIO or SIOF& Is Executed	When TIO Is Executed	When CLRIO Is Executed +	When HIO or HDV Is Executed	When I/O Interruption Occurs
			Subch	Ctrl Unit	I/O Dev						
Deferred condition code 1						C•		S	CS≠		S#
Deferred* condition code 3						C•		S	S		S

<u>Explanation</u>:

C The channel can create a deferred-condition-code indication as a result of and subsequent to the execution of the designated instruction. In the case of CLRIO, the indication is created at the time the instruction is executed. The indication is not created as a result of the SIO instruction. In all other cases, the creation of the indication generates an interruption condition.

S The deferred-condition-code indication is stored in the CSW at the designated time.

* When the CSW is stored, it contains zero unit status.

≠ The deferred condition code that is indicated in the CSW can also be the result of CLRIO terminating a pending I/O operation that was initiated by means of SIOF executed independent of the device, or by terminating a suspended I/O operation.

The deferred condition code that is indicated in the CSW can also be the result of HDV or HIO terminating a pending I/O operation that was initiated by means of SIOF executed independent of the device, or by terminating a suspended I/O operation.

+ The entries in this column apply only when the CLRIO function is executed. When CLEAR I/O causes the TIO function to be executed, the entries in the TIO column apply.

& When executed as SIO

@ When executed independent of the device

• Applies only to RIO or SIOF executed independent of the device

<u>Note</u>: The absence of an entry indicates that no deferred condition code is created or stored.

Contents of the Deferred-Condition-Code Field

<u>CCW Address</u>

When the CSW is formed to reflect the progress of the I/O operation at the subchannel, the CCW address is normally 8 higher than the address of the last CCW used in the operation.

The figure "Contents of the CCW-Address Field in the CSW" lists the contents of the CCW-address field for all situations that can cause the CSW to be stored. They are listed in order of priority; that is, if two situations occur, the CSW appears as indicated for the situation higher on the list. When a CSW has been stored and the situation exists that a command-retry request has been recognized but the CCW has not been re-executed, the "last-used CCW + 8" is the CCW that is to be retried.

Situations	Contents of Field
I/O instruction issued when channel logout-pending	Zero
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 spec 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, CCW format, IDAW-address specification, or count	Address of invalid CCW + 8
Invalid data address, invalid IDAW address, or IDAW specification	Address of current CCW + 8
Invalid sequence - 2 TICs	Address of second TIC + 8
Invalid key on CCW fetch	Address of protected CCW + 8
Invalid key on data or IDAW access	Address of current 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 or HALT DEVICE	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, attention, or unit exception with device end, channel end, or control-unit end	Address of last CCW used in the completed operation + 8
Termination on command chaining by busy, attention, unit check, or unit exception	Address of CCW specifying the new operation + 8
Deferred condition code 1 or 3	Address of CCW specifying the new or suspended operation + 8
PCI flag in CCW	Address of CCW that contained the last recognized PCI flag + 8, or address of CCW which has subsequently become current + 8. When the suspended bit (bit 4) of the CSW is stored as one, the address stored is the address of the CCW containing the S flag + 8.
Interface control check	Unpredictable
Channel end after HALT I/O or HALT DEVICE on selector channel (and, depending on design of the subchannel, on block-multiplexer channel)	Zero
Channel end after CLEAR I/O	Zero
Control-unit end	Zero
Device end	Zero
Attention	Zero
Busy	Zero
Status modifier	Zero
Channel-available interruption	Zero

Contents of the CCW-Address Field in the CSW

Count

The residual count, in conjunction with the original count specified in the last CCW used, indicates the number of bytes transferred to or from the area designated by the CCW. When an input operation is concluded, the difference between the original count in the CCW and the residual count in the CSW is equal to the number of bytes transferred to storage; on an output operation, the difference is equal to the number of bytes transferred to the I/O device.

The figure "Contents of the Count Field in the CSW" lists the contents of the count field for all situations that can cause the CSW to be stored. They are listed in the order of priority; that is, if two situations occur, the CSW appears as for the situation higher on the list.

Situations	Contents of Field
I/O instruction issued when channel logout-pending	Zero
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, attention, or unit exception with device end, channel end, or control-unit end	Correct. Residual count of last CCW used in the completed operation.
Termination on command chaining by busy, attention, unit check, or unit exception	Correct. Original count of CCW specifying the new operation.
Deferred condition code 1 or 3	Unpredictable
PCI flag in CCW	Unpredictable
Interface-control check	Unpredictable
Channel end after HALT I/O or HALT DEVICE on selector channel (and, depending on design of the subchannel, on block-multiplexer channel)	Zero
Channel end after CLEAR I/O	Zero
Control-unit end	Zero
Device end	Zero
Attention	Zero
Busy	Zero
Status modifier	Zero
Channel-available interruption	Zero

Contents of the Count Field in the CSW

Status

The status bits identify the situations that have been detected during the I/O operation, that have caused a command to be rejected, or that have been generated by external events.

When the channel detects several errors, all corresponding status bits in the CSW may be set to ones or only one may be set, depending on the error and model. Errors associated with equipment malfunctioning have precedence, and whenever malfunctioning causes an operation to be terminated, channel-control check, interface-control check, or channel-data check is indicated, depending on the error. When an operation is concluded by program check, protection check, or chaining check, the channel identifies the situation responsible for the conclusion and may or may not indicate incorrect length. When a data error has been detected and the operation is concluded prematurely because of a program check, protection check, or chaining check, both channel-data check and the other error are identified.

If the CCW fetched on command chaining has the PCI flag set to one but a programming error in the contents of the CCW precludes the initiation of the operation, it is unpredictable whether the PCI bit is one in the CSW associated with the interruption condition.

However, if the CCW fetched on command chaining has the PCI flag set to one but an unusual situation detected by the device precludes the initiation of the operation, the PCI bit is one in the CSW associated with the interruption condition. Similarly, the PCI bit is unpredictable in a CSW stored by START I/O or START I/O FAST RELEASE or in a CSW that has a nonzero deferred condition code.

Situations detected by the channel are not related to those identified by the I/O device.

The figure "Contents of the CSW Status Fields" summarizes the handling of status bits. The figure lists the states and activities that can cause status indications to be created and the methods by which these indications can be placed in the CSW.

Status	When I/O Is Idle	When Subch Is Working with Device	Upon Termination of Operation at			After SIOF∂ or RIO or During Command Retry or Chaining	When SIO or SIOF& Is Executed	When TIO Is Executed	When CLRIO Is Executed +	When HIO or HDV Is Executed	When I/O Interruption Occurs
			Subch	Ctrl Unit	I/O Dev						
Attention	C*				C*	C*	S	S	S		S
Status modifier			C	C	C	C	CS	CS	S	CS	S
Control-unit end				C*		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	C*	CS	S	S		S
Program-controlled-interruption		C*	C*			C*	CS	S	S		S
Incorrect length		C	C					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

<u>Explanation</u>:

C The channel or device can create or present status at the indicated time. A CSW or its status portion is not necessarily stored at this time.

Status such as channel end or device end is created at the indicated time. Other status bits may have been created previously but are made accessible to the program only at the indicated time. Examples of such status bits are program check and channel-data check, which are detected while data is transferred but are made available to the program only with channel end, unless the PCI flag or an equipment malfunction has cause an interruption condition to be generated earlier.

S The status indication is stored in the CSW at the indicated time.

An "S" appearing alone indicates that the status has been created previously. The letter "C" appearing with the "S" indicates that the status did not necessarily exist previously in the form that causes the program to be alerted, and may have been created by the I/O instruction or I/O interruption. For example, an equipment malfunction may be detected during an I/O interruption, causing channel-control or interface-control check to be indicated; or a device such as the IBM 3705 may signal temporary control unit busy in response to interrogation by an I/O instruction, causing status modifier, busy, and control-unit end to be indicated in the CSW.

* The status generates an interruption condition.

Channel end and device end do not result in interruption conditions when command chaining is specified or command retry is signaled, and no other unusual situations have been detected. Unit check does not result in an interruption condition when command retry is signaled and is honored by the channel.

Contents of the CSW Status Fields (Part 1 of 2)

Explanation (Continued):

≠ This indication is created at the indicated time only by an immediate operation.

H When an operation on the selector channel has been concluded by HALT DEVICE or HALT I/O, or an operation has been concluded by CLEAR I/O, channel end indicates the conclusion of the data-handling portion of the operation at the control unit.

& When executed as SIO.

@ When executed independent of the device.

+ The entries in this column apply only when the CLRIO function is executed. When CLEAR I/O causes the TIO function to be executed, the entries in the TIO column apply.

◊ Channel-control-check status may also be generated, and is then stored in the CSW, when the STIDC function is executed.

Contents of the CSW Status Fields (Part 2 of 2)

CHANNEL LOGOUT

When a channel stores a CSW that indicates channel-control check in the absence of logout pending, or interface-control check, or, on some channels, channel-data check, a channel logout accompanies the storing of the CSW. Such a logout is useful for error recovery. The logout may be a limited channel logout, a full channel logout, or both. The type of logout that occurs and, for the full channel logout, the length of the full channel logout and the location at which it is stored, depend on the channel type and model.

The limited channel logout contains model-independent information and is stored at real locations 176-179 of the CPU to which the channel is connected. 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) address in real locations 173-175 of the CPU to which the channel is connected. When the length of the full channel logout is 96 bytes or fewer, the channel may either use the IOEL address or store the full channel logout in the fixed-logout area, real locations 256-351 of the CPU to which the channel is connected. 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. The full-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 connected is one.

I/O-COMMUNICATION AREA

Real locations 168-191 of the CPU to which the channel is connected constitute a permanently assigned area of storage used by channels, designated the I/O-communication area (IOCA). (See the figure "I/O-Communication Area.")

Real location 172, 180-184, and 188-191 are reserved for future I/O use.

Channel ID (Locations 168-171): Locations 168-171, when stored during the execution of a STORE CHANNEL ID instruction, contain information which describes the addressed channel.

I/O Extended-Logout Address (Locations 173-175): The I/O extended-logout (IOEL) address (real locations 173-175) should be set by the program to designate an area in absolute storage to be used by channels not capable of storing or not choosing to store the full channel logout in the fixed-logout area (real locations 256-351). The rightmost three bits of the I/O-extended-logout address are reserved and are ignored by the channel so that the full channel logout always begins on a doubleword boundary.

Whether the IOEL facility is used depends on the channel type and model. Channels with a full-channel-logout length not exceeding 96 bytes use either the IOEL area or real locations 256-351 as the full-channel-logout area. Channels with a full-channel-logout length exceeding 96 bytes use the IOEL area.

Programming Note

The extent of the full-channel-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 full channel logout by means of STORE CHANNEL ID whenever a storage area for the full channel logout is to be assigned.

168	Channel ID		
172		IOEL Address	
176	Limited Channel Logout		
180			
184		Measurement Byte	I/O Address
188			

I/O-Communication Area

Limited Channel Logout (Locations 176-179): The limited-channel-logout (LCL) field (real locations 176-179) contains model-independent information related to equipment errors detected by the channel. This information is used to provide detailed machine status when errors have affected I/O operations. The field may be stored only when the CSW or a portion of the CSW is stored.

The limited-channel-logout facility may not be available on all channels. The field, if stored, may or may not be accompanied by the full channel logout. Channels which do not store the limited-channel-logout field usually store equivalent information in the full channel logout.

The bits of the field are defined as follows:

0 This bit is always stored as a zero when a limited channel logout 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 limited channel logout was stored by the channel. The limited channel logout cannot be stored by a channel unless one of these three channel-status bits is set to one.

1-3 Identity of the storage-control unit (SCU). This identifies the SCU through which storage references were directed when an error was detected. This identity is not necessarily the identity of the storage unit involved with data transfer. When only one physical path exists between channel and storage, the storage-control unit has the identity of the CPU to which the channel is connected. If more than one path exists, the storage-control unit has its own identity.

When bit 3 is zero, bits 1 and 2 are undefined. In this case, the SCU identity is implied to be the same as the identity of the CPU to which the channel is connected. 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. This identifies the type of unit that detected the error. At least one bit is present in this field, and multiple bits may be set when more than one unit detects the error.

Bit 4 -- CPU
Bit 5 -- Channel
Bit 6 -- Main-storage control
Bit 7 -- Main storage

8-12 Source field. This indicates the most likely source of the error. The determination is made by the channel on the basis of the type of error check, the location of the checking station, the information flow path, and the success or failure of transmission through previous check stations.

Normally, only one bit will be present in this field. However, when interunit communication cannot be resolved to a single unit, such as when the interface between units is at fault, multiple bits (normally two) may be set to ones in this field. When a reasonable determination cannot be made, all bits in this field are set to zeros.

If the detect and source fields indicate different units, the interface between them can also be considered suspect.

Bit 8 -- CPU
Bit 9 -- Channel
Bit 10 -- Main-storage control
Bit 11 -- Main storage
Bit 12 -- Control unit

13-14 Reserved. Stored zero.

15-23 Field-validity flags. These bits indicate the validity of the information stored in the designated fields. When the validity bit is set to one, the field is stored and usable. When the validity bit is set to zero, the field is not usable.

The fields designated are:

Bit 15 -- Full channel logout. This bit is set to one, by models that implement the recovery-extension facility, when full-channel-logout information with correct contents is stored by the channel. Otherwise, the bit is stored as zero.
Bit 16 -- Reserved. Stored zero
Bit 17 -- Reserved. Stored zero
Bit 18 -- Reserved. Stored zero
Bit 19 -- Sequence code
Bit 20 -- Unit status
Bit 21 -- CCW address and subchannel key in CSW
Bit 22 -- Channel address
Bit 23 -- Device address

24-25 Type of termination that has occurred is indicated by these two bits.

This encoded field has meaning only when a channel-control check or an interface-control check is indicated in the CSW. When neither of these two checks is indicated, no termination has been forced by the channel.

00	Interface disconnect
01	Stop, stack, or normal termination
10	Selective reset
11	System reset

26 Reserved. Stored zero.

27 Interface inoperative. When the recovery-extension facility is installed, this bit is set to one when the channel detects an I/O-interface malfunction which persists after selective reset is signaled on the interface. Interface-control check, channel-control check, or both are also set when this condition is detected. When the recovery-extension facility is not installed, bit 27 is stored as zero.

Programming note: This bit implies that devices involved in active I/O operations related to the identified channel may have been left in the working state. CLEAR CHANNEL addressed to that channel can be used to relieve the condition.

28 I/O-error alert. This bit, when set to one, indicates that the limited channel logout resulted from the signaling of I/O-error alert by the indicated unit. The I/O-error-alert signal indicates that the control unit has detected a malfunction which prevents it from communicating properly with the channel. The channel, in response, performs an I/O selective reset and causes interface-control check to be set.

29-31 Sequence code. This code identifies the I/O sequence in progress at the time of error. It is meaningless if stored during the execution of HALT I/O or HALT DEVICE.

For all cases, the CCW address in the CSW, if validly stored and nonzero, is the address of the current CCW plus 8.

The sequence code assignments are:

000 A channel-detected error occurred during the execution of a TEST I/O or CLEAR I/O instruction.

001 A nonzero command byte has been sent by the channel, but device status has not yet been analyzed by the channel. This code is set during initial selection.

010 The command has been accepted by the device, but no data has been transferred. This code is set if the initial status is either channel end alone, or channel end and device end, or channel end, device end, and status modifier, or all zeros.

011 At least one byte of data has been transferred between the channel and the device. This code is also used when the channel is in an idle or polling state.

100 The command in the current CCW has either not yet been sent to the device or else was sent but not accepted by the device. This code is set when one of the following situations occurs:

1. When the CCW address is updated during command chaining, resuming a suspended channel program, START I/O, or START I/O FAST RELEASE

2. When an initial selection sequence resulted in status 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 the control unit responds with busy status instead of the device address when the channel attempts to select the device

4. When command retry is signaled

5. When the channel interrogates the device in the process of clearing an interruption condition

6. When the channel signals the conclusion of the chain of operations to the device during command chaining while performing the suspend function

101 The command in the current CCW has been accepted, but data transfer is unpredictable. This code applies from the time a device is logically connected to the channel until the time it is determined that a new sequence code applies. The code may also be used when a channel is in the polling or idle state, and it is not possible to determine that code 010 or 011 applies. The code may also be used at other times when a channel cannot distinguish between code 010 or 011.

110 Reserved.

111 Reserved.

Measurement Byte (Location 185): A value is stored at real location 185 whenever an I/O address is stored at real locations 186-187. Whenever the channel stores a complete CSW during an interruption in EC mode and the CSW indicates the conclusion of an operation initiated via START I/O FAST RELEASE executed independent of the device for a subchannel provided with start-I/O-fast queuing, the measurement byte (which is otherwise stored as zeros) has the following format:

```
Location 185 (real)  ┌──┬───┬───┐
                     │DC│NPO│000│
                     └──┴───┴───┘
                      0  2   5 7
```

The bits of the measurement byte are defined as follows:

0-1 Delay Code (DC). This code indicates the condition encountered by the channel on the first attempt by the channel to initiate the I/O operation at the device. Delay codes are as follows:

Delay Code	Meaning
00	No busy condition encountered or no valid code available
01	Channel busy
10	Control unit busy
11	Device busy

2-4 Number of Pending Operations (NPO). These bits contain the binary count of the number of pending I/O operations for the channel at the time the measurement byte is stored. A value of all ones represents seven or more pending I/O operations. A value of all zeros represents either no pending I/O operations or no valid number available.

Otherwise, the measurement byte is stored as zeros.

Errors detected during the execution of an I/O operation do not affect the validity of the values stored in the measurement byte unless the channel-control-check condition is indicated in the CSW. A channel-control-check condition that affects the validity of the delay code or the number of pending I/O operations causes the channel to store zeros in the measurement byte.

I/O Address (Locations 186-187): A two-byte field is provided at real locations 186-187 for storing the I/O address on each I/O interruption in the EC mode, and at the conclusion of a successful initial-program-loading sequence in the EC mode.

Programming Note

I/O-busy conditions result from contention for shared resources in the I/O system. Such contention is not apparent to the program to the extent that I/O-busy conditions are handled by channels when start-I/O-fast queuing is provided. In order to provide some indication of I/O-busy conditions handled by channels, the measurement byte is provided in systems that provide start-I/O-fast queuing and are operating in EC mode.

APPENDIX A. NUMBER REPRESENTATION AND INSTRUCTION-USE EXAMPLES

NUMBER REPRESENTATION

BINARY INTEGERS

Signed Binary Integers

Signed binary integers are most commonly represented as halfwords (16 bits) or words (32 bits). In both lengths, the leftmost bit (bit 0) is the sign of the number. The remaining bits (bits 1-15 for halfwords and 1-31 for words) are used to specify the magnitude of the number. Binary integers are also referred to as fixed-point numbers, because the radix point (binary point) is considered to be fixed at the right, and any scaling is done by the programmer.

Positive binary integers are in true binary notation with a zero sign bit. Negative binary integers are in two's-complement notation with a one bit in the sign position. In all cases, the bits between the sign bit and the leftmost significant bit of the integer are the same as the sign bit (that is, all zeros for positive numbers, all ones for negative numbers).

Negative binary integers are formed in two's-complement notation by inverting each bit of the positive binary integer and adding one. As an example using the halfword format, the binary number with the decimal value +26 is made negative (-26) in the following manner:

```
+26     0 000 0000 0001 1010
Invert  1 111 1111 1110 0101
Add 1                      1
        ____________________
-26     1 111 1111 1110 0110 (Two's
                              complement
                              form)
```

(S is the sign bit.)

This is equivalent to subtracting the number:

```
        00000000 00011010
from
      1 00000000 00000000
```

Negative binary integers are changed to positive in the same manner.

The following addition examples illustrate two's-complement arithmetic and overflow conditions. Only eight bit positions are used.

```
1. +57 = 0011 1001
   +35 = 0010 0011
   _______________
   +92 = 0101 1100
```

```
2.  +57 = 0011 1001
    -35 = 1101 1101
    ---------------
    +22 = 0001 0110 No overflow -- carry
                    into leftmost posi-
                    tion and carry out

3.  +35 = 0010 0011
    -57 = 1100 0111
    ---------------
    -22 = 1110 1010 Sign change only --
                    no carry into left-
                    most position and no
                    carry out

4.  -57 = 1100 0111
    -35 = 1101 1101
    ---------------
    -92 = 1010 0100 No overflow -- carry
                    into leftmost posi-
                    tion and carry out

5.  +57 = 0011 1001
    +92 = 0101 1100
    ---------------
  +149 =*1001 0101 *Overflow -- carry
                    into leftmost posi-
                    tion, no carry out

6.  -57 = 1100 0111
    -92 = 1010 0100
    ---------------
  -149 =*0110 1011 *Overflow -- no carry
                    into leftmost posi-
                    tion but carry out
```

The presence or absence of an overflow condition may be recognized from the carries:

- There is no overflow:

  a. If there is no carry into the leftmost bit position and no carry out (examples 1 and 3).

  b. If there is a carry into the leftmost position and also a carry out (examples 2 and 4).

- There is an overflow:

  a. If there is a carry into the leftmost position but no carry out (example 5).

  b. If there is no carry into the leftmost position but there is a carry out (example 6).

The following are 16-bit signed binary integers. The first is the maximum positive 16-bit binary integer. The last is the maximum negative 16-bit binary integer (the negative 16-bit binary integer with the greatest absolute value).

$2^{15}-1$	=	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 figure illustrates several 32-bit signed binary integers arranged in descending order. The first is the maximum positive binary integer that can be represented by 32 bits, and the last is the maximum negative binary integer that can be represented by 32 bits.

$2^{31}-1$	=	2 147 483 647	=	0 111 1111 1111 1111 1111 1111 1111 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

32-Bit Signed Binary Integers

Unsigned Binary Integers

Certain instructions, such as ADD LOGICAL, treat binary integers as unsigned rather than signed. Unsigned binary integers have the same format as signed binary integers, except that the leftmost bit is interpreted as another numeric bit rather than a sign bit. There is no complement notation because all unsigned binary integers are considered positive.

The following examples illustrate the addition of unsigned binary integers. Only eight bit positions are used. The examples are numbered the same as the corresponding examples for signed binary integers.

```
1.  57 = 0011 1001
    35 = 0010 0011
    ______________
    92 = 0101 1100
```

```
2.  57 = 0011 1001
   221 = 1101 1101
   _______________
   278 =*0001 0110 *Carry out of
                    leftmost position
```

```
3.  35 = 0010 0011
   199 = 1100 0111
   _______________
   234 = 1110 1010
```

```
4. 199 = 1100 0111
   221 = 1101 1101
   _______________
   420 =*1010 0100 *Carry out of
                    leftmost position
```

```
5.  57 = 0011 1001
    92 = 0101 1100
    ______________
   149 = 1001 0101
```

```
6. 199 = 1100 0111
   164 = 1010 0100
   _______________
   363 =*0110 1011 *Carry out of
                    leftmost position
```

A carry out of the leftmost bit position may or may not imply an overflow, depending on the application.

The following figure illustrates several 32-bit unsigned binary integers arranged in descending order.

$2^{32}-1$	=	4 294 967 295	=	1111 1111 1111 1111 1111 1111 1111 1111
2^{31}	=	2 147 483 648	=	1000 0000 0000 0000 0000 0000 0000 0000
$2^{31}-1$	=	2 147 483 647	=	0111 1111 1111 1111 1111 1111 1111 1111
2^{16}	=	65 536	=	0000 0000 0000 0001 0000 0000 0000 0000
2^{0}	=	1	=	0000 0000 0000 0000 0000 0000 0000 0001
0	=	0	=	0000 0000 0000 0000 0000 0000 0000 0000

32-Bit Unsigned Binary Integers

DECIMAL INTEGERS

Decimal integers consist of one or more decimal digits and a sign. Each digit and the sign are represented by a 4-bit code. The decimal digits are in binary-coded decimal (BCD) form, with the values 0-9 encoded as 0000-1001. The sign is usually represented as 1100 (C hex) for plus and 1101 (D hex) for minus. These are the preferred sign codes, which are generated by the machine for the results of decimal-arithmetic operations. There are also several alternate sign codes (1010, 1110, and 1111 for plus; 1011 for minus). The alternate sign codes are accepted by the machine as valid in source operands but are not generated for results.

Decimal integers may have different lengths, from one to 16 bytes. There are two decimal formats: packed and zoned. In the packed format, each byte contains two decimal digits, except for the rightmost byte, which contains the sign code in the right half. For decimal arithmetic, the number of decimal digits in the packed format can vary from one to 31. Because decimal integers must consist of whole bytes and there must be a sign code on the right, the number of decimal digits is always odd. If an even number of significant digits is desired, a leading zero must be inserted on the left.

In the zoned format, each byte consists of a decimal digit on the right and the zone code 1111 (F hex) on the left, except for the rightmost byte where the sign code replaces the zone code. Thus, a decimal integer in the zoned format can have from one to 16 digits. The zoned format may be used directly for input and output in the extended binary-coded-decimal interchange code (EBCDIC), except that the sign must be separated from the rightmost digit and handled as a separate character. For positive (unsigned) numbers, however, the sign can simply be represented by the zone code of the rightmost digit because the zone code is one of the acceptable alternate codes for plus.

In either format, negative decimal integers are represented in true notation with a separate sign. As for binary integers, the radix point (decimal point) of decimal integers is considered to be fixed at the right, and any scaling is done by the programmer.

The following are some examples of decimal integers shown in hexadecimal notation:

Decimal Value	Packed Format	Zoned Format
+123	12 3C	F1 F2 C3
	or	or
	12 3F	F1 F2 F3
-4321	04 32 1D	F4 F3 F2 D1
+000050	00 00 05 0C	F0 F0 F0 F0 F5 C0
	or	or
	00 00 05 0F	F0 F0 F0 F0 F5 F0
-7	7D	D7
00000	00 00 0C	F0 F0 F0 F0 C0
	or	or
	00 00 0F	F0 F0 F0 F0 F0

Under some circumstances, a zero with a minus sign (negative zero) is produced. For example, the multiplicand:

```
     00 12 3D   (-123)
```

times the multiplier:

```
        0C      (+0)
```

generates the product:

```
     00 00 0D   (-0)
```

because the product sign follows the algebraic rule of signs even when the value is zero. A negative zero, however, is equivalent to a positive zero in that they compare equal in a decimal comparison.

FLOATING-POINT NUMBERS

A floating-point number is expressed as a hexadecimal fraction multiplied by a separate power of 16. The term floating point indicates that the placement, of the radix (hexadecimal) point, or scaling, is automatically maintained by the machine.

The part of a floating-point number which represents the significant digits of the number is called the fraction. A second part specifies the power (exponent) to which 16 is raised and indicates the location of the radix point of the number. The fraction and exponent may be represented by 32 bits (short format), 64 bits (long format), or 128 bits (extended format).

Short Floating-Point Number

S	Characteristic	6-Digit Fraction	
0	1	8	31

Long Floating-Point Number

S	Characteristic	14-Digit Fraction
0	1	8 63

Extended Floating-Point Number

High-Order Part

S	High-Order Characteristic	Leftmost 14 Digits of 28-Digit Fraction
0	1	8 63

Low-Order Part

S	Low-Order Characteristic	Rightmost 14 Digits of 28-Digit Fraction
64		72 127

A floating-point number has two signs: one for the fraction and one for the exponent. The fraction sign, which is also the sign of the entire number, is the leftmost bit of each format (0 for plus, 1 for minus). The numeric part of the fraction is in true notation regardless of the sign. The numeric part is contained in bits 8-31 for the short format, in bits 8-63 for the long format, and in bits 8-63 followed by bits 72-127 for the extended format.

The exponent sign is obtained by expressing the exponent in excess-64 notation; that is, the exponent is added as a signed number to 64. The resulting number is called the characteristic. It is located in bits 1-7 for all formats. The characteristic can vary from 0 to 127, permitting the exponent to vary from -64 through 0 to +63. This provides a scale multiplier in the range of 16^{-64} to 16^{+63}. A nonzero fraction, if normalized, has a value less than one and greater than or equal to 1/16, so that the range covered by the magnitude M of a normalized floating-point number is:

$$16^{-65} \le M < 16^{63}$$

In decimal terms:

16^{-65} is approximately 5.4×10^{-79}

16^{63} is approximately 7.2×10^{75}

More precisely,

In the short format:

$$16^{-65} \le M \le (1 - 16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \le M \le (1 - 16^{-14}) \times 16^{63}$$

In the extended format:

$$16^{-65} \le M \le (1 - 16^{-28}) \times 16^{63}$$

Within a given fraction length (6, 14, or 28 digits), a floating-point operation will provide the greatest precision if the fraction is normalized. A fraction is normalized when the leftmost digit (bit positions 8, 9, 10, and 11) is nonzero. It is unnormalized if the leftmost digit contains all zeros.

If normalization of the operand is desired, the 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 leftmost digit position. The characteristic is reduced by one for each digit shifted.

The following figure illustrates sample normalized short floating-point numbers. The last two numbers represent the smallest and the largest positive normalized numbers.

1.0	=	$+1/16x16^{1}$	=	0 100 0001 0001 0000 0000 0000 0000 0000{2}
0.5	=	$+8/16x16^{0}$	=	0 100 0000 1000 0000 0000 0000 0000 0000{2}
1/64	=	$+4/16x16^{-1}$	=	0 011 1111 0100 0000 0000 0000 0000 0000{2}
0.0	=	$+0 \quad x16^{-64}$	=	0 000 0000 0000 0000 0000 0000 0000 0000{2}
-15.0	=	$-15/16x16^{1}$	=	1 100 0001 1111 0000 0000 0000 0000 0000{2}
$5.4x10^{-79}$	~	$+1/16x16^{-64}$	=	0 000 0000 0001 0000 0000 0000 0000 0000{2}
$7.2x10^{75}$	~	$(1-16^{-6})x16^{63}$	=	0 111 1111 1111 1111 1111 1111 1111 1111{2}

[The symbol ~ means "approximately equal."]

Normalized Short Floating-Point Numbers

CONVERSION EXAMPLE

Convert the decimal number 59.25 to a short floating-point number. (In another appendix are tables for the conversion of hexadecimal and decimal integers and fractions.)

1. The number is separated into a decimal integer and a decimal fraction.

   59.25 = 59 plus 0.25

2. The decimal integer is converted to its hexadecimal representation.

   59{10} = 3B{16}

3. The decimal fraction is converted to its hexadecimal representation.

   0.25{10} = 0.4{16}

4. The integral and fractional parts are combined and expressed as a fraction times a power of 16 (exponent).

   3B.4{16} = 0.3B4{16} x 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.

   .3B4{16} = .0011 1011 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	0011 1011 0100 0000 0000 0000

Examples of instruction sequences that may be used to convert between signed binary integers and floating-point numbers are shown in the section "Floating-Point-Number Conversion" later in this appendix.

INSTRUCTION-USE EXAMPLES

The following examples illustrate the use of many of the unprivileged instructions. Before studying one of these examples, the reader should consult the instruction description.

The instruction-use examples are written principally for assembler-language programmers, to be used in conjunction with the appropriate assembler-language publications.

Most examples present one particular instruction, both as it is written in an assembler-language statement and as it appears when assembled in storage (machine format).

In the instruction-use examples, the notation {2}, {10}, or {16} may be used, indicating that the preceding number is binary, decimal, or hexadecimal, respectively.

MACHINE FORMAT

All machine-format values are given in hexadecimal notation unless otherwise specified. Storage addresses are also given in hexadecimal. Hexadecimal operands are shown converted into binary, decimal, or both if such conversion helps to clarify the example for the reader.

ASSEMBLER-LANGUAGE FORMAT

In assembler-language statements, registers and lengths are presented in decimal. Displacements, immediate operands, and masks may be shown in decimal, hexadecimal, or binary notation; for example, 12, X'C', and B'1100' represent the same value. Whenever the value in a register or storage location is referred to as "not significant," this value is replaced during the execution of the instruction.

When SS-format instructions are written in the assembler language, lengths are given as the total number of bytes in the field. This differs from the machine definition, in which the length field specifies the number of bytes to be added to the field address to obtain the address of the last byte of the field. Thus, the machine length is one less than the assembler-language length. The assembler program automatically subtracts one from the length specified when the instruction is assembled.

In some of the examples, symbolic addresses are used in order to simplify the examples. In assembler-language statements, a symbolic address is represented as a mnemonic term written in all capitals, such as FLAGS, which may denote the address of a storage location containing data or program-control information. When symbolic addresses are used, the assembler supplies actual base and displacement values according to the programmer's specifications. Therefore, the actual values for base and displacement are not shown in the assembler-language format or in the machine-language format. For assembler-language formats, in the labels that designate instruction fields, the letter "S" is used to indicate the combination of base and displacement fields for an operand address. (For example, S2 represents the combination of B2 and D2.) In the machine-language format, the base and displacement address components are shown as asterisks (****).

GENERAL INSTRUCTIONS

(See Chapter 7 for a complete description of the general instructions.)

ADD HALFWORD (AH)

The ADD HALFWORD instruction algebraically adds the contents of a two-byte field in storage to the contents of a register. The storage operand is expanded to 32 bits after it is fetched and before it is used in the add operation. The expansion consists in propagating the leftmost (sign) bit 16 positions to the left. For example, assume that the contents of storage locations 2000-2001 are to be added to register 5. Initially:

Register 5 contains 00 00 00 19 = 25{10}.
Storage locations 2000-2001 contain FF FE = -2{10}.
Register 12 contains 00 00 18 00.
Register 13 contains 00 00 01 50.

The format of the required instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4A	5	D	C	6B0

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
AH	5,X'6B0'(13,12)

After the instruction is executed, register 5 contains 00 00 00 17 = 23{10}. Condition code 2 is set to indicate a result greater than zero.

AND (N, NC, NI, NR)

When the Boolean operator AND is applied to two bits, the result is one when both bits are one; otherwise, the result is zero. When two bytes are ANDed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of ANDing two bytes:

First-operand byte:	0011 0101{2}
Second-operand byte:	0101 1100{2}
Result byte:	0001 0100{2}

NI Example

A frequent use of the AND instruction is to set a particular bit to zero. For example, assume that storage location 4891 contains 0100 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 (the I_2 field of the instruction):

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 back in location 4891. Condition code 1 is set.

LINKAGE INSTRUCTIONS (BAL, BALR, BAS, BASR)

The BRANCH AND LINK (BAL or BALR) instruction is commonly used to branch to a subroutine with the option of later returning to the main instruction sequence. On models with the branch-and-save facility, the BRANCH AND SAVE (BAS or BASR) instructions may be used for the same purpose. Both save the address of the next instruction as link information in a general register and then cause execution to continue from a different instruction sequence at the branch address specified by this instruction. They differ in that BRANCH AND LINK places additional information (the instruction-length code, condition code, and program mask) in the leftmost byte of the link information, whereas BRANCH AND SAVE places zeros in that byte.

BRANCH AND SAVE, when available, is recommended for use in place of BRANCH AND LINK in programs that are intended to be executed on System/370 models equipped with the extended-architecture (370-XA) mode. When such a model is operating in the 370-XA mode, the information placed by BRANCH AND LINK in the leftmost byte of the linkage register while 24-bit addressing is in effect may lead to problems if the same program may be used with 31-bit addressing; BRANCH AND SAVE sets the leftmost byte to zero with 24-bit addressing, which is compatible with 31-bit addressing. (For more information on 31-bit addressing and on subroutine linkage methods for the 370-XA mode, see the IBM System/370 Extended Architecture Principles of Operation, SA22-7085.)

The following example compares the operation of these instructions and of the unconditional-branch instruction BRANCH ON CONDITION (BC or BCR with a mask of 15). Assume that each instruction in turn is located at the current instruction address, ready to be executed next. Assume also that general register 5 is to receive the linkage information, and that general register 6 contains the branch address.

The format of the BALR instruction is:

Machine Format

Op Code	R_1	R_2
05	5	6

Assembler Format

Op Code R_1,R_2

BALR 5,6

The BASR instruction has the same format, but the op code is 0D.

For comparison with the RR-format instructions, the results of two RX-format instructions are also shown.

The format of the BAL instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
45	5	0	6	000

Assembler Format

Op Code $R_1,D_2(X_2,B_2)$

BAL 5,0(0,6)

The BAS instruction has the same format, but the op code is 4D.

The BCR instruction specifies only one register:

Machine Format

Op Code	M_1	R_2
07	F	6

Assembler Format

Op Code M_1,R_2

BCR 15,6

Assume that:

Register 5 contains BB BB BB BB.
Register 6 contains 82 46 8A CE.
PSW bits 32-63 contain 00 00 10 D6.
Condition code is 01{2}.
Program mask is 1100{2}.

The effect of executing each instruction in turn is as follows:

Instruction	Register 5	PSW (32-63)
Before	BB BB BB BB	00 00 10 D6
BCR 15,6	BB BB BB BB	00 46 8A CE
BAL 5,0(0,6)	9C 00 10 DA	00 46 8A CE
BAS 5,0(0,6)	00 00 10 DA	00 46 8A CE
BALR 5,6	5C 00 10 D8	00 46 8A CE
BASR 5,6	00 00 10 D8	00 46 8A CE

Note that a value of zero in the R_2 field of any of the RR-format instructions indicates that the branching function is not to be performed; it does not refer to register 0. Thus, the instruction BALR 8,0 may be used to preserve the current condition code in bits 2 and 3 of register 8 for future inspection. Register 0 can be designated by the R_1 field, however. In the RX-format branch instructions, branching occurs independent of whether there is a value of zero in the B_2 field or X_2

field of the instruction. However, when the field is zero, instead of using the contents of general register 0, a value of zero is used for that component of address generation.

Other BALR and BASR Examples

The BALR or BASR instruction with the R_2 field set to zero may be used to load a register for use as a base register. For example, in the assembler language, the two statements:

```
     BALR    15,0
     USING   *,15

     or

     BASR    15,0
     USING   *,15
```

indicate that the address of the next sequential instruction following the BALR or BASR instruction will be placed in register 15, and that the assembler may use register 15 as a base register until otherwise instructed. (The USING statement is an "assembler instruction" and is thus not a part of the object program.)

BRANCH ON CONDITION (BC, BCR)

The BRANCH ON CONDITION instruction tests the condition code to see whether a branch should or should not occur. The branch occurs only if the current condition code corresponds to a one bit in a mask specified by the instruction.

Condition Code	Instruction (Mask) Bit	Mask Value
0	8	8
1	9	4
2	10	2
3	11	1

For example, assume that an ADD (A or AR) operation has been performed and that a branch to address 6050 is desired if the sum is zero or less (condition code is 0 or 1). Also assume:

Register 10 contains 00 00 50 00.
Register 11 contains 00 00 10 00.

The RX form of the instruction performs the required test (and branch if necessary) when written as:

Machine Format

Op Code	M_1	X_2	B_2	D_2
47	C	B	A	050

Assembler Format

Op Code	$M_1,D_2(X_2,B_2)$
BC	12,X'50'(11,10)

A mask of 12{10} means that there are ones in instruction bits 8 and 9 and zeros in bits 10 and 11, so that branching takes place when the condition code is either 0 or 1.

A mask of 15 would indicate a branch on any condition (an unconditional branch). A mask of zero would indicate that no branch is to occur (a no-operation).

(See also the section on "Linkage Instructions (BAL, BALR, BAS, BASR)" for an example of the BCR instruction.)

BRANCH ON COUNT (BCT, BCTR)

The BRANCH ON COUNT instruction is often used to execute a program loop for a specified number of times. For example, assume that the following represents some lines of coding in an assembler-language program:

```
          .
          .
          .
     LUPE  AR   8,1
          .
          .
     BACK  BCT  6,LUPE
          .
          .
          .
```

where register 6 contains 00 00 00 03 and the address of LUPE is 6826. Assume that, in order to address this location, register 10 is used as a base register and contains 00 00 68 00.

The format of the BCT instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
46	6	0	A	026

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
BCT	6,X'26'(0,10)

The effect of the coding is to execute three times the loop defined by the instructions labeled LUPE through BACK, while register 6 is decremented from three to zero.

BRANCH ON INDEX HIGH (BXH)

BXH Example 1

The BRANCH ON INDEX HIGH instruction is an index-incrementing and loop-controlling instruction that causes a branch whenever the sum of an index value and an increment value is greater than some compare value. For example, assume that:

Register 4 contains 00 00 00 8A = 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 compare value.
Register 10 contains 00 00 71 30 = the branch address.

The format of the BXH instruction is:

Machine Format

Op Code	R_1	R_3	B_2	D_2
86	4	6	A	000

Assembler Format

Op Code	$R_1,R_3,D_2(B_2)$
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 whether 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 yet greater than the value in register 7, the branch to address 7130 is not taken. Repeated use of the instruction will eventually cause the branch to be taken when the value in register 4 reaches 172{10}.

BXH Example 2

When the register used to contain the increment is odd, that register also becomes the compare-value register. The following assembler-language subroutine illustrates how this may be used to search a table.

Table	
2 Bytes	2 Bytes
ARG1	FUNCT1
ARG2	FUNCT2
ARG3	FUNCT3
ARG4	FUNCT4
ARG5	FUNCT5
ARG6	FUNCT6

Assume that:

Register 8 contains the search argument.
Register 9 contains the width of the table in bytes (00 00 00 04).
Register 10 contains the length of the table in bytes (00 00 00 18).
Register 11 contains the starting address of the table.
Register 14 contains the return address to the main program.

As the following subroutine is executed, the argument in register 8 is successively compared with the arguments in the table, starting with argument 6 and working backward to argument 1. If an equality is found, the corresponding function replaces the argument in register 8. If an equality is not found, zero replaces the argument in register 8.

```
SEARCH     LNR  9,9
NOTEQUAL   BXH  10,9,LOOP
NOTFOUND   SR   8,8
           BCR  15,14
LOOP       CH   8,0(10,11)
           BC   7,NOTEQUAL
           LH   8,2(10,11)
           BCR  15,14
```

The first instruction (LNR) causes the value in register 9 to be made negative. After execution of this instruction, register 9 contains FF FF FF FC = -4{10}. Considering the case when no equality is found, the BXH instruction will be executed seven times. Each time BXH is executed, a value of -4 is added to register 10, thus reducing the value in register 10 by 4. The new value in register 10 is compared with the -4 value in register 9. The branch is taken each time until the value in register 10 is -4. Then the branch is not taken, and the SR instruction sets register 8 to zero.

BRANCH ON INDEX LOW OR EQUAL (BXLE)

The BRANCH ON INDEX LOW OR EQUAL instruction performs the same operation as BRANCH ON INDEX HIGH, except that branching occurs when the sum is lower than or equal to (instead of higher than) the compare value. As the instruction which increments and tests an index value in a program loop, BXLE is useful at the end of the loop and BXH at the beginning. The following assembler-language routines illustrate loops with BXLE.

BXLE Example 1

Assume that a group of ten 32-bit signed binary integers are stored at consecutive locations, starting at location GROUP. The integers are to be added together, and the sum is to be stored at location SUM.

```
     SR   5,5        Set sum to zero
     LA   6,GROUP    Load first address
     SR   7,7        Set index to zero
     LA   8,4        Load increment 4
     LA   9,39       Load compare value
LOOP A    5,0(7,6)   Add integer to sum
     BXLE 7,8,LOOP   Test end of loop
     ST   5,SUM      Store sum
```

The two-instruction loop contains an ADD (A) instruction which adds each integer to the contents of general register 5. The ADD instruction uses the contents of general register 7 as an index value to modify the starting address obtained from register 6. Next, BXLE increments the index value by 4, the increment previously loaded into register 8, and compares it with the compare value in register 9, the odd register of this even-odd pair. The compare value was previously set to 39, which is one less than the number of bytes in the data area; this is also the address, relative to the starting address, of the rightmost byte of the last integer to be added. When the last integer has been added, BXLE increments the index value to the next relative address (40), which is found to be greater than the compare value (39) so that no branching takes place.

BXLE Example 2

The technique illustrated in Example 1 is restricted to loops containing instructions in the RX instruction format. That format allows both a base register and an index register to be specified (double indexing).

For instructions in other formats, where an index register cannot be specified, the previous technique may be modified by having the address itself serve as the index value in a BXLE instruction and by using as the compare value the address of the last byte rather than its relative address. The base register then provides the address directly at each iteration of the loop, and it is not necessary to specify a second register to hold the index value (single indexing).

In the following example, an AND (NI) instruction in the SI instruction format sets to zero the rightmost bit of each of the same group of integers as in Example 1, thus making all of them even. The I_2 field of the NI instruction contains the byte X'FE', which consists of seven ones and a zero. That byte is ANDed into byte 3, the rightmost byte, of each of the integers in turn.

```
     LA   6,GROUP      Load first address
     LA   8,4          Load increment 4
     LA   9,GROUP+39   Load compare value
LOOP NI   3(6),X'FE'   AND immediate
     BXLE 6,8,LOOP     Test end of loop
```

COMPARE HALFWORD (CH)

The COMPARE HALFWORD instruction compares a 16-bit signed binary integer in storage with the contents of a register. For example, assume that:

Register 4 contains FF FF 80 00 = -32,768{10}.
Register 13 contains 00 01 60 50.
Storage locations 16080-16081 contain 8000 = -32,768{10}.

When the instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
49	4	0	D	030

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
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, CLC, CLI, CLR)

The COMPARE LOGICAL instruction differs from the signed-binary comparison instructions (C, CH, CR) in that all quantities are handled as unsigned binary integers or as unstructured data.

CLC Example

The COMPARE LOGICAL (CLC) instruction can be used to perform the byte-by-byte comparison of storage fields up to 256 bytes in length. For example, assume that the following two fields of data are in storage:

Field 1
1886 1891

D1	D6	C8	D5	E2	D6	D5	6B	C1	4B	C2	4B

Field 2
1900 190B

D1	D6	C8	D5	E2	D6	D5	6B	C1	4B	C3	4B

Also assume:

Register 9 contains 00 00 18 80.
Register 7 contains 00 00 19 00.

Execution of the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D5	0B	9	006	7	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
CLC	6(12,9),0(7)

sets condition code 1, indicating that the contents of field 1 are lower in value than the contents of field 2.

Because the collating sequence of the EBCDIC code is determined simply by a logical comparison of the bits in the code, the CLC instruction can be used to collate EBCDIC-coded fields. For example, in EBCDIC, the above two data fields are:

Field 1:	JOHNSON,A.B.
Field 2:	JOHNSON,A.C.

Condition code 1 indicates that JOHNSON,A.B. should precede JOHNSON,A.C. for the fields to be in alphabetic sequence.

CLI Example

The COMPARE LOGICAL (CLI) instruction compares a byte from the instruction stream with a byte from storage. For example, assume that:

Register 10 contains 00 00 17 00.
Storage location 1703 contains 7E.

Execution of the instruction:

Machine Format

Op Code	I_2	B_1	D_1
95	AF	A	003

Assembler Format

Op Code	$D_1(B_1),I_2$
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.

CLR Example

Assume that:

Register 4 contains 00 00 00 01 = 1.
Register 7 contains FF FF FF FF = 2^{32} - 1.

Execution of the instruction:

Machine Format

Op Code	R_1	R_2
15	4	7

Assembler Format

```
Op Code  R1,R2
CLR      4,7
```

sets condition code 1. Condition code 1 indicates that the first operand is lower than the second.

If, instead, the signed-binary comparison instruction COMPARE (CR) had been executed, the contents of register 4 would have been interpreted as +1 and the contents of register 7 as -1. Thus, the first operand would have been higher, so that condition code 2 would have been set.

COMPARE LOGICAL CHARACTERS UNDER MASK (CLM)

The COMPARE LOGICAL CHARACTERS UNDER MASK (CLM) instruction provides a means of comparing bytes selected from a general register to a contiguous field of bytes in storage. The M_3 field of the CLM instruction is a four-bit mask that selects zero to four bytes from a general register, each mask bit corresponding, left to right, to a register byte. In the comparison, the register bytes corresponding to ones in the mask are treated as a contiguous field. The operation proceeds left to right. For example, assume that:

Storage locations 10200-10202 contain F0 BC 7B.
Register 12 contains 00 01 00 00.
Register 6 contains F0 BC 5C 7B.

Execution of the instruction:

Machine Format

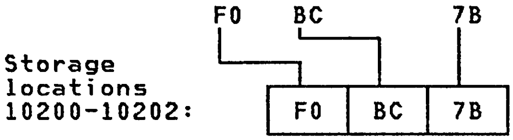
Op Code	R_1	M_3	B_2	D_2
BD	6	D	C	200

Assembler Format

```
Op Code  R1,M3,D2(B2)
CLM      6,B'1101',X'200'(12)
```

causes the following comparison:

Register 6:	F0	BC	5C	7B
Mask M_3:	1	1	0	1
	--	--		--

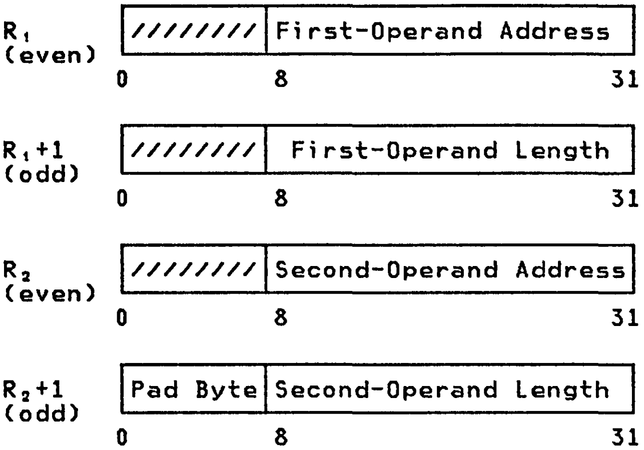


Because the selected bytes are equal, condition code 0 is set.

COMPARE LOGICAL LONG (CLCL)

The COMPARE LOGICAL LONG (CLCL) instruction is used to compare two operands in storage, byte by byte. Each operand can be of any length. Two even-odd pairs of general registers (four registers in all) are used to locate the operands and to control the execution of the CLCL instruction, as illustrated in the following diagram. The first register of each pair must be an even register, and it contains the storage address of an operand. The odd register of each pair contains the length of the operand it covers, and the leftmost byte of the second-operand odd register contains a padding byte which is used to extend the shorter operand, if any, to the same length as the longer operand.

The following illustrates the assignment of registers:


R_1 (even)
//////// First-Operand Address
0 8 31
R_1+1 (odd)
//////// First-Operand Length
0 8 31
R_2 (even)
//////// Second-Operand Address
0 8 31
R_2+1 (odd)
Pad Byte Second-Operand Length
0 8 31

Since the CLCL instruction may be interrupted during execution, the interrupting program must preserve the contents of the four registers for use when the instruction is resumed.

The following instructions set up two register pairs to control a text-string comparison. For example, assume:

Operand 1

Address: 20800{16}
Length: 100{10}

Operand 2

Address: 20A00{16}
Length: 132{10}

Padding Byte

Address: 20003{16}
Length: 1
Value: 40{16}

Register 12 contains 00 02 00 00.

The setup instructions are:

```
LA  4,X'800'(12)      Set register 4 to
                      start of first
                      operand
LA  5,100             Set register 5 to
                      length of first
                      operand
LA  8,X'A00'(12)      Set register 8 to
                      start of second
                      operand
LA  9,132             Set register 9 to
                      length of second
                      operand
ICM 9,B'1000',3(12)   Insert padding byte
                      in leftmost byte
                      position of regis-
                      ter 9
```

Register pair 4,5 defines the first operand. Bits 8-31 of register 4 contain the storage address of the start of an EBCDIC text string, and bits 8-31 of register 5 contain the length of the string, in this case 100 bytes.

Register pair 8,9 defines the second operand, with bits 8-31 of register 8 containing the starting location of the second operand and bits 8-31 of register 9 containing the length of the second operand, in this case 132 bytes. Bits 0-7 of register 9 contain an EBCDIC blank character (X'40') to pad the shorter operand. In this example, the padding byte is used in the first operand, after the 100th byte, to compare with the remaining bytes in the second operand.

With the register pairs thus set up, the format of the CLCL instruction is:

Machine Format

Op Code	R_1	R_2
0F	4	8

Assembler Format

Op Code	R_1,R_2
CLCL	4,8

When this instruction is executed, the comparison starts at the left end of each operand and proceeds to the right. The operation ends as soon as an inequality is detected or the end of the longer operand is reached.

If this CLCL instruction is interrupted after 60 bytes have compared equal, the operand lengths in registers 5 and 9 will have been decremented to 40 and 72, respectively. The operand addresses in registers 4 and 8 will have been incremented to X'2083C' and X'20A3C'; the leftmost byte of registers 4 and 8 will have been set to zero. The padding byte X'40' remains in register 9. When the CLCL instruction is reexecuted with these register contents, the comparison resumes at the point of interruption.

Now, assume that the instruction is interrupted after 110 bytes. That is, the first 100 bytes of the second operand have compared equal to the first operand, and the next 10 bytes of the second operand have compared equal to the padding byte (blank). The residual operand lengths in registers 5 and 9 are 0 and 22, respectively, and the operand addresses in registers 4 and 8 are X'20864' (the value when the first operand was exhausted) and X'20A6E' (the current value for the second operand).

When the comparison ends, the condition code is set to 0, 1, or 2, depending on whether the first operand is equal to, less than, or greater than the second operand, respectively.

When the operands are unequal, the addresses in registers 4 and 8 indicate the bytes that caused the mismatch.

CONVERT TO BINARY (CVB)

The CONVERT TO BINARY instruction converts an eight-byte, packed-decimal number into a signed binary integer and loads the result into a general register. After the conversion operation is completed, the number is in the proper form for use as an operand in signed binary arithmetic. For example, assume:

Storage locations 7608-760F contain a decimal number in the packed format: 00 00 00 00 00 25 59 4C (+25,594).
The contents of register 7 are not significant.
Register 13 contains 00 00 76 00.

The format of the conversion instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4F	7	0	D	008

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
CVB	7,8(0,13)

After the instruction is executed, register 7 contains 00 00 63 FA.

CONVERT TO DECIMAL (CVD)

The CONVERT TO DECIMAL instruction is the opposite of the CONVERT TO BINARY instruction. CVD converts a signed binary integer in a register to packed decimal and stores the eight-byte result. For example, assume:

Register 1 contains the signed binary integer: 00 00 0F 0F.
Register 13 contains 00 00 76 00.

The format of the instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4E	1	0	D	008

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
CVD	1,8(0,13)

After the instruction is executed, storage locations 7608-760F contain 00 00 00 00 00 03 85 5C (+3855).

The plus sign generated is the preferred plus sign, 1100{2}.

DIVIDE (D, DR)

The DIVIDE instruction divides the dividend in an even-odd register pair by the divisor in a register or in storage. Since the instruction assumes the dividend to be 64 bits long, it is important first to extend a 32-bit dividend on the left with bits equal to the sign bit. For example, assume that:

Storage locations 3550-3553 contain 00 00 08 DE = 2270{10} (the dividend).
Storage locations 3554-3557 contain 00 00 00 32 = 50{10} (the divisor).
The initial contents of registers 6 and 7 are not significant.
Register 8 contains 00 00 35 50.

The following assembler-language statements load the registers properly and perform the divide operation:

Statement	Comments
L 6,0(0,8)	Places 00 00 08 DE into register 6.
SRDA 6,32(0)	Shifts 00 00 08 DE into register 7. Register 6 is filled with zeros (sign bits).
D 6,4(0,8)	Performs the division.

The machine format of the above DIVIDE instruction is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
5D	6	0	8	004

After the instructions listed above are executed:

Register 6 contains 00 00 00 14 = 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 might not have been filled with the proper dividend-sign bits (zeros in this example), and the DIVIDE instruction might not have given the expected results.

EXCLUSIVE OR (X, XC, XI, XR)

When the Boolean operator EXCLUSIVE OR is applied to two bits, the result is one when either, but not both, of the two bits is one; otherwise, the result is zero. When two bytes are EXCLUSIVE ORed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of the EXCLUSIVE OR of two bytes:

```
First-operand byte:   0011 0101{2}
Second-operand byte:  0101 1100{2}
--------------------------------------
Result byte:          0110 1001{2}
```

XC Example

The EXCLUSIVE OR (XC) instruction can be used to exchange the contents of two areas in storage without the use of an intermediate storage area. For example, assume two three-byte fields in storage:

	359		35B
Field 1	00	17	90

	360		362
Field 2	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_1	D_2
D7	02	7	001	7	008

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
XC	1(3,7),8(7)

Field 1 is EXCLUSIVE ORed with field 2 as follows:

```
Field 1:  00000000 00010111 10010000{2}
        = 00 17 90{16}
Field 2:  00000000 00010100 00000001{2}
        = 00 14 01{16}
------------------------------------------
Result:   00000000 00000011 10010001{2}
        = 00 03 91{16}
```

The result replaces the former contents of field 1. Condition code 1 is set to indicate a nonzero result.

Now, execution of the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D7	02	7	008	7	001

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
XC	8(3,7),1(7)

produces the following result:

```
Field 1: 00000000 00000011 10010001{2}
       = 00 03 91{16}
Field 2: 00000000 00010100 00000001{2}
       = 00 14 01{16}
-----------------------------------------
Result:  00000000 00010111 10010000{2}
       = 00 17 90{16}
```

The result of this operation replaces the former contents of field 2. Field 2 now contains the original value of field 1. Condition code 1 is set to indicate a nonzero result.

Lastly, execution of the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D7	02	7	001	7	008

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
XC	1(3,7),8(7)

produces the following result:

```
Field 1: 00000000 00000011 10010001{2}
       = 00 03 91{16}
Field 2: 00000000 00010111 10010000{2}
       = 00 17 90{16}
-------------------------------------
Result:  00000000 00010100 00000001{2}
       = 00 14 01{16}
```

The result of this operation replaces the former contents of field 1. Field 1 now contains the original value of field 2. Condition code 1 is set to indicate a nonzero result.

XI Example

A frequent use of the EXCLUSIVE OR (XI) instruction is to invert a bit (change a zero bit to a one or a one bit to a zero). For example, assume that storage location 8082 contains 0110 1001{2}. To invert the leftmost and rightmost bits without affecting any of the other bits, the following instruction can be used (assume that register 9 contains 00 00 80 80):

Machine Format

Op Code	I_2	B_1	D_1
97	81	9	002

Assembler Format

Op Code	$D_1(B_1),I_2$
XI	2(9),X'81'

When the instruction is executed, the byte in storage is EXCLUSIVE ORed with the immediate byte (the I_2 field of the instruction):

```
Location 8082:   0110 1001{2}
Immediate byte:  1000 0001{2}
-----------------------------
Result:          1110 1000{2}
```

The resulting byte is stored back in location 8082. Condition code 1 is set to indicate a nonzero result.

Notes:

1. With the XC instruction, fields up to 256 bytes in length can be exchanged.
2. With the XR instruction, the contents of two registers can be exchanged.
3. Because the X instruction operates storage to register only, an exchange cannot be made solely by the use of X.
4. A field EXCLUSIVE ORed with itself is cleared to zeros.
5. For additional examples of the use of EXCLUSIVE OR, see the section "Floating-Point-Number Conversion" later in this appendix.

EXECUTE (EX)

The EXECUTE instruction causes one target instruction in main storage to be executed out of sequence without actually branching to the target instruction. Unless the R_1 field of the EXECUTE instruction is zero, bits 8-15 of the target instruction are ORed with bits 24-31 of the R_1 register before the target instruction is executed. Thus, EXECUTE may be used to supply the length field for an SS instruction without modifying the SS instruction in storage. For example, assume that a MOVE (MVC) instruction is the target that is located at address 3820, with a format as follows:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D2	00	C	003	D	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
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, the rightmost byte of register 1 is ORed with the second byte of the target instruction:

Instruction byte:	0000 0000{2} = 00
Register byte:	0000 0011{2} = 03
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).

INSERT CHARACTERS UNDER MASK (ICM)

The INSERT CHARACTERS UNDER MASK (ICM) instruction may be used to replace all or selected bytes in a general register with bytes from storage and to set the condition code to indicate the value of the inserted field.

For example, if it is desired to insert a three-byte address from FIELDA into register 5 and leave the leftmost byte of the register unchanged, assume:

Machine Format

Op Code	R_1	M_3	S_2
BF	5	7	* * * *

Assembler Format

Op Code	R_1,M_3,S_2
ICM	5,B'0111',FIELDA

FIELDA:	FE DC BA
Register 5 (before):	12 34 56 78
Register 5 (after):	12 FE DC BA
Condition code (after):	1 (leftmost bit of inserted field is one)

As another example:

Machine Format

Op Code	R_1	M_3	S_2
BF	6	9	* * * *

Assembler Format

Op Code	R_1,M_3,S_2
ICM	6,B'1001',FIELDB

FIELDB:	12 34
Register 6 (before):	00 00 00 00
Register 6 (after):	12 00 00 34
Condition code (after):	2 (inserted field is nonzero with leftmost zero bit)

When the mask field contains 1111, the ICM instruction produces the same result as LOAD (L) (provided that the indexing capability of the RX format is not needed), except that ICM also sets the condition code. The condition-code setting is useful when an all-zero field (condition code 0) or a leftmost one bit (condition code 1) is used as a flag.

LOAD (L, LR)

The LOAD instruction takes four bytes from storage or from a general register and place them unchanged into a general register. For example, assume that the four bytes starting with location 21003 are to be loaded into register 10. Initially:

Register 5 contains 00 02 00 00.
Register 6 contains 00 00 10 03.
The contents of register 10 are not significant.
Storage locations 21003-21006 contain 00 00 AB CD.

To load register 10, the RX form of the instruction can be used:

Machine Format

Op Code	R_1	X_2	B_2	D_2
58	A	5	6	000

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
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 binary integer up to 4095{10} in a register without first defining a constant and then using it as an operand. For example, the following instruction places the number 2048{10} in register 1:

Machine Format

Op Code	R_1	X_2	B_2	D_2
41	1	0	0	800

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
LA	1,2048(0,0)

The LOAD ADDRESS instruction can also be used to increment a register by an amount up to 4095{10} specified in the D_2 field. Only the rightmost 24 bits of the sum are retained, however. The leftmost eight bits of the 32-bit result are set to zeros. For example, assume that register 5 contains 00 12 34 56.

The instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
41	5	0	5	00A

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
LA	5,10(0,5)

adds 10 (decimal) to the contents of register 5 as follows:

Register 5 (old): 00 12 34 56
D_2 field: 00 00 00 0A

Register 5 (new): 00 12 34 60

The register may be specified as either B_2 or X_2. Thus, the instruction LA 5,10(5,0) produces the same result.

As the most general example, the instruction LA 6,10(5,4) forms the sum of three values: the contents of register 4, the contents of register 5, and a displacement of 10 and places the 24-bit sum with eight zeros appended on the left in register 6.

LOAD HALFWORD (LH)

The LOAD HALFWORD instruction places unchanged a halfword from storage into the right half of a register. The left half of the register is loaded with zeros or ones according to the sign (leftmost bit) of the halfword.

For example, assume that the two bytes in storage locations 1803-1804 are to be loaded into register 6. Also assume:

The contents of register 6 are not significant.
Register 14 contains 00 00 18 03.
Locations 1803-1804 contain 00 20.

The instruction required to load the register is:

Machine Format

Op Code	R_1	X_2	B_2	D_2
48	6	0	E	000

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
LH	6,0(0,14)

After the instruction is executed, register 6 contains 00 00 00 20. If locations 1803-1804 had contained a negative number, for example, A7 B6, a minus sign would have been propagated to the left, giving FF FF A7 B6 as the final result in register 6.

MOVE (MVC, MVI)

MVC Example

The MOVE (MVC) instruction can be used to move data from one storage location to another. For example, assume that the following two fields are in storage:

	2048										2052
Field 1	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB

	3840								3848
Field 2	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:

	2048										2052
Field 1	F1	F2	F3	F4	F5	F6	F7	F8	C9	CA	CB

Field 2 is unchanged.

MVC can also be used to propagate a byte through a field by starting the first-operand field one byte location to the right of the second-operand field. For example, suppose that an area in storage starting with address 358 contains the following data:

358								360
00	F1	F2	F3	F4	F5	F6	F7	F8

With the following MVC instruction, the zeros in location 358 can be propagated throughout the entire field (assume that register 11 contains 00 00 03 58):

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 is executed as if one byte were processed at a time, the above instruction, in effect, 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.

MVI Example

The MOVE (MVI) instruction places one byte of information from the instruction stream into storage. For example, the instruction:

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'$'

may be used, in conjunction with the instruction EDIT AND MARK, to insert the EBCDIC code for a dollar symbol at the storage address contained in general register 1 (see also the example for EDIT AND MARK).

MOVE INVERSE (MVCIN)

The MOVE INVERSE (MVCIN) instruction can be used to move data from one storage location to another while reversing the order of the bytes within the field. For example, assume that the following two fields are in storage:

	2048										2052
Field 1	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB

	3840								3848
Field 2	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
E8	07	1	000	2	007

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
MVCIN	0(8,1),7(2)

After the instruction is executed, field 1 becomes:

	2048										2052
Field 1	F8	F7	F6	F5	F4	F3	F2	F1	C9	CA	CB

Field 2 is unchanged.

Note: This example uses the same general registers, storage locations, and original values as the first example for MVC. For MVCIN, the second-operand address must designate the rightmost byte of the field to be moved, in this case location 3847. This is accomplished by means of the 7 in the D_2 field of the instruction.

MOVE LONG (MVCL)

The MOVE LONG (MVCL) instruction can be used for moving data in storage as in the first example of the MVC instruction, provided that the two operands do not overlap. MVCL differs from MVC in that the address and length of each operand are specified in an even-odd pair of general registers. Consequently, MVCL can be used to move more than 256 bytes of data with one instruction. As an example, assume:

Register 2 contains 00 0A 00 00.
Register 3 contains 00 00 08 00.
Register 8 contains 00 06 00 00.
Register 9 contains 00 00 08 00.

Execution of the instruction:

Machine Format

Op Code	R_1	R_2
0E	8	2

Assembler Format

Op Code	R_1,R_2
MVCL	8,2

moves 2,048{10} bytes from locations A0000-A07FF to locations 60000-607FF. Bits 8-31 of registers 2 and 8 are incremented by 800{16}, and bits 0-7 of registers 2 and 8 are set to zeros. Bits 8-31 of registers 3 and 9 are decremented to zero. Condition code 0 is set to indicate that the operand lengths are equal.

If register 3 had contained F0 00 04 00, only the 1,024{10} bytes from locations A0000-A03FF would have been moved to locations 60000-603FF. The remaining locations 60400-607FF of the first operand would have been filled with 1,024 copies of the padding byte X'F0', as specified by the leftmost byte of register 3. Bits 8-31 of registers 2 and 8 would have been incremented by 400{16}, and bits 0-7 of registers 2 and 8 set to zeros. Bits 8-31 of registers 3 and 9 would still have been decremented to zero. Condition code 2 would have been set to indicate that the first operand was longer than the second.

The technique for setting a field to zeros that is illustrated in the second example of MVC cannot be used with MVCL. If the registers were set up to attempt such an operation with MVCL, no data movement would take place and condition code 3 would indicate destructive overlap.

Instead, MVCL may be used to clear a storage area to zeros as follows. Assume register 8 and 9 are set up as before. Register 3 contains only zeros, specifying zero length for the second operand and a zero padding byte. Register 2 is not used to access storage, and its contents are not significant. Executing the instruction MVCL 8,2 causes locations 60000-607FF to be filled with zeros. Bits 8-31 of register 8 are incremented by 800{16}, and bits 0-7 of registers 2 and 8 are set to zeros. Bits 8-31 of register 9 are decremented to zero, and condition code 2 is set to indicate that the first operand is longer than the second.

MOVE NUMERICS (MVN)

Two related instructions, MOVE NUMERICS and MOVE ZONES, may be used with decimal data in the zoned format to operate separately on the rightmost four bits (the numeric bits) and the leftmost four bits (the zone bits) of each byte. Both are similar to MOVE (MVC), except that MOVE NUMERICS moves only the numeric bits and MOVE ZONES moves only the zone bits.

To illustrate the operation of the MOVE NUMERICS instruction, assume that the following two fields are in storage:

	7090			7093
Field A	C6	C7	C8	C9

	7041					7046
Field B	F0	F1	F2	F3	F4	F5

Also assume:

Register 14 contains 00 00 70 90.
Register 15 contains 00 00 70 40.

After the instruction:

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 B becomes:

7041					7046
F6	F7	F8	F9	F4	F5

The numeric bits of the bytes at locations 7090-7093 have been stored in the numeric bits of the bytes at locations 7041-7044. The contents of locations 7090-7093 and 7045-7046 are unchanged.

MOVE WITH OFFSET (MVO)

MOVE WITH OFFSET may be used to shift a packed-decimal number an odd number of digit positions or to concatenate a sign to an unsigned packed-decimal number.

Assume that the three-byte unsigned packed-decimal number in storage locations 4500-4502 is to be moved to locations 5600-5603 and given the sign of the packed-decimal number ending at location 5603. Also assume:

Register 12 contains 00 00 56 00.
Register 15 contains 00 00 45 00.
Storage locations 5600-5603 contain 77 88 99 0C.
Storage locations 4500-4502 contain 12 34 56.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F1	3	2	C	000	F	000

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
MVO	0(4,12),0(3,15)

is executed, the storage locations 5600-5603 contain 01 23 45 6C. Note that the second operand is extended on the left with one zero to fill out the first-operand field.

MOVE ZONES (MVZ)

The MOVE ZONES instruction can operate on overlapping or nonoverlapping fields, as can the instructions MOVE (MVC) and MOVE NUMERICS. When operating on nonoverlapping fields, MOVE ZONES works like the MOVE NUMERICS instruction (see its example), except that MOVE ZONES moves only the zone bits of each byte. To illustrate the use of MOVE ZONES with overlapping fields, assume that the following data field is in storage:

800					805
F1	C2	F3	C4	F5	C6

Also assume that register 15 contains 00 00 08 00. The instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
D3	04	F	001	F	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
MVZ	1(5,15),0(15)

propagates the zone bits from the byte at address 800 through the entire field, so that the field becomes:

800					805
F1	F2	F3	F4	F5	F6

MULTIPLY (M, MR)

Assume that a number in register 5 is to be multiplied by the contents of a four-byte field at address 3750. Initially:

The contents of register 4 are not significant.
Register 5 contains 00 00 00 9A = 154{10} = the multiplicand.
Register 11 contains 00 00 06 00.
Register 12 contains 00 00 30 00.
Storage locations 3750-3753 contain 00 00 00 83 = 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, the product is in the register pair 4 and 5:

Register 4 contains 00 00 00 00.
Register 5 contains 00 00 4E CE = 20,174{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 01 00 05. The contents of register 6 are not significant. The instruction:

Machine Format

Op Code	R_1	R_2
1C	6	7

Assembler Format

Op Code	R_1,R_2
MR	6,7

multiplies the number in register 7 by itself and places the result in the pair of registers 6 and 7:

Register 6 contains 00 00 00 01.
Register 7 contains 00 0A 00 19.

MULTIPLY HALFWORD (MH)

The MULTIPLY HALFWORD instruction is used to multiply the contents of a register by a two-byte field in storage. For example, assume that:

Register 11 contains 00 00 00 15 =21{10} = the multiplicand.
Register 14 contains 00 00 01 00.

Register 15 contains 00 00 20 00.
Storage locations 2102-2103 contain FF D9 = -39{10} = the multiplier.

The instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
4C	B	E	F	002

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
OR MH	11,2(14,15)

multiplies the two numbers. The product, FF FF FC CD = -819{10}, replaces the original contents of register 11.

Only the rightmost 32 bits of a product are stored in a register; any significant bits on the left are lost. No program interruption occurs on overflow.

OR (O, OC, OI, OR)

When the Boolean operator OR is applied to two bits, the result is one when either bit is one; otherwise, the result is zero. When two bytes are ORed, each pair of bits is handled separately; there is no connection from one bit position to another. The following is an example of ORing two bytes:

```
First-operand byte:   0011 0101{2}
Second-operand byte:  0101 1100{2}
---------------------------------
Result byte:          0111 1101{2}
```

OI Example

A frequent use of the OR instruction is to set a particular bit to one. For example, assume that storage location 4891 contains 0100 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 (the I_2 field of the instruction):

```
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 back in location 4891. Condition code 1 is set.

PACK (PACK)

Assume that storage locations 1000-1003 contain the following zoned-decimal number that is to be converted to a packed-decimal number and left in the same location:

	1000			1003
Zoned number	F1	F2	F3	C4

Also assume that register 12 contains 00 00 10 00. After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F2	3	3	C	000	C	000

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
PACK	0(4,12),0(4,12)

is executed, the result in locations 1000-1003 is in the packed-decimal format:

	1000			1003
Packed number	00	01	23	4C

Notes:

1. This example illustrates the operation of PACK when the first- and second-operand fields overlap completely.

2. During the operation, the second operand was extended on the left with zeros.

SHIFT LEFT DOUBLE (SLDA)

The SHIFT LEFT DOUBLE instruction shifts the 63 numeric bits of an even-odd register pair to the left, leaving the sign bit unchanged. Thus, the instruction performs an algebraic left shift of a 64-bit signed binary integer.

For example, if the contents of registers 2 and 3 are:

```
00 7F 0A 72   FE DC BA 98 =
00000000 01111111 00001010 01110010
11111110 11011100 10111010 10011000{2}
```

The instruction:

Machine Format

Op Code	R_1		B_2	D_2
8F	2	////	0	01F

Assembler Format

Op Code	$R_1,D_2(B_2)$
SLDA	2,31(0)

results in registers 2 and 3 both being left-shifted 31 bit positions, so that their new contents are:

```
7F 6E 5D 4C   00 00 00 00 =
01111111 01101110 01011101 01001100
00000000 00000000 00000000 00000000{2}
```

Because significant bits are shifted out of bit position 1 of register 2, overflow is indicated by setting condition code 3, and, if the fixed-point-overflow mask bit in the PSW is one, a fixed-point-overflow program interruption occurs.

SHIFT LEFT SINGLE (SLA)

The SHIFT LEFT SINGLE instruction is similar to SHIFT LEFT DOUBLE, except that it shifts only the 31 numeric bits of a single register. Therefore, this instruction performs an algebraic left shift of a 32-bit signed binary integer.

For example, if the contents of register 2 are:

```
00 7F 0A 72 = 00000000 01111111 00001010
01110010{2}
```

The instruction:

Machine Format

Op Code	R_1		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 = 01111111 00001010 01110010
00000000{2}
```

Condition code 2 is set to indicate that the result is greater than zero.

If a left shift of nine places had been specified, a significant bit would have been shifted out of bit position 1. Condition code 3 would have been set to indicate this overflow and, if the fixed-point-overflow mask bit in the PSW were one, a fixed-point overflow interruption would have occurred.

STORE CHARACTERS UNDER MASK (STCM)

STORE CHARACTERS UNDER MASK (STCM) may be used to place selected bytes from a register into storage. For example, if it is desired to store a three-byte address from general register 8 into location FIELD3, assume:

Machine Format

Op Code	R_1	M_3	S_2
BE	8	7	* * * *

Register Format

Op Code	R_1,M_3,S_2
STCM	8,B'0111',FIELD3

Register 8:	12 34 56 78
FIELD3 (before):	not significant
FIELD3 (after):	34 56 78

As another example:

Machine Format

Op Code	R_1	M_3	S_2
BE	9	5	* * * *

Register Format

Op Code	R_1,M_3,S_2
STCM	9,B'0101',FIELD2

Register 9:	01 23 45 67
FIELD2 (before):	not significant
FIELD2 (after):	23 67

STORE MULTIPLE (STM)

Assume that the contents of general registers 14, 15, 0, and 1 are to be stored in consecutive four-byte fields starting with location 4050 and that:

Register 14 contains 00 00 25 63.
Register 15 contains 00 01 27 36.
Register 0 contains 12 43 00 62.
Register 1 contains 73 26 12 57.
Register 6 contains 00 00 40 00.
The initial contents of locations 4050-405F are not significant.

The STORE MULTIPLE instruction allows the use of just one instruction to store the contents of the four registers:

Machine Format

Op Code	R_1	R_3	B_2	D_2
90	E	1	6	050

Assembler Format

Op Code	$R_1,R_3,D_2(B_2)$
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 selected bits of a byte and sets the condition code accordingly. For example, assume that:

Storage location 9999 contains FB.
Register 7 contains 00 00 99 90.

Assume the instruction to be:

Machine Format

Op Code	I_2	B_1	D_1
91	C3	7	009

Assembler Format

Op Code	$D_1(B_1),I_2$
TM	9(7),B'11000011'

The instruction tests only those bits of the byte in storage for which the mask bits are ones:

```
FB   = 1111 1011{2}
Mask = 1100 0011{2}
--------------------
Test = 11xx xx11{2}
```

Condition code 3 is set: all selected bits in the test result are ones. (The bits marked "x" are ignored.)

If location 9999 had contained B9, the test would have been:

```
B9   = 1011 1001{2}
Mask = 1100 0011{2}
--------------------
Test = 10xx xx01{2}
```

Condition code 1 is set: the selected bits are both zeros and ones.

If location 9999 had contained 3C, the test would have been:

```
3C   = 0011 1100{2}
Mask = 1100 0011{2}
--------------------
Test = 00xx xx00{2}
```

Condition code 0 is set: all selected bits are zeros.

Note: Storage location 9999 remains unchanged.

TRANSLATE (TR)

The TRANSLATE instruction can be used to translate data from any character code to any other desired code, provided that each character code consists of eight bits or fewer. An appropriate translation table is required in storage.

In the following example, EBCDIC code is translated to ASCII code. The first step is to create a 256-byte table in storage locations 1000-10FF. This table contains the characters of the ASCII code in the sequence of the binary representation of the EBCDIC code; that is, the ASCII representation of a character is placed in storage at the starting address of the table plus the binary value of the EBCDIC representation of the same character.

For simplicity, the example shows only the part of the table containing the decimal digits:

10F0									10F9
30	31	32	33	34	35	36	37	38	39

Assume that the four-byte field at storage location 2100 contains the EBCDIC code for the digits 1984:

Locations 2100-2103 contain F1 F9 F8 F4.
Register 12 contains 00 00 21 00.
Register 15 contains 00 00 10 00.

As the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DC	03	C	000	F	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
TR	0(4,12),0(15)

is executed, the binary value of each EBCDIC byte is added to the starting address of the table, and the resulting address is used to fetch an ASCII byte:

Table starting address:	1000
First EBCDIC byte:	F1
Address of ASCII byte:	10F1

After execution of the instruction:

Locations 2100-2103 contain 31 39 38 34.

Thus, the ASCII code for the digits 1984 has replaced the EBCDIC code in the four-byte field at storage location 2100.

TRANSLATE AND TEST (TRT)

The TRANSLATE AND TEST instruction can be used to scan a data field for characters with a special meaning. To indicate which characters have a special meaning, a table similar to the one used for the TRANSLATE instruction is set up, except that zeros in the table indicate characters without any special meaning and nonzero values indicate characters with a special meaning.

The figure "Translate-and-Test Table" that follows has been set up to distinguish alphameric characters (A to Z and 0 to 9) from blanks, certain special symbols, and all other characters which are considered invalid. EBCDIC coding is assumed. The 256-byte table is assumed stored at locations 2000-20FF.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
200_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
201_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
202_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
203_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
204_	04	40	40	40	40	40	40	40	40	40	40	08	40	0C	10	40
205_	14	40	40	40	40	40	40	40	40	40	40	18	1C	20	40	40
206_	24	28	40	40	40	40	40	40	40	40	40	2C	40	40	40	40
207_	40	40	40	40	40	40	40	40	40	40	40	30	34	38	3C	40
208_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
209_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20A_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20B_	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
20C_	40	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20D_	40	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20E_	40	40	00	00	00	00	00	00	00	00	40	40	40	40	40	40
20F_	00	00	00	00	00	00	00	00	00	00	40	40	40	40	40	40

Note: If the character codes in the statement being translated occupy a range smaller than 00 through FF{16}, a table of fewer than 256 bytes can be used.

Translate and Test Table

The table entries for the alphameric characters in EBCDIC are 00; thus, the letter A (code C1) corresponds to byte location 20C1, which contains 00.

The 15 special symbols have nonzero entries from 04{16} to 3C{16} in increments of 4. Thus, the blank (code 40) has the entry 04{16}, the period (code 4B) has the entry 08{16}, and so on.

All other table positions have the entry 40{16} to indicate an invalid character.

The table entries are chosen so that they may be used to select one of a list of 16 words containing addresses of different routines to be entered for each special symbol or invalid character encountered during the scan.

Assume that this list of 16 branch addresses is stored at locations 3004-3043.

Starting at storage location CA80, there is the following sequence of 21{10} EBCDIC characters, where "b" stands for a blank.

Locations CA80-CA94:

UNPKbPROUT(9),WORD(5)

Also assume:

Register 1 contains 00 00 CA 7F.
Register 2 contains 00 00 30 00.
Register 15 contains 00 00 20 00.

As the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DD	14	1	001	F	000

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
TRT	1(21,1),0(15)

is executed, the value of the first source byte, the EBCDIC code for the letter U, is added to the starting address of the table to produce the address of the table entry to be examined:

Table starting address	2000
First source byte (U)	E4
Address of table entry	20E4

Because zeros were placed in storage location 20E4, no special action occurs. The operation continues with the second and subsequent source bytes until it reaches the blank in location CA84. When this symbol is reached, its value is added to the starting address of the table, as usual:

Table starting address	2000
Source byte (blank)	40
Address of table entry	2040

Because location 2040 contains a nonzero value, the following actions occur:

1. The address of the source byte, 00CA84, is placed in the rightmost 24 bits of register 1.

2. The table entry, 04, is placed in the rightmost eight bits of register 2, which now contains 00 00 30 04.

3. Condition code 1 is set (scan not completed).

The TRANSLATE AND TEST instruction may be followed by instructions to branch to the routine at the address found at location 3004, which corresponds to the blank character encountered in the scan. When this routine is completed, program control may return to the TRANSLATE AND TEST instruction to continue the scan, except that the length must first be adjusted for the characters already scanned.

For this purpose, the TRANSLATE AND TEST may be executed by the use of an EXECUTE instruction, which supplies the length specification from a general register. In this way, a complete statement scan can be performed with a single TRANSLATE AND TEST instruction used repeatedly by means of EXECUTE, and without modifying any instructions in storage. In the example, after the first execution of TRANSLATE AND TEST, register 1 contains the address of the last source byte translated. It is then a simple matter to subtract this address from the address of the last source byte (CA94) to produce a length specification. This length minus one is placed in the register that is referenced as the R1 field of the EXECUTE instruction. (Note that the length code in the machine format is one less than the total number of bytes in the field.) The second-operand address of the EXECUTE instruction points to the TRANSLATE AND TEST instruction, which is the same as illustrated above, except for the length (L) which is set to zero.

UNPACK (UNPK)

Assume that storage locations 2501-2502 contain a signed, packed-decimal number that is to be unpacked and placed in storage locations 1000-1004. Also assume:

Register 12 contains 00 00 10 00.
Register 13 contains 00 00 25 00.
Storage locations 2501-2502 contain 12 3D.
The initial contents of storage locations 1000-1004 are not significant.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F3	4	1	C	000	D	001

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
UNPK	0(5,12),1(2,13)

is executed, the storage locations 1000-1004 contain F0 F0 F1 F2 D3.

DECIMAL INSTRUCTIONS

(See Chapter 8 for a complete description of the decimal instructions.)

ADD DECIMAL (AP)

Assume that the signed, packed-decimal number at storage locations 500-503 is to be added to the signed, packed-decimal number at locations 2000-2002. Also assume:

Register 12 contains 00 00 20 00.
Register 13 contains 00 00 05 00.
Storage locations 2000-2002 contain 38 46 0D (a negative number).
Storage locations 500-503 contain 01 12 34 5C (a positive number).

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
FA	2	3	C	000	D	000

Assembler Format

Op Code $D_1(L_1,B_1),D_2(L_2,B_2)$

```
    AP    0(3,12),0(4,13)
```

is executed, the storage locations 2000-2002 contain 73 88 5C; condition code 2 is set to indicate that the result is greater than zero. Note that:

1. Because the two numbers had different signs, they were in effect subtracted.

2. Although the second operand is longer than the first operand, no overflow interruption occurs because the result can be entirely contained within the first operand.

COMPARE DECIMAL (CP)

Assume that the signed, packed-decimal contents of storage locations 700-703 are to be algebraically compared with the signed, packed-decimal contents of locations 500-502. Also assume:

Register 12 contains 00 00 06 00.
Register 13 contains 00 00 03 00.
Storage locations 700-703 contain 17 25 35 6D.
Storage locations 500-502 contain 72 14 2D.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F9	3	2	C	100	D	200

Assembler Format

Op Code $D_1(L_1,B_1),D_2(L_2,B_2)$

```
    CP   X'100'(4,12),X'200'(3,13)
```

is executed, condition code 1 is set, indicating that the first operand (the contents of locations 700-703) is less than the second.

DIVIDE DECIMAL (DP)

Assume that the signed, packed-decimal number at storage locations 2000-2004 (the dividend) is to be divided by the signed, packed-decimal number at locations 3000-3001 (the divisor). Also assume:

Register 12 contains 00 00 20 00.
Register 13 contains 00 00 30 00.
Storage locations 2000-2004 contain 01 23 45 67 8C.
Storage locations 3000-3001 contain 32 1D.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
FD	4	1	C	000	D	000

Assembler Format

Op Code $D_1(L_1,B_1),D_2(L_2,B_2)$

```
    DP    0(5,12),0(2,13)
```

is executed, the dividend is entirely replaced by the signed quotient and remainder, as follows:

	2000				2004
Locations 2000-2004	38	46	0D	01	8C
	quotient			remainder	

Notes:

1. Because the dividend and divisor have different signs, the quotient receives a negative sign.

2. The remainder receives the sign of the dividend and the length of the divisor.

3. If an attempt were made to divide the dividend by the one-byte field at location 3001, the quotient would be too long to fit within the four bytes allotted to it. A decimal-divide exception would exist, causing a program interruption.

EDIT (ED)

Before decimal data in the packed format can be used in a printed report, digits and signs must be converted to printable characters. Moreover, punctuation marks, such as commas and decimal points, may have to be inserted in appropriate places. The highly flexible EDIT instruction performs these functions in a single instruction execution.

This example shows step-by-step one way that the EDIT instruction can be used. The field to be edited (the source) is four bytes long; it is edited against a pattern 13 bytes long. The following symbols are used:

Symbol	Meaning
b (Hexadecimal 40)	Blank character
((Hexadecimal 21)	Significance starter
d (Hexadecimal 20)	Digit selector

Assume that register 12 contains:

    00 00 10 00

and that the source and pattern fields are:

Source

1200			1203
02	57	42	6C
			↑ +

Pattern

1000												100C
40	20	20	6B	20	21	20	4B	20	20	40	C3	D9
b	d	d	,	d	(	d	.	d	d	b	C	R

Execution of the instruction:

Machine Format

Op Code	L	B_1	D_1	B_2	D_2
DE	0C	C	000	C	200

Assembler Format

Op Code	$D_1(L,B_1),D_2(B_2)$
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,d(d.ddbCR
d	0	off/off	fill	bbd,d(d.ddbCR
d	2	off/on(2)	digit	bb2,d(d.ddbCR
,		on/on	leave	same
d	5	on/on	digit	bb2,5(d.ddbCR
(	7	on/on	digit	bb2,57d.ddbCR
d	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 is the fill byte.
2. First nonzero decimal source digit turns on significance indicator.
3. Plus sign in the four rightmost bits of the byte turns off significance indicator.

Thus, after the instruction is executed, the pattern field contains the result 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

This pattern field prints as:

2,574.26

The source field remains unchanged. Condition code 2 is set because the number was greater than zero.

If the number in the source field is changed to the negative number 00 00 02 6D and the original pattern is used, the edited result this time is:

Pattern

1000 100C

40	40	40	40	40	40	F0	4B	F2	F6	40	C3	D9
b	b	b	b	b	b	0	.	2	6	b	C	R

This pattern field prints as:

0.26 CR

The significance starter forces the significance indicator to the on state and hence causes a leading zero and the decimal point to be preserved. Because the minus-sign code has no effect on the significance indicator, the characters CR are printed to show a negative (credit) amount.

Condition code 1 is set (number less than zero).

EDIT AND MARK (EDMK)

The EDIT AND MARK instruction may be used, in addition to the functions of EDIT, to insert a currency symbol, such as a dollar sign, at the appropriate position in the edited result. Assume the same source in storage locations 1200-1203, the same pattern in locations 1000-100C, and the same contents of general register 12 as for the EDIT instruction above. The previous contents of general register 1 (GR1) are not significant; a LOAD ADDRESS instruction is used to set up the first digit position that is forced to print if no significant digits occur to the left.

The instructions:

```
LA   1,6(0,12)         Load address of
                       forced significant
                       digit into GR1
EDMK 0(13,12),X'200'(12) Leave address
                       of first signif-
                       icant digit in GR1
BCTR 1,0               Subtract 1 from
                       address in GR1
MVI  0(1),C'$'         Store dollar sign
                       at address in GR1
```

produce the following results for the two examples under EDIT:

Pattern

1000 100C

40	5B	F2	6B	F5	F7	F4	4B	F2	F6	40	40	40
b	$	2	,	5	7	4	.	2	6	b	b	b

This pattern field prints as:

    $2,574.26

Condition code 2 is set to indicate that the number edited was greater than zero.

Pattern

1000 100C

40	40	40	40	40	5B	F0	4B	F2	F6	40	C3	D9
b	b	b	b	b	$	0	.	2	6	b	C	R

This pattern field prints as:

    $0.26 CR

Condition code 1 is set because the number is less than zero.

MULTIPLY DECIMAL (MP)

Assume that the signed, packed-decimal number in storage locations 1202-1204 (the multiplicand) is to be multiplied by the signed, packed-decimal number in locations 500-501 (the multiplier).

	1202		1204
Multiplicand	38	46	0D

	500	501
Multiplier	32	1D

The multiplicand must first be extended to have at least two bytes of leftmost zeros, corresponding to the multiplier length, so as to avoid a data exception during the multiplication. ZERO AND ADD can be used to move the multiplicand into a longer field. Assume:

Register 4 contains 00 00 12 00.
Register 6 contains 00 00 05 00.

Then execution of the instruction:

```
ZAP X'100'(5,4),2(3,4)
```

sets up a new multiplicand in storage locations 1300-1304:

	1300				1304
Multiplicand (new)	00	00	38	46	0D

Now, after the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
FC	4	1	4	100	6	000

Assembler Format

Op Code $D_1(L_1,B_1),D_2(L_2,B_2)$

```
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 SHIFT AND ROUND DECIMAL (SRP) instruction can be used for shifting decimal numbers in storage to the left or right. When a number is shifted right, rounding can also be done.

Decimal Left Shift

In this example, the contents of storage location FIELD1 are shifted three places to the left, effectively multiplying the contents of FIELD1 by 1000. FIELD1 is six bytes long. The following instruction performs the operation:

Machine Format

Op Code	L_1	I_3	S_1	B_2	D_2
F0	5	0	****	0	003

Assembler Format

Op Code $S_1(L_1),S_2,I_3$

```
SRP  FIELD1(6),3,0
```

FIELD1 (before): 00 01 23 45 67 8C

FIELD1 (after): 12 34 56 78 00 0C

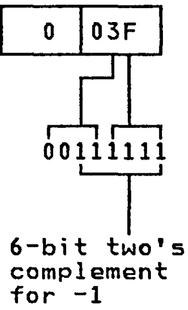
The second-operand address in this instruction specifies the shift amount (three places). The rounding digit, I_3, is not used in a left shift, but it must be a valid decimal digit. After execution, condition code 2 is set to show that the result is greater than zero.

Decimal Right Shift

In this example, the contents of storage location FIELD2 are shifted one place to the right, effectively dividing the contents of FIELD2 by 10 and discarding the remainder. FIELD2 is five bytes in length. The following instruction performs this operation:

Machine Format

Op Code	L_1	I_3	S_1	B_2	D_2
F0	4	0	****	0	03F



Assembler Format

Op Code	$S_1(L_1),S_2,I_3$
SRP	FIELD2(5),64-1,0

FIELD 2 (before): 01 23 45 67 8C

FIELD 2 (after): 00 12 34 56 7C

In the SRP instruction, shifts to the right are specified in the second-operand address by negative shift values, which are represented as a six-bit value in two's complement form.

The six-bit two's complement of a number, n, can be specified as 64 - n. In this example, a right shift of one is represented as 64 - 1.

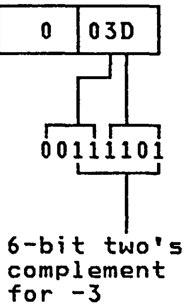
Condition code 2 is set.

Decimal Right Shift and Round

In this example, the contents of storage location FIELD3 are shifted three places to the right and rounded, in effect dividing by 1000 and rounding up. FIELD3 is four bytes in length.

Machine Format

Op Code	L_1	I_3	S_1	B_2	D_2
F0	3	5	****	0	03D



Assembler Format

Op Code	$S_1(L_1),S_2,I_3$
SRP	FIELD3(4),64-3,5

FIELD 3 (before): 12 39 60 0D

FIELD 3 (after): 00 01 24 0D

The shift amount (three places) is specified in the D_2 field. The I_3 field specifies a rounding digit of 5. The rounding digit is added to the last digit shifted out (which is a 6), and the carry is propagated to the left. The sign is ignored during the addition.

Condition code 1 is set because the result is less than zero.

Multiplying by a Variable Power of 10

Since the shift value specified by the SRP instruction specifies both the direction and amount of the shift, the operation is equivalent to multiplying the decimal first operand by 10 raised to the power specified by the shift value.

If the shift value is to be variable, it may be specified by the B_2 field instead of the displacement D_2 of the SRP instruction. The general register designated by B_2 should contain the shift value (power of 10) as a signed binary integer.

A fixed scale factor modifying the variable power of 10 may be specified by using both the B_2 field (variable part in a general register) and the D_2 field (fixed part in the displacement).

The SRP instruction uses only the rightmost six bits of the effective address $D_2(B_2)$ and interprets them as a six-bit signed binary integer to control the left or right shift as in the preceding shift examples.

ZERO AND ADD (ZAP)

Assume that the signed, packed-decimal number at storage locations 4500-4502 is to be moved to locations 4000-4004 with four leading zeros in the result field. Also assume:

Register 9 contains 00 00 40 00.
Storage locations 4000-4004 contain 12 34 56 78 90.
Storage locations 4500-4502 contain 38 46 0D.

After the instruction:

Machine Format

Op Code	L_1	L_2	B_1	D_1	B_2	D_2
F8	4	2	9	000	9	500

Assembler Format

Op Code	$D_1(L_1,B_1),D_2(L_2,B_2)$
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 without overflow.

Note that, because the first operand is not checked for valid sign and digit codes, it may contain any combination of hexadecimal digits before the operation.

FLOATING-POINT INSTRUCTIONS

(See Chapter 9 for a complete description of the floating-point instructions.)

In this section, the abbreviations FPR0, FPR2, FPR4, and FPR6 stand for floating-point registers 0, 2, 4, and 6 respectively.

ADD NORMALIZED (AD, ADR, AE, AER, AXR)

The ADD NORMALIZED instruction performs the addition of two floating-point numbers and places the normalized result in a floating-point register. Neither of the two numbers to be added must necessarily be in normalized form before addition occurs. For example, assume that:

FPR6 contains the unnormalized number C3 08 21 00 00 00 00 00 = -82.1{16} = -130.06{10} approximately.
Storage locations 2000-2007 contain the normalized number 41 12 34 56 00 00 00 00 = +1.23456{16} = +1.14{10} approximately.
Register 13 contains 00 00 20 00.

The instruction:

Machine Format

Op Code	R_1	X_2	B_2	D_2
7A	6	0	D	000

Assembler Format

Op Code	$R_1,D_2(X_2,B_2)$
AE	6,0(0,13)

performs the short-precision addition of the two operands, as follows.

The characteristics of the two numbers (43 and 41) are compared. Since the number in storage has a characteristic that is smaller by 2, it is right-shifted two hexadecimal digit positions. One guard digit is retained on the right. The fractions of the two numbers are then added algebraically:

	Fraction	GD[1]
FPR6	-43 08 21 00	
Shifted number from storage	+43 00 12 34	5
Intermediate sum	-43 08 0E CB	B
Left-shifted sum	-42 80 EC BB	

[1] Guard digit

Because the intermediate sum is unnormalized, it is left-shifted to form the normalized floating-point number -80.ECBB{16} = -128.92{10} approximately. Combining the sign with the characteristic, the result is C2 80 EC BB, which replaces the left half of FPR6. The right half of FPR6 and the contents of storage locations 2000-2007 are unchanged. Condition code 1 is set to indicate a result less than zero.

If the long-precision instruction AD were used, the result in FPR6 would be C2 80 EC BA A0 00 00 00. Note that use of the long-precision instruction would avoid a loss of precision in this example.

ADD UNNORMALIZED (AU, AUR, AW, AWR)

The ADD UNNORMALIZED instruction operates the same as the ADD NORMALIZED instruction, except that the final result is not normalized. For example, using the 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:

	Fraction	GD[1]
FPR6	-43 08 21 00	
Shifted number from storage	+43 00 12 34	5
Intermediate sum	-43 08 0E CB	B

[1] Guard digit

The guard digit participates in the addition but is discarded. The unnormalized sum replaces the left half of FPR6. Condition code 1 is set because the result is less than zero.

The truncated result in FPR6 (C3 08 0E CB 00 00 00 00) shows a loss of a significant digit when compared to the result of short-precision normalized addition.

COMPARE (CD, CDR, CE, CER)

Assume that FPR4 contains 43 00 00 00 00 00 00 00 (zero), and FPR6 contains 35 12 34 56 78 9A BC DE (a positive number). The contents of the two registers are to be compared using a long-precision COMPARE instruction.

Machine Format

Op Code	R_1	R_2
29	4	6

Assembler Format

Op Code R_1,R_2

CDR 4,6

The number with the smaller characteristic, which is in register FPR6, is right-shifted 43 - 35 hex (67 - 53 decimal) or 14 digit positions, so that the two characteristics agree. The shifted number is 43 00 00 00 00 00 00 00, with a guard digit of one. Therefore, when the two numbers are compared, condition code 1 is set, indicating that operand 1 in FPR4 is less than operand 2 in FPR6.

If the example is changed to a second operand with a characteristic of 34 instead of 35, so that FPR6 contains 34 12 34 56 78 9A BC DE, the operand is right-shifted 15 positions, leaving all fraction digits and the guard digit as zeros. Condition code 0 is set, indicating equality. This example shows that two floating-point numbers with different characteristics or fractions may compare equal if the numbers are unnormalized or zero.

As another example of comparing unnormalized floating-point numbers, 41 00 12 34 56 78 9A BC compares equal to all numbers of the form 3F 12 34 56 78 9A BC 0X (X represents any hexadecimal digit). When the COMPARE instruction is executed, the two rightmost digits are shifted right two places, the 0 becomes the guard digit, and the X does not participate in the comparison.

However, when two normalized floating-point numbers are compared, the relationship between numbers that compare equal is unique: each digit in one number must be the same as the corresponding digit in the other number.

DIVIDE (DD, DDR, DE, DER)

Assume that the first operand (the dividend) is in FPR2 and the second operand (the divisor) in FPR0. If the operands are in the short-precision format, the resulting quotient is returned to FPR2 by the instruction:

Machine Format

Op Code	R_1	R_2
3D	2	0

Assembler Format

Op Code R_1,R_2

DER 2,0

Several examples of short-precision floating-point division, with the dividend in FPR2 and the divisor in FPR0, are shown below. For case A, the result, which replaces the dividend, is obtained in the following steps.

```
                7.2522F
.123400|.821000
         7F6C00
         ------
          2A400 0
          24680 0
          -------
           5D80 00
           5B04 00
           -------
            27C 000
            246 800
            -------
             35 8000
             24 6800
             -------
             11 18000
             11 10C00
             --------
                 7400
```

Case	FPR2 Before (Dividend)	FPR0 (Divisor)	FPR2 After (Quotient)
A	-43 082100	+43 001234	-42 72522F
B	+42 101010	+45 111111	+3D F0F0F0
C	+48 30000F	+41 400000	+47 C0003C
D	+48 30000F	+41 200000	+48 180007
E	+48 180007	+41 200000	+47 C00038

Case C shows a number being divided by 4.0. Case D divides the same number by 2.0, and case E divides the result of case D again by 2.0. The results of cases C and E differ in the rightmost hexadecimal digit position, which illustrates an effect of result truncation.

HALVE (HDR, HER)

HALVE produces the same result as floating-point DIVIDE with a divisor of 2.0. Assume FPR2 contains the long-precision number +48 30 00 00 00 00 00 0F. The following HALVE instruction produces the result +48 18 00 00 00 00 00 07 in FPR2:

Machine Format

Op Code	R_1	R_2
24	2	2

Assembler Format

Op Code	R_1,R_2
HDR	2,2

MULTIPLY (MD, MDR, ME, MER, MXD, MXDR, MXR)

For this example, the following long-precision operands are in FPR0 and FPR2:

```
FPR0:  -33 606060 60606060
FPR2:  -5A 200000 20000020
```

A long-precision product is generated by the instruction:

Machine Format

Op Code	R_1	R_2
2C	0	2

Assembler Format

Op Code	R_1,R_2
MDR	0,2

If the operands were not already normalized, the instruction would first normalize them. It then generates an intermediate result consisting of the full 28-digit hexadecimal product fraction obtained by multiplying the 14-digit hexadecimal operand fractions, together with the appropriate sign and a characteristic that is the sum of the operand characteristics less 64 (40 hex):

The fraction multiplication is performed as follows:

```
                 .60606060606060
                 .20000020000020
                 ---------------
                 C0C0C0C0C0C0C00
           C0C0C0C0C0C0C0
     C0C0C0C0C0C0C0
  ------------------------------
  .0C0C0C181818241818180C0C0C00
```

Attaching the sign and characteristic to the fraction gives:

```
+4D 0C0C0C 18181824 1818180C 0C0C00
```

Because this intermediate product has a leading zero, it is then normalized. The truncated final result placed in FPR0 is:

```
+4C C0C0C1 81818241
```

FLOATING-POINT-NUMBER CONVERSION

The following examples illustrate one method of converting between binary fixed-point numbers (32-bit signed binary integers) and normalized floating-point numbers. Conversion must provide for the different representations used with negative numbers: the two's-complement form for signed binary integers, and the signed-absolute-value form for the fractions of floating-point numbers.

Fixed Point to Floating Point

The method used here inverts the leftmost bit of the 32-bit signed binary integer, which is equivalent to adding 2^{31} to the number and considering the result to be positive. This changes the number from a signed integer in the range $2^{31} - 1$ through -2^{31} to an unsigned integer in the range $2^{32} - 1$ through 0. After conversion to the long floating-point format, the value 2^{31} is subtracted again.

Assume that general register 9 (GR9) contains the integer -59 in two's-complement form:

```
     GR9:      FF FF FF C5
```

Further, assume two eight-byte fields in storage: TEMP, for use as temporary storage, and TWO31, which contains the floating-point constant 2^{31} in the following format:

```
     TWO31:   4E 00 00 00 80 00 00 00
```

This is an unnormalized long floating-point number with the characteristic 4E, which corresponds to a radix point (hexadecimal point) to the right of the number.

The following instruction sequence performs the conversion:

```
                         Result

X   9,TWO31+4            GR9:
                         7FFF FFC5

ST  9,TEMP+4             TEMP:
                         xxxx xxxx 7FFF FFC5

MVC TEMP(4),TWO31        TEMP:
                         4E00 0000 7FFF FFC5

LD  2,TEMP               FPR2:
                         4E00 0000 7FFF FFC5

SD  2,TWO31              FPR2:
                         C23B 0000 0000 0000
```

The EXCLUSIVE OR (X) instruction inverts the leftmost bit in general register 9, using the right half of the constant as the source for a leftmost one bit. The next two instructions assemble the modified number in an unnormalized long floating-point format, using the left half of the constant as the plus sign, the characteristic, and the leading zeros of the fraction. LOAD (LD) places the number unchanged in floating-point register 2. The SUBTRACT NORMALIZED (SD) instruction performs the final two steps by subtracting 2^{31} in floating-point form and normalizing the result.

Floating Point to Fixed Point

The procedure described here consists basically in reversing the steps of the previous procedure. Two additional considerations must be taken into account. First: the floating-point number may not be an exact integer. Truncating the excess hexadecimal digits on the right requires shifting the number one digit position farther to the right than desired for the final result, so that the units digit occupies the position of the guard digit. Second: the floating-point number may have to be tested as to whether it is outside the range of numbers representable as a 32-bit signed binary integer.

Assume that floating-point register 6 contains the number 59.25{10} = 3B.4{16} in normalized form:

```
     FPR6:     42 3B 40 00 00 00 00 00
```

Further, assume three eight-byte fields in storage: TEMP, for use as temporary storage, and the constants 2^{32} (TWO32) and 2^{31} (TWO31R) in the following formats:

```
     TWO32:    4E 00 00 01 00 00 00 00
     TWO31R:   4F 00 00 00 08 00 00 00
```

The constant TWO31R is shifted right one more position than the constant TWO31 of the previous example, so as to force the units digit into the guard-digit position.

The following instruction sequence performs the integer truncation, range tests, and conversion to a signed binary integer in general register 8 (GR8):

```
                         Result

SD  6,TWO31R             FPR6:
                         C87F FFFF C500 0000
BC  11,OVERFLOW          Branch to overflow
                           routine if result
                           is greater than or
                           equal to zero
AW  6,TWO32              FPR6:
                         4E00 0000 8000 003B
BC  4,OVERFLOW           Branch to overflow
                           routine if result
                           is less than zero
STD 6,TEMP               TEMP:
                         4E00 0000 8000 003B
XI  TEMP+4,X'80'         TEMP:
                         4E00 0000 0000 003B
L   8,TEMP+4             GR8:
                         0000 003B
```

The SUBTRACT NORMALIZED (SD) instruction shifts the fraction of the number to the right until it lines up with TWO31R, which causes the fraction digit 4 to fall to the right of the guard digit and be lost; the result of subtracting 2^{31} from the remaining digits is renormalized. The result should be less than zero; if not, the original number was too large in the positive direction. The first BRANCH ON CONDITION (BC) performs this test.

The ADD UNNORMALIZED (AW) instruction adds 2^{32}: 2^{31} to correct for the previous subtraction and another 2^{31} to change to an all-positive range. The second BC tests for a result less than zero, showing that the original number was too large in the negative direction. The unnormalized result is placed in temporary storage by the STORE (STD) instruction. There the leftmost bit of the binary integer is inverted by the EXCLUSIVE OR (XI) instruction to subtract 2^{31} and thus convert the unsigned number to the signed format. The final result is loaded into GR8.

MULTIPROGRAMMING AND MULTIPROCESSING EXAMPLES

When two or more programs sharing common storage locations are being executed concurrently in a multiprogramming or multiprocessing environment, one program may, for example, set a flag bit in the common-storage area for testing by another program. It should be noted that the instructions AND (NI or NC), EXCLUSIVE OR (XI or XC), and OR (OI or OC) could be used to set flag bits in a multiprogramming environment; but the same instructions may cause program logic errors in a multiprocessing configuration where two or more CPUs can fetch, modify, and store data in the same storage locations simultaneously.

EXAMPLE OF A PROGRAM FAILURE USING OR IMMEDIATE

Assume that two independent programs try to set different bits to one in a common byte in storage. The following example shows how the use of the instruction OR immediate (OI) can fail to accomplish this, if the programs are executed simultaneously on two different CPUs. One of the possible error situations is depicted.

Execution of instruction OI FLAGS,X'01' on CPU A	FLAGS	Execution of instruction OI FLAGS,X'80' on CPU B
	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 data is now invalid.

The COMPARE AND SWAP instruction has been provided to overcome this and similar problems.

CONDITIONAL SWAPPING INSTRUCTIONS (CS, CDS)

The COMPARE AND SWAP (CS) and COMPARE DOUBLE AND SWAP (CDS) instructions can be used in multiprogramming or multiprocessing environments to serialize access to counters, flags, control words, and other common storage areas.

The following examples of the use of the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions illustrate the applications for which the instructions are intended. It is important to note that these are examples of functions that can be performed by programs while the CPU is enabled for interruption (multiprogramming) or by programs that are being executed in a multiprocessing configuration. That is, the routine allows a program to modify the contents of a storage location while the CPU is enabled, even though the routine may be interrupted by another program on the same CPU that will update the location, and even though the possibility exists that another CPU may simultaneously update the same location.

The COMPARE AND SWAP instruction first checks the value of a storage location and then modifies it only if the value is what the program expects; normally this would be a previously fetched value. If the value in storage is not what the program expects, then the location is not modified; instead, the current value of the location is loaded into a general register, in preparation for the program to loop back and try again. During the execution of COMPARE AND SWAP, no other CPU can perform a store access or interlocked-update access at the specified location.

Setting a Single Bit

The following instruction sequence shows how the COMPARE AND SWAP instruction can be used to set a single bit in storage to one. Assume that the first byte of a word in storage called "WORD" contains eight flag bits.

```
      LA  6,X'80'    Put bit to be ORed
                       into GR6
      SLL 6,24       Shift left 24 places
                       to align the byte
                       to be ORed with
                       the location of
                       the flag bits
                       within WORD
      L   7,WORD     Fetch current flag
                       values
RETRY LR  8,7        Load flags into GR8
      OR  8,6        Set bit to one
      CS  7,8,WORD   Store new flags if
                       current flags un-
                       changed, or re-
                       fetch current
                       flag values if
                       changed
      BC  4,RETRY    If new flags are not
                       stored, try again
```

The format of the COMPARE AND SWAP instruction is:

Machine Format

Op Code	R_1	R_3	S_2
BA	7	8	****

Assembler Format

```
Op Code  R₁,R₃,S₂
  CS     7,8,WORD
```

The COMPARE AND SWAP instruction compares the first operand (general register 7 containing the current flag values) to the second operand in storage (WORD) while no CPU other than the one executing the COMPARE AND SWAP instruction is permitted to perform a store access or interlocked-update access at the specified storage location.

If the comparison is successful, indicating that the flag bits have not been changed since they were fetched, the modified copy in general register 8 is stored into WORD. If the flags have been changed, the compare will not be successful, and their new values are loaded into general register 7.

The conditional branch (BC) instruction tests the condition code and reexecutes the flag-modifying instructions if the COMPARE AND SWAP instruction indicated an unsuccessful comparison (condition code 1). When the COMPARE AND SWAP instruction is successful (condition code 0), the flags contain valid data, and the program exits from the loop.

The branch to RETRY will be taken only if some other program modifies the contents of WORD. This type of a loop differs from the typical "bit-spin" loop. In a bit-spin loop, the program continues to loop until the bit changes. In this example, the program continues to loop only if the value does change during each iteration. If a number of CPUs simultaneously attempt to modify a single location by using the sample instruction sequence, one CPU will fall through on the first try, another will loop once, and so on until all CPUs have succeeded.

Updating Counters

In this example, a 32-bit counter is updated by a program using the COMPARE AND SWAP instruction to ensure that the counter will be correctly updated. The original value of the counter is obtained by loading the word containing the counter into general register 7. This value is moved into general register 8 to provide a modifiable copy, and general register 6 (containing an increment to the counter) is added to the modifiable copy to provide the updated counter value. The COMPARE AND SWAP instruction is used to ensure valid storing of the counter.

The program updating the counter checks the result by examining the condition code. The condition code 0 indicates a successful update, and the program can proceed. If the counter had been changed between the time that the program loaded its original value and the time that it executed the COMPARE AND SWAP instruction, the execution would have loaded the new counter value into general register 7 and set the condition code to 1, indicating an unsuccessful update. The program must then repeat the update sequence until the execution of the COMPARE AND SWAP instruction results in a successful update.

The following instruction sequence performs the above procedure:

```
     LA   6,1       Put increment (1) into
                      GR6
     L    7,CNTR    Put original counter
                      value into GR7
LOOP LR   8,7       Set up copy in GR8 to
                      modify
     AR   8,6       Increment copy
     CS   7,8,CNTR  Update counter in
                      storage
     BC   4,LOOP    If original value had
                      changed, update new
                      value
```

The following shows two CPUs, A and B, executing this instruction sequence simultaneously: both CPUs attempt to add one to CNTR.

CPU A			CPU B		Comments
GR7	GR8	CNTR	GR7	GR8	
		16			
16	16				CPU A loads GR7 and GR8 from CNTR
			16	16	CPU B loads GR7 and GR8 from CNTR
				17	CPU B adds one to GR8
	17				CPU A adds one to GR8
		17			CPU A executes CS; successful match, store
			17		CPU B executes CS; no match, GR7 changed to CNTR value
				18	CPU B loads GR8 from GR7, adds one to GR8
		18			CPU B executes CS; successful match, store

BYPASSING POST AND WAIT

BYPASS POST Routine

The following routine allows the SVC "POST" as used in OS/VS to be bypassed whenever the corresponding WAIT has not yet been executed, provided that the supervisor WAIT and POST routines use COMPARE AND SWAP to manipulate event control blocks (ECBs).

Initial Conditions:

GR0 contains the POST code.
GR1 contains the address of the ECB.
GR5 contains 40 00 00 00{16}

```
HSPOST  OR    0,5         Set bit 1 of
                            GR1 to one
        L     3,0(1)      GR3 = contents
                            of ECB
        LTR   3,3         ECB marked
                            'waiting'?
        BC    4,PSVC      Yes, execute
                            post
                            SVC
        CS    3,0,0(1)    No, store post
                            code
        BC    8,EXITHP    Continue
PSVC    POST (1),(0)      ECB address is
                            in GR1, post
                            code in GR0
EXITHP  [Any instruction]
```

The following routine may be used in place of the previous HSPOST routine if it is assumed that bit 1 of the contents of GR0 is already set to one and if the ECB is assumed to contain zeros when it is not marked "WAITING."

```
HSPOST  SR    3,3
        CS    3,0,0(1)
        BC    8,EXITHP
        POST (1),(0)
EXITHP  [Any instruction]
```

BYPASS WAIT Routine

A BYPASS WAIT function, corresponding to the BYPASS POST, does not use the CS instruction, but the FIFO LOCK/UNLOCK routines which follow assume its use.

```
HSWAIT  TM    0(1),X'40'
        BC    1,EXITHW  If bit 1 is one,
                        then ECB is al-
                        ready posted;
                        branch to exit
        WAIT ECB=(1)
EXITHW  [Any instruction]
```

LOCK/UNLOCK

When a common storage area larger than a doubleword is to be updated, it is usually necessary to provide special interlocks to ensure that a single program at a time updates the common area. Such an area is called a serially reusable resource (SRR).

In general, updating a list, or even scanning a list, cannot be safely accomplished without first "freezing" the list. However, the COMPARE AND SWAP and COMPARE DOUBLE AND SWAP instructions can be used in certain restricted situations to perform queuing and list manipulation. Of prime importance is the capability to perform the lock/unlock functions and to provide sufficient queuing to resolve contentions, either in a LIFO or FIFO manner. The lock/unlock functions can then be used as the interlock mechanism for updating an SRR of any complexity.

The lock/unlock functions are based on the use of a "header" associated with the SRR. The header is the common starting point for determining the states of the SRR, either free or in use, and also is used for queuing requests when contentions occur. Contentions are resolved using WAIT and POST. The general programming technique requires that the program that encounters a "locked" SRR must "leave a mark on the wall" indicating the address of an ECB on which it will WAIT. The "unlocking" program sees the mark and posts the ECB, thus permitting the waiting program to continue. In the two examples given, all programs using a particular SRR must use either the LIFO queuing scheme or the FIFO scheme; the two cannot be mixed. When more complex queuing is required, it is suggested that the queue for the SRR be locked using one of the two methods shown.

LOCK/UNLOCK with LIFO Queuing for Contentions

The header consists of a word, that is, a four-byte field aligned on a word boundary. The word can contain zero, a positive value, or a negative value.

- A zero value indicates that the serially reusable resource (SRR) is free.
- A negative value indicates that the SRR is in use but no additional programs are waiting for the SRR.
- A positive value indicates that the SRR is in use and that one or more additional programs are waiting for the SRR. Each waiting program is identified by an element in a chained list. The positive value in the header is the address of the element most recently added to the list.

Each element consists of two words. The first word is used as an ECB; the second word is used as a pointer to the next element in the list. A negative value in a pointer indicates that the element is the last element in the list. The element is required only if the program finds the SRR locked and desires to be placed in the list.

The following chart describes the action taken for LIFO LOCK and LIFO UNLOCK routines. The routines following the chart allow enabled code to perform the actions described in the chart.

Function	Action		
	Header Contains Zero	Header Contains Positive Value	Header Contains Negative Value
LIFO LOCK (the incoming element is at location A)	SRR is free. Set the header to a negative value. Use the SRR.	SRR is in use. Store the contents of the header into location A+4. Store address A into the header. WAIT; the ECB is at location A.	
LIFO UNLOCK	Error	Some program is waiting for the SRR. Move the pointer from the "last in" element into the header. POST; the ECB is in the "last in" element.	The list is empty. Store zeros into the header. The SRR is free.

LIFO LOCK Routine:

Initial Conditions:

GR1 contains the address of the incoming element.
GR2 contains the address of the header.

```
LLOCK  SR    3,3       GR3 = 0
       ST    3,0(1)    Initialize the ECB
       LNR   0,1       GR0 = a negative
                         value
TRYAGN CS    3,0,0(2)  Set the header to
                         a negative value
                         if the header
                         contains zeros
       BC    8,USE     Did the header
                         contain zeros?
       ST    3,4(1)    No, store the
                         value of the
                         header into the
                         pointer in the
                         incoming element
       CS    3,1,0(2)  Store the address
                         of the incoming
                         element into the
                         header
       LA    3,0(0)    GR3 = 0
       BC    7,TRYAGN  Did the header get
                         updated?
       WAIT  ECB=(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 neg-
       BC    4,B         ative value?
       L     0,4(1)    No, load the
       CS    1,0,0(2)    pointer from the
                         "last in" element
                         and store it in
                         the header
       BC    7,A       Did the header get
                         updated?
       POST  (1)       Yes, post the "last
                         in" element
       BC    15,EXIT   Continue
B      SR    0,0       The header contains
       CS    1,0,0(2)    a negative value;
                         free the header
       BC    7,A         and continue
EXIT   [Any instruction]
```

Note that the LOAD instruction L 1,0(2) at location LUNLK would have to be CS 1,1,0(2) if it were not for the rule concerning storage-operand consistency. This rule requires the LOAD instruction to fetch a four-byte operand aligned on a word boundary such that, if another CPU changes the word being fetched by an operation which is also at least word-consistent, either the entire new or the entire old value of the word is obtained, and not a combination of the two. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

LOCK/UNLOCK with FIFO Queuing for Contentions

The header always contains the address of the most recently entered element. The header is originally initialized to contain the address of a posted ECB. Each program using the serially reusable resource (SRR) must provide an element regardless of whether contention occurs.

Each program then enters the address of the element which it has provided into the header, while simultaneously it removes the address previously contained in the header. Thus, associated with any particular program attempting to use the SRR are two elements, called the "entered element" and the "removed element." The "entered element" of one program becomes the "removed element" for the immediately following program. Each program then waits on the removed element, uses the SRR, and then posts the entered element.

When no contention occurs, that is, when the second program does not attempt to use the SRR until after the first program is finished, then the POST of the first program occurs before the WAIT of the second program. In this case, the bypass-post and bypass-wait routines described in the preceding section are applicable. For simplicity, these two routines are shown only by name rather than as individual instructions.

In the example, the element need be only a single word, that is, an ECB. However, in actual practice, the element could be made larger to include a pointer to the previous element, along with a program identification. Such information would be useful in an error situation to permit starting with the header and chaining through the list of elements to find the program currently holding the SRR.

It should be noted that the element provided by the program remains pointed to by the header until the next program attempts to lock. Thus, in general, the entered element cannot be reused by the program. However, the removed element is available, so each program gives up one element and gains a new one. It is expected that the element removed by a particular program during one use of the SRR would then be used by that program as the entry element for the next request to the SRR.

It should be noted that, since the elements are exchanged from one program to the next, the elements cannot be allocated from storage that would be freed and reused when the program ends. It is expected that a program would obtain its first element and release its last element by means of the routines described in the section "Free-Pool Manipulation" in this appendix.

The following chart describes the action taken for FIFO LOCK and FIFO UNLOCK.

Function	Action
FIFO LOCK (the incoming element is at location A)	Store address A into the header. WAIT; the ECB is at the location addressed by the old contents of the header.
FIFO UNLOCK	POST; the ECB is at location A.

The following routines allow enabled code to perform the actions described in the previous chart.

<u>FIFO</u> <u>LOCK</u> <u>Routine</u>:

Initial conditions:

GR3 contains the address of the header.
GR4 contains the address, A, of the element currently owned by this program. This element becomes the entered element.

```
FLOCK  LR    2,4         GR2 now contains
                           address of ele-
                           ment to be
                           entered
       SR    1,1         GR1 = 0
       ST    1,0(2)      Initialize the ECB
       L     1,0(3)      GR1 = contents of
                           the header
TRYAGN CS    1,2,0(3)    Enter address A
                           into header
       BC    7,TRYAGN      while remember-
                           ing old contents
                           of header into
                           GR1; GR1 now
                           contains address
                           of removed
                           element
       LR    4,1         Removed element
                           becomes new cur-
                           rently owned
                           element
       HSWAIT            Perform bypass-
                           wait routine; if
                           ECB already
                           posted, con-
                           tinue; if not,
                           wait; GR1 con-
                           tains the ad-
                           dress of the ECB
USE    [Any instruction]
```

<u>FIFO</u> <u>UNLOCK</u> <u>Routine</u>:

Initial conditions:

GR2 contains the address of the removed element, obtained during the FLOCK routine.
GR5 contains 40 00 00 00{16}

```
FUNLK    LR    1,2  Place address of en-
                      tered element in
                      GR1; GR1 = address
                      of ECB to be posted
         SR    0,0  GR0 = 0; GR0 has a
                      post code of zero
         OR    0,5  Set bit 1 of GR0 to
                      one
         HSPOST     Perform bypass-post
                      routine; if ECB has
                      not been waited on,
                      then mark posted
                      and continue; if it
                      has been waited on,
                      then post
CONTINUE [Any instruction]
```

FREE-POOL MANIPULATION

It is anticipated that a program will need to add and delete items from a free list without using the lock/unlock routines. This is especially likely since the lock/unlock routines require storage elements for queuing and may require working storage. The lock/unlock routines discussed previously allow simultaneous lock routines but permit only one unlock routine at a time. In such a situation, multiple additions and a single deletion to the list may all occur simultaneously, but multiple deletions cannot occur at the same time. In the case of a chain of pointers containing free storage buffers, multiple deletions along with additions can occur simultaneously. In this case, the removal cannot be done using the COMPARE AND SWAP instruction without a certain degree of exposure.

Consider a chained list of the type used in the LIFO lock/unlock example. Assume that the first two elements are at locations A and B, respectively. If one program attempted to remove the first element and was interrupted between the fourth and fifth instructions of the LUNLK routine, the list could be changed so that elements A and C are the first two elements when the interrupted program resumes execution. The COMPARE AND SWAP instruction would then succeed in storing the value B into the header, thereby destroying the list.

The probability of the occurrence of such list destruction can be reduced to *near* zero by appending to the header a counter that indicates the number of times elements have been added to the list. The use of a 32-bit counter guarantees that the list will not be destroyed unless the following events occur, in the exact sequence:

1. An unlock routine is interrupted between the fetch of the pointer from the first element and the update of the header.

2. The list is manipulated, including the deletion of the element referenced in 1, and exactly $2^{32}-1$ 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 1 is added to the list.

4. The unlock routine interrupted in 1 resumes execution.

The following routines use such a counter in order to allow multiple, simultaneous additions and removals at the head of a chain of pointers.

The list consists of a doubleword header and a chain of elements. The first word of the header contains a pointer to the first element in the list. The second word of the header contains a 32-bit counter indicating the number of additions that have been made to the list. Each element contains a pointer to the next element in the list. A zero value indicates the end of the list.

The following chart describes the free-pool-list manipulation.

Function	Action	
	Header = 0,Count	Header = A,Count
ADD TO LIST (the incoming element is at location A)	Store the first word of the header into location A. Store the address A into the first word of the header. Decrement the second word of the header by one.	
DELETE FROM LIST	The list is empty.	Set the first word of the header to the value of the contents of location A. Use element A.

The following routines allow enabled code to perform the free-pool-list manipulation described in the above chart.

ADD TO FREE LIST Routine:

Initial Conditions:

GR2 contains the address of the element to be added.
GR4 contains the address of the header.

```
ADDQ   LM   0,1,0(4) GR0,GR1 = contents
                       of the header
TRYAGN ST   0,0(2)   Point the new ele-
                       ment to the top
                       of the list
       LR   3,1      Move the count to
                       GR3
       BCTR 3,0      Decrement the count
       CDS  0,2,0(4) Update the header
       BC   7,TRYAGN
```

DELETE FROM FREE LIST Routine:

Initial conditions:

GR4 contains the address of the header.

```
DELETQ LM   2,3,0(4)       GR2,GR3 = con-
                             tents of the
                             header
TRYAGN LTR  2,2            Is the list
                             empty?
       BC   8,EMPTY        Yes, get help
       L    0,0(2)         No, GR0 = the
                             pointer from
                             the first
                             element
       LR   1,3            Move the count
                             to GR1
       CDS  2,0,0(4)       Update the
                             header
       BC   7,TRYAGN
USE    [Any instruction] The address of
                             the removed
                             element is in
                             GR2
```

Note that the LM (LOAD MULTIPLE) instructions at locations ADDQ and DELETQ would have to be CDS (COMPARE DOUBLE AND SWAP) instructions if it were not for the rule concerning storage-operand consistency. This rule requires the LOAD MULTIPLE instructions to fetch an eight-byte operand aligned on a doubleword boundary such that, if another CPU changes the doubleword being fetched by an operation which is also at least doubleword-consistent, either the entire new or the entire old value of the doubleword is obtained, and not a combination of the two. (See the section "Storage-Operand Consistency" in Chapter 5, "Program Execution.")

APPENDIX B. LISTS OF INSTRUCTIONS

The following figures list instructions by name, mnemonic, operation code, and facility. Some models may offer instructions that do not appear in the figures, such as those provided for assists or as part of special or custom features.

The operation codes for the vector facility are not included in this appendix. See the publication _IBM System/370 Vector Operations_, SA22-7125, for operation codes associated with this facility.

The operation code 00 hex with a two-byte instruction format is allocated for use by the program when an indication of an invalid operation is required. It is improbable that this operation code will ever be assigned to an instruction implemented in the CPU.

Explanation of Symbols in "Characteristics" and "Op Code" Columns

¢ Causes serialization and checkpoint synchronization.

¢^{1} Causes serialization and checkpoint synchronization when the M_1 and R_2 fields contain all ones and all zeros, respectively.

$ Causes serialization.

* The handling of bits 8-15 of the operation code for some of the I/O instructions depends on the instruction and the facilities installed. See the description of the instruction for details.

A Access exceptions for logical addresses.

A^1 Access exceptions; not all access exceptions may occur; see instruction description for details.

AI Access exceptions for instruction address.

AS Access exceptions and ASN-translation-specification exception; see instruction description for details.

AT ASN-translation exceptions (which include addressing, ASN-translation specification, AFX translation, and ASX translation).

B PER branch event.

BS Branch-and-save facility.

C Condition code is set.

CK CPU-timer and clock-comparator facility.

CS Channel-set-switching facility.

D Data exception.

DC Direct-control facility.

DF Decimal-overflow exception.

DK Decimal-divide exception.

DM Depending on the model, DIAGNOSE may generate various program exceptions and may change the condition code.

DU Dual-address-space facility.

EF Extended facility.

EK Storage-key-instruction-extension facility.

EO Exponent-overflow exception.

EU Exponent-underflow exception.

EX Execute exception.

FK Floating-point-divide exception.

FP Floating-point facility.

G0 Instruction execution includes the implied use of general register 0.

G1 Instruction execution includes the implied use of general register 1.

G2 Instruction execution includes the implied use of general register 2.

GM Instruction execution includes the implied use of multiple general registers.

IF Fixed-point-overflow exception.

II Interruptible instruction.

IK Fixed-point-divide exception.

L New condition code is loaded.

LS Significance exception.

MI Move-inverse facility.

MO Monitor event.

MP Multiprocessing facility.

P Privileged-operation exception.

PK PSW-key-handling facility.

Q Privileged-operation exception for semiprivileged instructions.

R PER general-register-alteration event.

RE Recovery-extension facility.

RR RR instruction format.

RRE RRE instruction format.

RS RS instruction format.

RX RX instruction format.

S S instruction format.

SD PER storage-alteration event, which can be caused by READ DIRECT only when INVALIDATE PAGE TABLE ENTRY is not installed.

SI SI instruction format.

SO Special-operation exception.

SP Specification exception.

SR Suspend-and-resume facility.

SS SS instruction format.

SSE SSE instruction format.

ST PER storage-alteration event.

SW Conditional-swapping facility.

T Trace exceptions (which include access and specification).

TB Test-block facility.

TR Translation facility.

XP Extended-precision floating-point facility.

Z^1 Additional exceptions and events for PROGRAM CALL (which include addressing, EX translation, LX translation, PC-translation specification, and special-operation exceptions and space-switch event).

Z^2 Additional exceptions and events for PROGRAM TRANSFER (which include addressing, primary authority, and special-operation exceptions and space-switch event).

Z^3 Additional exceptions for SET SECONDARY ASN (which include addressing, secondary authority, and special operation).

Name	Mnemonic	Characteristics													Op Code	Page No.
ADD	AR	RR	C						IF				R		1A	7-7
ADD	A	RX	C			A			IF				R		5A	7-7
ADD DECIMAL	AP	SS	C			A		D	DF					ST	FA	8-5
ADD HALFWORD	AH	RX	C			A			IF				R		4A	7-7
ADD LOGICAL	ALR	RR	C										R		1E	7-8
ADD LOGICAL	AL	RX	C			A							R		5E	7-8
ADD NORMALIZED (extended)	AXR	RR	C	XP			SP	EU	EO		LS				36	9-6
ADD NORMALIZED (long)	ADR	RR	C	FP			SP	EU	EO		LS				2A	9-6
ADD NORMALIZED (long)	AD	RX	C	FP		A	SP	EU	EO		LS				6A	9-6
ADD NORMALIZED (short)	AER	RR	C	FP			SP	EU	EO		LS				3A	9-6
ADD NORMALIZED (short)	AE	RX	C	FP		A	SP	EU	EO		LS				7A	9-6
ADD UNNORMALIZED (long)	AWR	RR	C	FP			SP		EO		LS				2E	9-7
ADD UNNORMALIZED (long)	AW	RX	C	FP		A	SP		EO		LS				6E	9-7
ADD UNNORMALIZED (short)	AUR	RR	C	FP			SP		EO		LS				3E	9-7
ADD UNNORMALIZED (short)	AU	RX	C	FP		A	SP		EO		LS				7E	9-7
AND	NR	RR	C										R		14	7-8
AND	N	RX	C			A							R		54	7-8
AND (character)	NC	SS	C			A								ST	D4	7-8
AND (immediate)	NI	SI	C			A								ST	94	7-8
BRANCH AND LINK	BALR	RR										B	R		05	7-9
BRANCH AND LINK	BAL	RX										B	R		45	7-9
BRANCH AND SAVE	BASR	RR		BS								B	R		0D	7-9
BRANCH AND SAVE	BAS	RX		BS								B	R		4D	7-9
BRANCH ON CONDITION	BCR	RR								¢[1]		B			07	7-10
BRANCH ON CONDITION	BC	RX										B			47	7-10
BRANCH ON COUNT	BCTR	RR										B	R		06	7-11
BRANCH ON COUNT	BCT	RX										B	R		46	7-11
BRANCH ON INDEX HIGH	BXH	RS										B	R		86	7-11
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS										B	R		87	7-11
CLEAR CHANNEL	CLRCH	S	C	RE	P					¢					9F01*	13-16
CLEAR I/O	CLRIO	S	C		P					¢					9D01*	13-17
COMPARE	CR	RR	C												19	7-12
COMPARE	C	RX	C			A									59	7-12
COMPARE (long)	CDR	RR	C	FP			SP								29	9-8
COMPARE (long)	CD	RX	C	FP		A	SP								69	9-8
COMPARE (short)	CER	RR	C	FP			SP								39	9-8
COMPARE (short)	CE	RX	C	FP		A	SP								79	9-8
COMPARE AND SWAP	CS	RS	C	SW		A	SP			$			R	ST	BA	7-12
COMPARE DECIMAL	CP	SS	C			A		D							F9	8-5
COMPARE DOUBLE AND SWAP	CDS	RS	C	SW		A	SP			$			R	ST	BB	7-12
COMPARE HALFWORD	CH	RX	C			A									49	7-14
COMPARE LOGICAL	CLR	RR	C												15	7-14
COMPARE LOGICAL	CL	RX	C			A									55	7-14
COMPARE LOGICAL (character)	CLC	SS	C			A									D5	7-14
COMPARE LOGICAL (immediate)	CLI	SI	C			A									95	7-14
COMPARE LOGICAL C. UNDER MASK	CLM	RS	C			A									BD	7-15
COMPARE LOGICAL LONG	CLCL	RR	C			A	SP	II					R		0F	7-15
CONNECT CHANNEL SET	CONCS	S	C	CS	P					$					B200	10-4
CONVERT TO BINARY	CVB	RX				A		D		IK			R		4F	7-16
CONVERT TO DECIMAL	CVD	RX				A								ST	4E	7-17
DIAGNOSE			DM		P	DM									83	10-5
DISCONNECT CHANNEL SET	DISCS	S	C	CS	P					$					B201	10-6
DIVIDE	DR	RR					SP			IK			R		1D	7-17
DIVIDE	D	RX				A	SP			IK			R		5D	7-17
DIVIDE (long)	DDR	RR		FP			SP	EU	EO	FK					2D	9-9

Instructions Arranged by Name (Part 1 of 4)

Name	Mne-monic	Characteristics				Op Code	Page No.
DIVIDE (long)	DD	RX FP	A SP	EU EO FK		6D	9-9
DIVIDE (short)	DER	RR FP	SP	EU EO FK		3D	9-9
DIVIDE (short)	DE	RX FP	A SP	EU EO FK		7D	9-9
DIVIDE DECIMAL	DP	SS	A SP	D DK	ST	FD	8-5
EDIT	ED	SS C	A	D	ST	DE	8-6
EDIT AND MARK	EDMK	SS C	A	D G1	R ST	DF	8-9
EXCLUSIVE OR	XR	RR C			R	17	7-18
EXCLUSIVE OR	X	RX C	A		R	57	7-18
EXCLUSIVE OR (character)	XC	SS C	A		ST	D7	7-18
EXCLUSIVE OR (immediate)	XI	SI C	A		ST	97	7-18
EXECUTE	EX	RX	AI SP	EX		44	7-19
EXTRACT PRIMARY ASN	EPAR	RRE DU	Q	SO	R	B226	10-6
EXTRACT SECONDARY ASN	ESAR	RRE DU	Q	SO	R	B227	10-7
HALT DEVICE	HDV	S C	P	¢		9E01*	13-19
HALT I/O	HIO	S C	P	¢		9E00*	13-23
HALVE (long)	HDR	RR FP	SP	EU		24	9-10
HALVE (short)	HER	RR FP	SP	EU		34	9-10
INSERT ADDRESS SPACE CONTROL	IAC	RRE C DU	Q	SO	R	B224	10-7
INSERT CHARACTER	IC	RX	A		R	43	7-20
INSERT CHARACTERS UNDER MASK	ICM	RS C	A		R	BF	7-20
INSERT PSW KEY	IPK	S PK	Q	G2	R	B20B	10-8
INSERT STORAGE KEY	ISK	RR	P A^1 SP	SO	R	09	10-8
INSERT STORAGE KEY EXTENDED	ISKE	RRE EK	P A^1		R	B229	10-9
INSERT VIRTUAL STORAGE KEY	IVSK	RRE DU	Q A^1	SO	R	B223	10-10
INVALIDATE PAGE TABLE ENTRY	IPTE	RRE EF	P A^1	$		B221	10-11
LOAD	LR	RR			R	18	7-20
LOAD	L	RX	A		R	58	7-20
LOAD (long)	LDR	RR FP	SP			28	9-10
LOAD (long)	LD	RX FP	A SP			68	9-10
LOAD (short)	LER	RR FP	SP			38	9-10
LOAD (short)	LE	RX FP	A SP			78	9-10
LOAD ADDRESS	LA	RX			R	41	7-21
LOAD ADDRESS SPACE PARAMETERS	LASP	SSE C DU	P AS SP	SO		E500	10-12
LOAD AND TEST	LTR	RR C			R	12	7-21
LOAD AND TEST (long)	LTDR	RR C FP	SP			22	9-11
LOAD AND TEST (short)	LTER	RR C FP	SP			32	9-11
LOAD COMPLEMENT	LCR	RR C		IF	R	13	7-21
LOAD COMPLEMENT (long)	LCDR	RR C FP	SP			23	9-11
LOAD COMPLEMENT (short)	LCER	RR C FP	SP			33	9-11
LOAD CONTROL	LCTL	RS	P A SP			B7	10-20
LOAD HALFWORD	LH	RX	A		R	48	7-22
LOAD MULTIPLE	LM	RS	A		R	98	7-22
LOAD NEGATIVE	LNR	RR C			R	11	7-22
LOAD NEGATIVE (long)	LNDR	RR C FP	SP			21	9-11
LOAD NEGATIVE (short)	LNER	RR C FP	SP			31	9-11
LOAD POSITIVE	LPR	RR C		IF	R	10	7-22
LOAD POSITIVE (long)	LPDR	RR C FP	SP			20	9-12
LOAD POSITIVE (short)	LPER	RR C FP	SP			30	9-12
LOAD PSW	LPSW	S L	P A SP	¢		82	10-20
LOAD REAL ADDRESS	LRA	RX C TR	P A^1		R	B1	10-21
LOAD ROUNDED (ext. to long)	LRDR	RR XP	SP	EO		25	9-12
LOAD ROUNDED (long to short)	LRER	RR XP	SP	EO		35	9-12
MONITOR CALL	MC	SI	SP	MO		AF	7-23
MOVE (character)	MVC	SS	A		ST	D2	7-23
MOVE (immediate)	MVI	SI	A		ST	92	7-23

Instructions Arranged by Name (Part 2 of 4)

Name	Mnemonic	Characteristics				Op Code	Page No.
MOVE INVERSE	MVCIN	SS MI	A		ST	E8	7-24
MOVE LONG	MVCL	RR C	A SP	II	R ST	0E	7-24
MOVE NUMERICS	MVN	SS	A		ST	D1	7-27
MOVE TO PRIMARY	MVCP	SS C DU	Q A	SO ¢	ST	DA	10-22
MOVE TO SECONDARY	MVCS	SS C DU	Q A	SO ¢	ST	DB	10-22
MOVE WITH KEY	MVCK	SS C DU	Q A		ST	D9	10-24
MOVE WITH OFFSET	MVO	SS	A		ST	F1	7-27
MOVE ZONES	MVZ	SS	A		ST	D3	7-28
MULTIPLY	MR	RR	SP		R	1C	7-28
MULTIPLY	M	RX	A SP		R	5C	7-28
MULTIPLY (extended)	MXR	RR XP	SP	EU EO		26	9-13
MULTIPLY (long to extended)	MXDR	RR XP	SP	EU EO		27	9-13
MULTIPLY (long to extended)	MXD	RX XP	A SP	EU EO		67	9-13
MULTIPLY (long)	MDR	RR FP	SP	EU EO		2C	9-13
MULTIPLY (long)	MD	RX FP	A SP	EU EO		6C	9-13
MULTIPLY (short to long)	MER	RR FP	SP	EU EO		3C	9-13
MULTIPLY (short to long)	ME	RX FP	A SP	EU EO		7C	9-13
MULTIPLY DECIMAL	MP	SS	A SP	D	ST	FC	8-10
MULTIPLY HALFWORD	MH	RX	A		R	4C	7-29
OR	OR	RR C			R	16	7-29
OR	O	RX C	A		R	56	7-29
OR (character)	OC	SS C	A		ST	D6	7-29
OR (immediate)	OI	SI C	A		ST	96	7-29
PACK	PACK	SS	A		ST	F2	7-30
PROGRAM CALL	PC	S DU	Q AT	Z^1 T ¢ GM	B R ST	B218	10-25
PROGRAM TRANSFER	PT	RRE DU	Q AT SP	Z^2 T ¢	B ST	B228	10-31
PURGE TLB	PTLB	S TR	P	$		B20D	10-36
READ DIRECT	RDD	SI DC	P A^1	$	SD	85	10-36
RESET REFERENCE BIT	RRB	S C TR	P A^1	SO		B213	10-36
RESET REFERENCE BIT EXTENDED	RRBE	RRE C EK	P A^1			B22A	10-37
RESUME I/O	RIO	S C SR	P	¢		9C02*	13-26
SET ADDRESS SPACE CONTROL	SAC	S DU	SP	SO ¢		B219	10-38
SET CLOCK	SCK	S C	P A SP			B204	10-39
SET CLOCK COMPARATOR	SCKC	S CK	P A SP			B206	10-39
SET CPU TIMER	SPT	S CK	P A SP			B208	10-40
SET PREFIX	SPX	S MP	P A SP	$		B210	10-40
SET PROGRAM MASK	SPM	RR L				04	7-31
SET PSW KEY FROM ADDRESS	SPKA	S PK	Q			B20A	10-41
SET SECONDARY ASN	SSAR	RRE DU	AT	Z^3 T ¢	ST	B225	10-41
SET STORAGE KEY	SSK	RR	P A^1 SP	SO ¢		08	10-45
SET STORAGE KEY EXTENDED	SSKE	RRE EK	P A^1	¢		B22B	10-45
SET SYSTEM MASK	SSM	S	P A SP	SO		80	10-46
SHIFT AND ROUND DECIMAL	SRP	SS C	A	D DF	ST	F0	8-10
SHIFT LEFT DOUBLE	SLDA	RS C	SP	IF	R	8F	7-31
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS	SP		R	8D	7-32
SHIFT LEFT SINGLE	SLA	RS C		IF	R	8B	7-32
SHIFT LEFT SINGLE LOGICAL	SLL	RS			R	89	7-33
SHIFT RIGHT DOUBLE	SRDA	RS C	SP		R	8E	7-33
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS	SP		R	8C	7-33
SHIFT RIGHT SINGLE	SRA	RS C			R	8A	7-34
SHIFT RIGHT SINGLE LOGICAL	SRL	RS			R	88	7-34
SIGNAL PROCESSOR	SIGP	RS C MP	P	$	R	AE	10-46
START I/O	SIO	S C	P	¢		9C00*	13-27
START I/O FAST RELEASE	SIOF	S C	P	¢		9C01*	13-27
STORE	ST	RX	A		ST	50	7-34

Instructions Arranged by Name (Part 3 of 4)

Name	Mnemonic	Characteristics				Op Code	Page No.
STORE (long)	STD	RX FP	A SP		ST	60	9-14
STORE (short)	STE	RX FP	A SP		ST	70	9-14
STORE CHANNEL ID	STIDC	S C	P	¢		B203	13-32
STORE CHARACTER	STC	RX	A		ST	42	7-34
STORE CHARACTERS UNDER MASK	STCM	RS	A		ST	BE	7-35
STORE CLOCK	STCK	S C	A	$	ST	B205	7-35
STORE CLOCK COMPARATOR	STCKC	S CK	P A SP		ST	B207	10-47
STORE CONTROL	STCTL	RS	P A SP		ST	B6	10-48
STORE CPU ADDRESS	STAP	S MP	P A SP		ST	B212	10-48
STORE CPU ID	STIDP	S	P A SP		ST	B202	10-48
STORE CPU TIMER	STPT	S CK	P A SP		ST	B209	10-49
STORE HALFWORD	STH	RX	A		ST	40	7-36
STORE MULTIPLE	STM	RS	A		ST	90	7-36
STORE PREFIX	STPX	S MP	P A SP		ST	B211	10-49
STORE THEN AND SYSTEM MASK	STNSM	SI TR	P A		ST	AC	10-50
STORE THEN OR SYSTEM MASK	STOSM	SI TR	P A SP		ST	AD	10-50
SUBTRACT	SR	RR C		IF	R	1B	7-36
SUBTRACT	S	RX C	A	IF	R	5B	7-36
SUBTRACT DECIMAL	SP	SS C	A	D DF	ST	FB	8-11
SUBTRACT HALFWORD	SH	RX C	A	IF	R	4B	7-37
SUBTRACT LOGICAL	SLR	RR C			R	1F	7-37
SUBTRACT LOGICAL	SL	RX C	A		R	5F	7-37
SUBTRACT NORMALIZED (ext.)	SXR	RR C XP	SP	EU EO LS		37	9-14
SUBTRACT NORMALIZED (long)	SDR	RR C FP	SP	EU EO LS		2B	9-14
SUBTRACT NORMALIZED (long)	SD	RX C FP	A SP	EU EO LS		6B	9-14
SUBTRACT NORMALIZED (short)	SER	RR C FP	SP	EU EO LS		3B	9-14
SUBTRACT NORMALIZED (short)	SE	RX C FP	A SP	EU EO LS		7B	9-14
SUBTRACT UNNORMALIZED (long)	SWR	RR C FP	SP	EO LS		2F	9-15
SUBTRACT UNNORMALIZED (long)	SW	RX C FP	A SP	EO LS		6F	9-15
SUBTRACT UNNORMALIZED (short)	SUR	RR C FP	SP	EO LS		3F	9-15
SUBTRACT UNNORMALIZED (short)	SU	RX C FP	A SP	EO LS		7F	9-15
SUPERVISOR CALL	SVC	RR		¢		0A	7-38
TEST AND SET	TS	S C	A	$	ST	93	7-38
TEST BLOCK	TB	RRE C TB	P A[1]	II $ GO	R	B22C	10-50
TEST CHANNEL	TCH	S C	P	¢		9F00*	13-33
TEST I/O	TIO	S C	P	¢		9D00*	13-34
TEST PROTECTION	TPROT	SSE C EF	P A[1]			E501	10-52
TEST UNDER MASK	TM	SI C	A			91	7-38
TRANSLATE	TR	SS	A		ST	DC	7-39
TRANSLATE AND TEST	TRT	SS C	A	GM	R	DD	7-40
UNPACK	UNPK	SS	A		ST	F3	7-40
WRITE DIRECT	WRD	SI DC	P A[1]	$		84	10-54
ZERO AND ADD	ZAP	SS C	A	D DF	ST	F8	8-12

Instructions Arranged by Name (Part 4 of 4)

Mne- monic	Name	Characteristics													Op Code	Page No.
	DIAGNOSE		DM		P	DM									83	10-5
A	ADD	RX	C			A			IF				R		5A	7-7
AD	ADD NORMALIZED (long)	RX	C	FP		A	SP	EU	EO		LS				6A	9-6
ADR	ADD NORMALIZED (long)	RR	C	FP			SP	EU	EO		LS				2A	9-6
AE	ADD NORMALIZED (short)	RX	C	FP		A	SP	EU	EO		LS				7A	9-6
AER	ADD NORMALIZED (short)	RR	C	FP			SP	EU	EO		LS				3A	9-6
AH	ADD HALFWORD	RX	C			A			IF				R		4A	7-7
AL	ADD LOGICAL	RX	C			A							R		5E	7-8
ALR	ADD LOGICAL	RR	C										R		1E	7-8
AP	ADD DECIMAL	SS	C			A		D	DF					ST	FA	8-5
AR	ADD	RR	C						IF				R		1A	7-7
AU	ADD UNNORMALIZED (short)	RX	C	FP		A	SP		EO		LS				7E	9-7
AUR	ADD UNNORMALIZED (short)	RR	C	FP			SP		EO		LS				3E	9-7
AW	ADD UNNORMALIZED (long)	RX	C	FP		A	SP		EO		LS				6E	9-7
AWR	ADD UNNORMALIZED (long)	RR	C	FP			SP		EO		LS				2E	9-7
AXR	ADD NORMALIZED (extended)	RR	C	XP			SP	EU	EO		LS				36	9-6
BAL	BRANCH AND LINK	RX										B	R		45	7-9
BALR	BRANCH AND LINK	RR										B	R		05	7-9
BAS	BRANCH AND SAVE	RX		BS								B	R		4D	7-9
BASR	BRANCH AND SAVE	RR		BS								B	R		0D	7-9
BC	BRANCH ON CONDITION	RX										B			47	7-10
BCR	BRANCH ON CONDITION	RR								¢[1]		B			07	7-10
BCT	BRANCH ON COUNT	RX										B	R		46	7-11
BCTR	BRANCH ON COUNT	RR										B	R		06	7-11
BXH	BRANCH ON INDEX HIGH	RS										B	R		86	7-11
BXLE	BRANCH ON INDEX LOW OR EQUAL	RS										B	R		87	7-11
C	COMPARE	RX	C			A									59	7-12
CD	COMPARE (long)	RX	C	FP		A	SP								69	9-8
CDR	COMPARE (long)	RR	C	FP			SP								29	9-8
CDS	COMPARE DOUBLE AND SWAP	RS	C	SW		A	SP			$			R	ST	BB	7-12
CE	COMPARE (short)	RX	C	FP		A	SP								79	9-8
CER	COMPARE (short)	RR	C	FP			SP								39	9-8
CH	COMPARE HALFWORD	RX	C			A									49	7-14
CL	COMPARE LOGICAL	RX	C			A									55	7-14
CLC	COMPARE LOGICAL (character)	SS	C			A									D5	7-14
CLCL	COMPARE LOGICAL LONG	RR	C			A	SP	II					R		0F	7-15
CLI	COMPARE LOGICAL (immediate)	SI	C			A									95	7-14
CLM	COMPARE LOGICAL C. UNDER MASK	RS	C			A									BD	7-15
CLR	COMPARE LOGICAL	RR	C												15	7-14
CLRCH	CLEAR CHANNEL	S	C	RE	P					¢					9F01*	13-16
CLRIO	CLEAR I/O	S	C		P					¢					9D01*	13-17
CONCS	CONNECT CHANNEL SET	S	C	CS	P					$					B200	10-4
CP	COMPARE DECIMAL	SS	C			A		D							F9	8-5
CR	COMPARE	RR	C												19	7-12
CS	COMPARE AND SWAP	RS	C	SW		A	SP			$			R	ST	BA	7-12
CVB	CONVERT TO BINARY	RX				A		D		IK			R		4F	7-16
CVD	CONVERT TO DECIMAL	RX				A								ST	4E	7-17
D	DIVIDE	RX				A	SP			IK			R		5D	7-17
DD	DIVIDE (long)	RX		FP		A	SP	EU	EO	FK					6D	9-9
DDR	DIVIDE (long)	RR		FP			SP	EU	EO	FK					2D	9-9
DE	DIVIDE (short)	RX		FP		A	SP	EU	EO	FK					7D	9-9
DER	DIVIDE (short)	RR		FP			SP	EU	EO	FK					3D	9-9
DISCS	DISCONNECT CHANNEL SET	S	C	CS	P					$					B201	10-6
DP	DIVIDE DECIMAL	SS				A	SP	D		DK				ST	FD	8-5
DR	DIVIDE	RR					SP			IK			R		1D	7-17

Instructions Arranged by Mnemonic (Part 1 of 4)

Mne-monic	Name	Characteristics				Op Code	Page No.
ED	EDIT	SS C	A	D	ST	DE	8-6
EDMK	EDIT AND MARK	SS C	A	D G1	R ST	DF	8-9
EPAR	EXTRACT PRIMARY ASN	RRE DU	Q	SO	R	B226	10-6
ESAR	EXTRACT SECONDARY ASN	RRE DU	Q	SO	R	B227	10-7
EX	EXECUTE	RX	AI SP	EX		44	7-19
HDR	HALVE (long)	RR FP	SP	EU		24	9-10
HDV	HALT DEVICE	S C	P	¢		9E01*	13-19
HER	HALVE (short)	RR FP	SP	EU		34	9-10
HIO	HALT I/O	S C	P	¢		9E00*	13-23
IAC	INSERT ADDRESS SPACE CONTROL	RRE C DU	Q	SO	R	B224	10-7
IC	INSERT CHARACTER	RX	A		R	43	7-20
ICM	INSERT CHARACTERS UNDER MASK	RS C	A		R	BF	7-20
IPK	INSERT PSW KEY	S PK	Q	G2	R	B20B	10-8
IPTE	INVALIDATE PAGE TABLE ENTRY	RRE EF	P A^1	$		B221	10-11
ISK	INSERT STORAGE KEY	RR	P A^1 SP	SO	R	09	10-8
ISKE	INSERT STORAGE KEY EXTENDED	RRE EK	P A^1		R	B229	10-9
IVSK	INSERT VIRTUAL STORAGE KEY	RRE DU	Q A^1	SO	R	B223	10-10
L	LOAD	RX	A		R	58	7-20
LA	LOAD ADDRESS	RX			R	41	7-21
LASP	LOAD ADDRESS SPACE PARAMETERS	SSE C DU	P AS SP	SO		E500	10-12
LCDR	LOAD COMPLEMENT (long)	RR C FP	SP			23	9-11
LCER	LOAD COMPLEMENT (short)	RR C FP	SP			33	9-11
LCR	LOAD COMPLEMENT	RR C		IF	R	13	7-21
LCTL	LOAD CONTROL	RS	P A SP			B7	10-20
LD	LOAD (long)	RX FP	A SP			68	9-10
LDR	LOAD (long)	RR FP	SP			28	9-10
LE	LOAD (short)	RX FP	A SP			78	9-10
LER	LOAD (short)	RR FP	SP			38	9-10
LH	LOAD HALFWORD	RX	A		R	48	7-22
LM	LOAD MULTIPLE	RS	A		R	98	7-22
LNDR	LOAD NEGATIVE (long)	RR C FP	SP			21	9-11
LNER	LOAD NEGATIVE (short)	RR C FP	SP			31	9-11
LNR	LOAD NEGATIVE	RR C			R	11	7-22
LPDR	LOAD POSITIVE (long)	RR C FP	SP			20	9-12
LPER	LOAD POSITIVE (short)	RR C FP	SP			30	9-12
LPR	LOAD POSITIVE	RR C		IF	R	10	7-22
LPSW	LOAD PSW	S L	P A SP	¢		82	10-20
LR	LOAD	RR			R	18	7-20
LRA	LOAD REAL ADDRESS	RX C TR	P A^1		R	B1	10-21
LRDR	LOAD ROUNDED (ext. to long)	RR XP	SP	EO		25	9-12
LRER	LOAD ROUNDED (long to short)	RR XP	SP	EO		35	9-12
LTDR	LOAD AND TEST (long)	RR C FP	SP			22	9-11
LTER	LOAD AND TEST (short)	RR C FP	SP			32	9-11
LTR	LOAD AND TEST	RR C			R	12	7-21
M	MULTIPLY	RX	A SP		R	5C	7-28
MC	MONITOR CALL	SI	SP	MO		AF	7-23
MD	MULTIPLY (long)	RX FP	A SP	EU EO		6C	9-13
MDR	MULTIPLY (long)	RR FP	SP	EU EO		2C	9-13
ME	MULTIPLY (short to long)	RX FP	A SP	EU EO		7C	9-13
MER	MULTIPLY (short to long)	RR FP	SP	EU EO		3C	9-13
MH	MULTIPLY HALFWORD	RX	A		R	4C	7-29
MP	MULTIPLY DECIMAL	SS	A SP	D	ST	FC	8-10
MR	MULTIPLY	RR	SP		R	1C	7-28
MVC	MOVE (character)	SS	A		ST	D2	7-23
MVCIN	MOVE INVERSE	SS MI	A		ST	E8	7-24

Instructions Arranged by Mnemonic (Part 2 of 4)

Mne-monic	Name	Characteristics													Op Code	Page No.
MVCK	MOVE WITH KEY	SS	C	DU	Q	A								ST	D9	10-24
MVCL	MOVE LONG	RR	C			A	SP	II					R	ST	0E	7-24
MVCP	MOVE TO PRIMARY	SS	C	DU	Q	A		SO		¢				ST	DA	10-22
MVCS	MOVE TO SECONDARY	SS	C	DU	Q	A		SO		¢				ST	DB	10-22
MVI	MOVE (immediate)	SI				A								ST	92	7-23
MVN	MOVE NUMERICS	SS				A								ST	D1	7-27
MVO	MOVE WITH OFFSET	SS				A								ST	F1	7-27
MVZ	MOVE ZONES	SS				A								ST	D3	7-28
MXD	MULTIPLY (long to extended)	RX		XP		A	SP	EU	EO						67	9-13
MXDR	MULTIPLY (long to extended)	RR		XP			SP	EU	EO						27	9-13
MXR	MULTIPLY (extended)	RR		XP			SP	EU	EO						26	9-13
N	AND	RX	C			A							R		54	7-8
NC	AND (character)	SS	C			A								ST	D4	7-8
NI	AND (immediate)	SI	C			A								ST	94	7-8
NR	AND	RR	C										R		14	7-8
O	OR	RX	C			A							R		56	7-29
OC	OR (character)	SS	C			A								ST	D6	7-29
OI	OR (immediate)	SI	C			A								ST	96	7-29
OR	OR	RR	C										R		16	7-29
PACK	PACK	SS				A								ST	F2	7-30
PC	PROGRAM CALL	S		DU	Q	AT		Z^1	T	¢	GM	B	R	ST	B218	10-25
PT	PROGRAM TRANSFER	RRE		DU	Q	AT	SP	Z^2	T	¢		B		ST	B228	10-31
PTLB	PURGE TLB	S		TR	P					$					B20D	10-36
RDD	READ DIRECT	SI		DC	P	A^1				$				SD	85	10-36
RIO	RESUME I/O	S	C	SR	P					¢					9C02*	13-26
RRB	RESET REFERENCE BIT	S	C	TR	P	A^1		SO							B213	10-36
RRBE	RESET REFERENCE BIT EXTENDED	RRE	C	EK	P	A^1									B22A	10-37
S	SUBTRACT	RX	C			A			IF				R		5B	7-36
SAC	SET ADDRESS SPACE CONTROL	S		DU			SP	SO		¢					B219	10-38
SCK	SET CLOCK	S	C		P	A	SP								B204	10-39
SCKC	SET CLOCK COMPARATOR	S		CK	P	A	SP								B206	10-39
SD	SUBTRACT NORMALIZED (long)	RX	C	FP		A	SP	EU	EO		LS				6B	9-14
SDR	SUBTRACT NORMALIZED (long)	RR	C	FP			SP	EU	EO		LS				2B	9-14
SE	SUBTRACT NORMALIZED (short)	RX	C	FP		A	SP	EU	EO		LS				7B	9-14
SER	SUBTRACT NORMALIZED (short)	RR	C	FP			SP	EU	EO		LS				3B	9-14
SH	SUBTRACT HALFWORD	RX	C			A			IF				R		4B	7-37
SIGP	SIGNAL PROCESSOR	RS	C	MP	P					$			R		AE	10-46
SIO	START I/O	S	C		P					¢					9C00*	13-27
SIOF	START I/O FAST RELEASE	S	C		P					¢					9C01*	13-27
SL	SUBTRACT LOGICAL	RX	C			A							R		5F	7-37
SLA	SHIFT LEFT SINGLE	RS	C						IF				R		8B	7-32
SLDA	SHIFT LEFT DOUBLE	RS	C				SP		IF				R		8F	7-31
SLDL	SHIFT LEFT DOUBLE LOGICAL	RS					SP						R		8D	7-32
SLL	SHIFT LEFT SINGLE LOGICAL	RS											R		89	7-33
SLR	SUBTRACT LOGICAL	RR	C										R		1F	7-37
SP	SUBTRACT DECIMAL	SS	C			A		D	DF					ST	FB	8-11
SPKA	SET PSW KEY FROM ADDRESS	S		PK	Q										B20A	10-41
SPM	SET PROGRAM MASK	RR	L												04	7-31
SPT	SET CPU TIMER	S		CK	P	A	SP								B208	10-40
SPX	SET PREFIX	S		MP	P	A	SP			$					B210	10-40
SR	SUBTRACT	RR	C						IF				R		1B	7-36
SRA	SHIFT RIGHT SINGLE	RS	C										R		8A	7-34
SRDA	SHIFT RIGHT DOUBLE	RS	C				SP						R		8E	7-33
SRDL	SHIFT RIGHT DOUBLE LOGICAL	RS					SP						R		8C	7-33
SRL	SHIFT RIGHT SINGLE LOGICAL	RS											R		88	7-34

Instructions Arranged by Mnemonic (Part 3 of 4)

Mne-monic	Name	Characteristics												Op Code	Page No.
SRP	SHIFT AND ROUND DECIMAL	SS	C			A		D	DF				ST	F0	8-10
SSAR	SET SECONDARY ASN	RRE		DU		AT		Z^3	T	¢			ST	B225	10-41
SSK	SET STORAGE KEY	RR			P	A^1	SP	SO		¢				08	10-45
SSKE	SET STORAGE KEY EXTENDED	RRE		EK	P	A^1				¢				B22B	10-45
SSM	SET SYSTEM MASK	S			P	A	SP	SO						80	10-46
ST	STORE	RX				A							ST	50	7-34
STAP	STORE CPU ADDRESS	S		MP	P	A	SP						ST	B212	10-48
STC	STORE CHARACTER	RX				A							ST	42	7-34
STCK	STORE CLOCK	S	C			A				$			ST	B205	7-35
STCKC	STORE CLOCK COMPARATOR	S		CK	P	A	SP						ST	B207	10-47
STCM	STORE CHARACTERS UNDER MASK	RS				A							ST	BE	7-35
STCTL	STORE CONTROL	RS			P	A	SP						ST	B6	10-48
STD	STORE (long)	RX		FP		A	SP						ST	60	9-14
STE	STORE (short)	RX		FP		A	SP						ST	70	9-14
STH	STORE HALFWORD	RX				A							ST	40	7-36
STIDC	STORE CHANNEL ID	S	C		P					¢				B203	13-32
STIDP	STORE CPU ID	S			P	A	SP						ST	B202	10-48
STM	STORE MULTIPLE	RS				A							ST	90	7-36
STNSM	STORE THEN AND SYSTEM MASK	SI		TR	P	A							ST	AC	10-50
STOSM	STORE THEN OR SYSTEM MASK	SI		TR	P	A	SP						ST	AD	10-50
STPT	STORE CPU TIMER	S		CK	P	A	SP						ST	B209	10-49
STPX	STORE PREFIX	S		MP	P	A	SP						ST	B211	10-49
SU	SUBTRACT UNNORMALIZED (short)	RX	C	FP		A	SP		EO		LS			7F	9-15
SUR	SUBTRACT UNNORMALIZED (short)	RR	C	FP			SP		EO		LS			3F	9-15
SVC	SUPERVISOR CALL	RR								¢				0A	7-38
SW	SUBTRACT UNNORMALIZED (long)	RX	C	FP		A	SP		EO		LS			6F	9-15
SWR	SUBTRACT UNNORMALIZED (long)	RR	C	FP			SP		EO		LS			2F	9-15
SXR	SUBTRACT NORMALIZED (ext.)	RR	C	XP			SP	EU	EO		LS			37	9-14
TB	TEST BLOCK	RRE	C	TB	P	A^1		II		$	GO	R		B22C	10-50
TCH	TEST CHANNEL	S	C		P					¢				9F00*	13-33
TIO	TEST I/O	S	C		P					¢				9D00*	13-34
TM	TEST UNDER MASK	SI	C			A								91	7-38
TPROT	TEST PROTECTION	SSE	C	EF	P	A^1								E501	10-52
TR	TRANSLATE	SS				A							ST	DC	7-39
TRT	TRANSLATE AND TEST	SS	C			A					GM	R		DD	7-40
TS	TEST AND SET	S	C			A				$			ST	93	7-38
UNPK	UNPACK	SS				A							ST	F3	7-40
WRD	WRITE DIRECT	SI		DC	P	A^1				$				84	10-54
X	EXCLUSIVE OR	RX	C			A						R		57	7-18
XC	EXCLUSIVE OR (character)	SS	C			A							ST	D7	7-18
XI	EXCLUSIVE OR (immediate)	SI	C			A							ST	97	7-18
XR	EXCLUSIVE OR	RR	C									R		17	7-18
ZAP	ZERO AND ADD	SS	C			A		D	DF				ST	F8	8-12

Instructions Arranged by Mnemonic (Part 4 of 4)

Op Code	Name	Mne- monic	Characteristics				Page No.
04	SET PROGRAM MASK	SPM	RR L				7-31
05	BRANCH AND LINK	BALR	RR			B R	7-9
06	BRANCH ON COUNT	BCTR	RR			B R	7-11
07	BRANCH ON CONDITION	BCR	RR		¢[1]	B	7-10
08	SET STORAGE KEY	SSK	RR	P A[1] SP	SO ¢		10-45
09	INSERT STORAGE KEY	ISK	RR	P A[1] SP	SO	R	10-8
0A	SUPERVISOR CALL	SVC	RR		¢		7-38
0D	BRANCH AND SAVE	BASR	RR BS			B R	7-9
0E	MOVE LONG	MVCL	RR C	A SP	II	R ST	7-24
0F	COMPARE LOGICAL LONG	CLCL	RR C	A SP	II	R	7-15
10	LOAD POSITIVE	LPR	RR C		IF	R	7-22
11	LOAD NEGATIVE	LNR	RR C			R	7-22
12	LOAD AND TEST	LTR	RR C			R	7-21
13	LOAD COMPLEMENT	LCR	RR C		IF	R	7-21
14	AND	NR	RR C			R	7-8
15	COMPARE LOGICAL	CLR	RR C				7-14
16	OR	OR	RR C			R	7-29
17	EXCLUSIVE OR	XR	RR C			R	7-18
18	LOAD	LR	RR			R	7-20
19	COMPARE	CR	RR C				7-12
1A	ADD	AR	RR C		IF	R	7-7
1B	SUBTRACT	SR	RR C		IF	R	7-36
1C	MULTIPLY	MR	RR	SP		R	7-28
1D	DIVIDE	DR	RR	SP	IK	R	7-17
1E	ADD LOGICAL	ALR	RR C			R	7-8
1F	SUBTRACT LOGICAL	SLR	RR C			R	7-37
20	LOAD POSITIVE (long)	LPDR	RR C FP	SP			9-12
21	LOAD NEGATIVE (long)	LNDR	RR C FP	SP			9-11
22	LOAD AND TEST (long)	LTDR	RR C FP	SP			9-11
23	LOAD COMPLEMENT (long)	LCDR	RR C FP	SP			9-11
24	HALVE (long)	HDR	RR FP	SP	EU		9-10
25	LOAD ROUNDED (ext. to long)	LRDR	RR XP	SP	EO		9-12
26	MULTIPLY (extended)	MXR	RR XP	SP	EU EO		9-13
27	MULTIPLY (long to extended)	MXDR	RR XP	SP	EU EO		9-13
28	LOAD (long)	LDR	RR FP	SP			9-10
29	COMPARE (long)	CDR	RR C FP	SP			9-8
2A	ADD NORMALIZED (long)	ADR	RR C FP	SP	EU EO LS		9-6
2B	SUBTRACT NORMALIZED (long)	SDR	RR C FP	SP	EU EO LS		9-14
2C	MULTIPLY (long)	MDR	RR FP	SP	EU EO		9-13
2D	DIVIDE (long)	DDR	RR FP	SP	EU EO FK		9-9
2E	ADD UNNORMALIZED (long)	AWR	RR C FP	SP	EO LS		9-7
2F	SUBTRACT UNNORMALIZED (long)	SWR	RR C FP	SP	EO LS		9-15
30	LOAD POSITIVE (short)	LPER	RR C FP	SP			9-12
31	LOAD NEGATIVE (short)	LNER	RR C FP	SP			9-11
32	LOAD AND TEST (short)	LTER	RR C FP	SP			9-11
33	LOAD COMPLEMENT (short)	LCER	RR C FP	SP			9-11
34	HALVE (short)	HER	RR FP	SP	EU		9-10
35	LOAD ROUNDED (long to short)	LRER	RR XP	SP	EO		9-12
36	ADD NORMALIZED (extended)	AXR	RR C XP	SP	EU EO LS		9-6
37	SUBTRACT NORMALIZED (ext.)	SXR	RR C XP	SP	EU EO LS		9-14
38	LOAD (short)	LER	RR FP	SP			9-10
39	COMPARE (short)	CER	RR C FP	SP			9-8
3A	ADD NORMALIZED (short)	AER	RR C FP	SP	EU EO LS		9-6
3B	SUBTRACT NORMALIZED (short)	SER	RR C FP	SP	EU EO LS		9-14
3C	MULTIPLY (short to long)	MER	RR FP	SP	EU EO		9-13

Instructions Arranged by Operation Code (Part 1 of 4)

Op Code	Name	Mne- monic	Characteristics													Page No.
3D	DIVIDE (short)	DER	RR		FP			SP	EU	EO	FK					9-9
3E	ADD UNNORMALIZED (short)	AUR	RR	C	FP			SP		EO		LS				9-7
3F	SUBTRACT UNNORMALIZED (short)	SUR	RR	C	FP			SP		EO		LS				9-15
40	STORE HALFWORD	STH	RX				A								ST	7-36
41	LOAD ADDRESS	LA	RX											R		7-21
42	STORE CHARACTER	STC	RX				A								ST	7-34
43	INSERT CHARACTER	IC	RX				A							R		7-20
44	EXECUTE	EX	RX				AI	SP				EX				7-19
45	BRANCH AND LINK	BAL	RX										B	R		7-9
46	BRANCH ON COUNT	BCT	RX										B	R		7-11
47	BRANCH ON CONDITION	BC	RX										B			7-10
48	LOAD HALFWORD	LH	RX				A							R		7-22
49	COMPARE HALFWORD	CH	RX	C			A									7-14
4A	ADD HALFWORD	AH	RX	C			A			IF				R		7-7
4B	SUBTRACT HALFWORD	SH	RX	C			A			IF				R		7-37
4C	MULTIPLY HALFWORD	MH	RX				A							R		7-29
4D	BRANCH AND SAVE	BAS	RX		BS								B	R		7-9
4E	CONVERT TO DECIMAL	CVD	RX				A								ST	7-17
4F	CONVERT TO BINARY	CVB	RX				A		D		IK			R		7-16
50	STORE	ST	RX				A								ST	7-34
54	AND	N	RX	C			A							R		7-8
55	COMPARE LOGICAL	CL	RX	C			A									7-14
56	OR	O	RX	C			A							R		7-29
57	EXCLUSIVE OR	X	RX	C			A							R		7-18
58	LOAD	L	RX				A							R		7-20
59	COMPARE	C	RX	C			A									7-12
5A	ADD	A	RX	C			A			IF				R		7-7
5B	SUBTRACT	S	RX	C			A			IF				R		7-36
5C	MULTIPLY	M	RX				A	SP						R		7-28
5D	DIVIDE	D	RX				A	SP			IK			R		7-17
5E	ADD LOGICAL	AL	RX	C			A							R		7-8
5F	SUBTRACT LOGICAL	SL	RX	C			A							R		7-37
60	STORE (long)	STD	RX		FP		A	SP							ST	9-14
67	MULTIPLY (long to extended)	MXD	RX		XP		A	SP	EU	EO						9-13
68	LOAD (long)	LD	RX		FP		A	SP								9-10
69	COMPARE (long)	CD	RX	C	FP		A	SP								9-8
6A	ADD NORMALIZED (long)	AD	RX	C	FP		A	SP	EU	EO		LS				9-6
6B	SUBTRACT NORMALIZED (long)	SD	RX	C	FP		A	SP	EU	EO		LS				9-14
6C	MULTIPLY (long)	MD	RX		FP		A	SP	EU	EO						9-13
6D	DIVIDE (long)	DD	RX		FP		A	SP	EU	EO	FK					9-9
6E	ADD UNNORMALIZED (long)	AW	RX	C	FP		A	SP		EO		LS				9-7
6F	SUBTRACT UNNORMALIZED (long)	SW	RX	C	FP		A	SP		EO		LS				9-15
70	STORE (short)	STE	RX		FP		A	SP							ST	9-14
78	LOAD (short)	LE	RX		FP		A	SP								9-10
79	COMPARE (short)	CE	RX	C	FP		A	SP								9-8
7A	ADD NORMALIZED (short)	AE	RX	C	FP		A	SP	EU	EO		LS				9-6
7B	SUBTRACT NORMALIZED (short)	SE	RX	C	FP		A	SP	EU	EO		LS				9-14
7C	MULTIPLY (short to long)	ME	RX		FP		A	SP	EU	EO						9-13
7D	DIVIDE (short)	DE	RX		FP		A	SP	EU	EO	FK					9-9
7E	ADD UNNORMALIZED (short)	AU	RX	C	FP		A	SP		EO		LS				9-7
7F	SUBTRACT UNNORMALIZED (short)	SU	RX	C	FP		A	SP		EO		LS				9-15
80	SET SYSTEM MASK	SSM	S			P	A	SP	SO							10-46
82	LOAD PSW	LPSW	S	L		P	A	SP			¢					10-20
83	DIAGNOSE			DM		P	DM									10-5
84	WRITE DIRECT	WRD	SI		DC	P	A[1]				$					10-54

Instructions Arranged by Operation Code (Part 2 of 4)

Op Code	Name	Mnemonic	Characteristics				Page No.
85	READ DIRECT	RDD	SI DC	P A[1]	$	SD	10-36
86	BRANCH ON INDEX HIGH	BXH	RS			B R	7-11
87	BRANCH ON INDEX LOW OR EQUAL	BXLE	RS			B R	7-11
88	SHIFT RIGHT SINGLE LOGICAL	SRL	RS			R	7-34
89	SHIFT LEFT SINGLE LOGICAL	SLL	RS			R	7-33
8A	SHIFT RIGHT SINGLE	SRA	RS C			R	7-34
8B	SHIFT LEFT SINGLE	SLA	RS C		IF	R	7-32
8C	SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS	SP		R	7-33
8D	SHIFT LEFT DOUBLE LOGICAL	SLDL	RS	SP		R	7-32
8E	SHIFT RIGHT DOUBLE	SRDA	RS C	SP		R	7-33
8F	SHIFT LEFT DOUBLE	SLDA	RS C	SP	IF	R	7-31
90	STORE MULTIPLE	STM	RS	A		ST	7-36
91	TEST UNDER MASK	TM	SI C	A			7-38
92	MOVE (immediate)	MVI	SI	A		ST	7-23
93	TEST AND SET	TS	S C	A	$	ST	7-38
94	AND (immediate)	NI	SI C	A		ST	7-8
95	COMPARE LOGICAL (immediate)	CLI	SI C	A			7-14
96	OR (immediate)	OI	SI C	A		ST	7-29
97	EXCLUSIVE OR (immediate)	XI	SI C	A		ST	7-18
98	LOAD MULTIPLE	LM	RS	A		R	7-22
9C00*	START I/O	SIO	S C	P	¢		13-27
9C01*	START I/O FAST RELEASE	SIOF	S C	P	¢		13-27
9C02*	RESUME I/O	RIO	S C SR	P	¢		13-26
9D00*	TEST I/O	TIO	S C	P	¢		13-34
9D01*	CLEAR I/O	CLRIO	S C	P	¢		13-17
9E00*	HALT I/O	HIO	S C	P	¢		13-23
9E01*	HALT DEVICE	HDV	S C	P	¢		13-19
9F00*	TEST CHANNEL	TCH	S C	P	¢		13-33
9F01*	CLEAR CHANNEL	CLRCH	S C RE	P	¢		13-16
AC	STORE THEN AND SYSTEM MASK	STNSM	SI TR	P A		ST	10-50
AD	STORE THEN OR SYSTEM MASK	STOSM	SI TR	P A SP		ST	10-50
AE	SIGNAL PROCESSOR	SIGP	RS C MP	P	$	R	10-46
AF	MONITOR CALL	MC	SI	SP	MO		7-23
B1	LOAD REAL ADDRESS	LRA	RX C TR	P A[1]		R	10-21
B200	CONNECT CHANNEL SET	CONCS	S C CS	P	$		10-4
B201	DISCONNECT CHANNEL SET	DISCS	S C CS	P	$		10-6
B202	STORE CPU ID	STIDP	S	P A SP		ST	10-48
B203	STORE CHANNEL ID	STIDC	S C	P	¢		13-32
B204	SET CLOCK	SCK	S C	P A SP			10-39
B205	STORE CLOCK	STCK	S C	A	$	ST	7-35
B206	SET CLOCK COMPARATOR	SCKC	S CK	P A SP			10-39
B207	STORE CLOCK COMPARATOR	STCKC	S CK	P A SP		ST	10-47
B208	SET CPU TIMER	SPT	S CK	P A SP			10-40
B209	STORE CPU TIMER	STPT	S CK	P A SP		ST	10-49
B20A	SET PSW KEY FROM ADDRESS	SPKA	S PK	Q			10-41
B20B	INSERT PSW KEY	IPK	S PK	Q	G2	R	10-8
B20D	PURGE TLB	PTLB	S TR	P	$		10-36
B210	SET PREFIX	SPX	S MP	P A SP	$		10-40
B211	STORE PREFIX	STPX	S MP	P A SP		ST	10-49
B212	STORE CPU ADDRESS	STAP	S MP	P A SP		ST	10-48
B213	RESET REFERENCE BIT	RRB	S C TR	P A[1]	SO		10-36
B218	PROGRAM CALL	PC	S DU	Q AT	Z[1] T ¢ GM	B R ST	10-25
B219	SET ADDRESS SPACE CONTROL	SAC	S DU	SP	SO ¢		10-38
B221	INVALIDATE PAGE TABLE ENTRY	IPTE	RRE EF	P A[1]	$		10-11
B223	INSERT VIRTUAL STORAGE KEY	IVSK	RRE DU	Q A[1]	SO	R	10-10

Instructions Arranged by Operation Code (Part 3 of 4)

Op Code	Name	Mne-monic	Characteristics													Page No.
B224	INSERT ADDRESS SPACE CONTROL	IAC	RRE	C	DU	Q			SO					R		10-7
B225	SET SECONDARY ASN	SSAR	RRE		DU		AT		Z^3	T	¢				ST	10-41
B226	EXTRACT PRIMARY ASN	EPAR	RRE		DU	Q			SO					R		10-6
B227	EXTRACT SECONDARY ASN	ESAR	RRE		DU	Q			SO					R		10-7
B228	PROGRAM TRANSFER	PT	RRE		DU	Q	AT	SP	Z^2	T	¢		B		ST	10-31
B229	INSERT STORAGE KEY EXTENDED	ISKE	RRE		EK	P	A^1							R		10-9
B22A	RESET REFERENCE BIT EXTENDED	RRBE	RRE	C	EK	P	A^1									10-37
B22B	SET STORAGE KEY EXTENDED	SSKE	RRE		EK	P	A^1				¢					10-45
B22C	TEST BLOCK	TB	RRE	C	TB	P	A^1		II		$	G0		R		10-50
B6	STORE CONTROL	STCTL	RS			P	A	SP							ST	10-48
B7	LOAD CONTROL	LCTL	RS			P	A	SP								10-20
BA	COMPARE AND SWAP	CS	RS	C	SW		A	SP			$			R	ST	7-12
BB	COMPARE DOUBLE AND SWAP	CDS	RS	C	SW		A	SP			$			R	ST	7-12
BD	COMPARE LOGICAL C. UNDER MASK	CLM	RS	C			A									7-15
BE	STORE CHARACTERS UNDER MASK	STCM	RS				A								ST	7-35
BF	INSERT CHARACTERS UNDER MASK	ICM	RS	C			A							R		7-20
D1	MOVE NUMERICS	MVN	SS				A								ST	7-27
D2	MOVE (character)	MVC	SS				A								ST	7-23
D3	MOVE ZONES	MVZ	SS				A								ST	7-28
D4	AND (character)	NC	SS	C			A								ST	7-8
D5	COMPARE LOGICAL (character)	CLC	SS	C			A									7-14
D6	OR (character)	OC	SS	C			A								ST	7-29
D7	EXCLUSIVE OR (character)	XC	SS	C			A								ST	7-18
D9	MOVE WITH KEY	MVCK	SS	C	DU	Q	A								ST	10-24
DA	MOVE TO PRIMARY	MVCP	SS	C	DU	Q	A		SO		¢				ST	10-22
DB	MOVE TO SECONDARY	MVCS	SS	C	DU	Q	A		SO		¢				ST	10-22
DC	TRANSLATE	TR	SS				A								ST	7-39
DD	TRANSLATE AND TEST	TRT	SS	C			A					GM		R		7-40
DE	EDIT	ED	SS	C			A		D						ST	8-6
DF	EDIT AND MARK	EDMK	SS	C			A		D			G1		R	ST	8-9
E500	LOAD ADDRESS SPACE PARAMETERS	LASP	SSE	C	DU	P	AS	SP	SO							10-12
E501	TEST PROTECTION	TPROT	SSE	C	EF	P	A^1									10-52
E8	MOVE INVERSE	MVCIN	SS		MI		A								ST	7-24
F0	SHIFT AND ROUND DECIMAL	SRP	SS	C			A		D	DF					ST	8-10
F1	MOVE WITH OFFSET	MVO	SS				A								ST	7-27
F2	PACK	PACK	SS				A								ST	7-30
F3	UNPACK	UNPK	SS				A								ST	7-40
F8	ZERO AND ADD	ZAP	SS	C			A		D	DF					ST	8-12
F9	COMPARE DECIMAL	CP	SS	C			A		D							8-5
FA	ADD DECIMAL	AP	SS	C			A		D	DF					ST	8-5
FB	SUBTRACT DECIMAL	SP	SS	C			A		D	DF					ST	8-11
FC	MULTIPLY DECIMAL	MP	SS				A	SP	D						ST	8-10
FD	DIVIDE DECIMAL	DP	SS				A	SP	D		DK				ST	8-5

Instructions Arranged by Operation Code (Part 4 of 4)

Name	Mne- monic	Characteristics				Op Code	Page No.
BRANCH AND SAVE	BASR	RR BS			B R	0D	7-9
BRANCH AND SAVE	BAS	RX BS			B R	4D	7-9

Instructions Arranged by Facility: Branch and Save

Name	Mne- monic	Characteristics				Op Code	Page No.
CONNECT CHANNEL SET	CONCS	S C CS	P	$		B200	10-4
DISCONNECT CHANNEL SET	DISCS	S C CS	P	$		B201	10-6

Instructions Arranged by Facility: Channel-Set Switching

Name	Mne- monic	Characteristics				Op Code	Page No.
ADD	AR	RR C		IF	R	1A	7-7
ADD	A	RX C	A	IF	R	5A	7-7
ADD DECIMAL	AP	SS C	A	D DF	ST	FA	8-5
ADD HALFWORD	AH	RX C	A	IF	R	4A	7-7
ADD LOGICAL	ALR	RR C			R	1E	7-8
ADD LOGICAL	AL	RX C	A		R	5E	7-8
AND	NR	RR C			R	14	7-8
AND	N	RX C	A		R	54	7-8
AND (character)	NC	SS C	A		ST	D4	7-8
AND (immediate)	NI	SI C	A		ST	94	7-8
BRANCH AND LINK	BALR	RR			B R	05	7-9
BRANCH AND LINK	BAL	RX			B R	45	7-9
BRANCH ON CONDITION	BCR	RR		¢[1]	B	07	7-10
BRANCH ON CONDITION	BC	RX			B	47	7-10
BRANCH ON COUNT	BCTR	RR			B R	06	7-11
BRANCH ON COUNT	BCT	RX			B R	46	7-11
BRANCH ON INDEX HIGH	BXH	RS			B R	86	7-11
BRANCH ON INDEX LOW OR EQUAL	BXLE	RS			B R	87	7-11
CLEAR I/O	CLRIO	S C	P	¢		9D01*	13-17
COMPARE	CR	RR C				19	7-12
COMPARE	C	RX C	A			59	7-12
COMPARE DECIMAL	CP	SS C	A	D		F9	8-5
COMPARE HALFWORD	CH	RX C	A			49	7-14
COMPARE LOGICAL	CLR	RR C				15	7-14
COMPARE LOGICAL	CL	RX C	A			55	7-14
COMPARE LOGICAL (character)	CLC	SS C	A			D5	7-14
COMPARE LOGICAL (immediate)	CLI	SI C	A			95	7-14
COMPARE LOGICAL C. UNDER MASK	CLM	RS C	A			BD	7-15
COMPARE LOGICAL LONG	CLCL	RR C	A SP	II	R	0F	7-15
CONVERT TO BINARY	CVB	RX	A	D IK	R	4F	7-16
CONVERT TO DECIMAL	CVD	RX	A		ST	4E	7-17
DIAGNOSE		DM	P DM			83	10-5
DIVIDE	DR	RR	SP	IK	R	1D	7-17
DIVIDE	D	RX	A SP	IK	R	5D	7-17
DIVIDE DECIMAL	DP	SS	A SP	D DK	ST	FD	8-5

Instructions Arranged by Facility: Commercial Instruction Set (Part 1 of 3)

Name	Mnemonic	Characteristics				Op Code	Page No.
EDIT	ED	SS C	A	D	ST	DE	8-6
EDIT AND MARK	EDMK	SS C	A	D G1	R ST	DF	8-9
EXCLUSIVE OR	XR	RR C			R	17	7-18
EXCLUSIVE OR	X	RX C	A		R	57	7-18
EXCLUSIVE OR (character)	XC	SS C	A		ST	D7	7-18
EXCLUSIVE OR (immediate)	XI	SI C	A		ST	97	7-18
EXECUTE	EX	RX	AI SP	EX		44	7-19
HALT DEVICE	HDV	S C	P	¢		9E01*	13-19
HALT I/O	HIO	S C	P	¢		9E00*	13-23
INSERT CHARACTER	IC	RX	A		R	43	7-20
INSERT CHARACTERS UNDER MASK	ICM	RS C	A		R	BF	7-20
INSERT STORAGE KEY	ISK	RR	P A^1 SP	SO	R	09	10-8
LOAD	LR	RR			R	18	7-20
LOAD	L	RX	A		R	58	7-20
LOAD ADDRESS	LA	RX			R	41	7-21
LOAD AND TEST	LTR	RR C			R	12	7-21
LOAD COMPLEMENT	LCR	RR C		IF	R	13	7-21
LOAD CONTROL	LCTL	RS	P A SP			B7	10-20
LOAD HALFWORD	LH	RX	A		R	48	7-22
LOAD MULTIPLE	LM	RS	A		R	98	7-22
LOAD NEGATIVE	LNR	RR C			R	11	7-22
LOAD POSITIVE	LPR	RR C		IF	R	10	7-22
LOAD PSW	LPSW	S L	P A SP	¢		82	10-20
MONITOR CALL	MC	SI	SP	MO		AF	7-23
MOVE (character)	MVC	SS	A		ST	D2	7-23
MOVE (immediate)	MVI	SI	A		ST	92	7-23
MOVE LONG	MVCL	RR C	A SP	II	R ST	0E	7-24
MOVE NUMERICS	MVN	SS	A		ST	D1	7-27
MOVE WITH OFFSET	MVO	SS	A		ST	F1	7-27
MOVE ZONES	MVZ	SS	A		ST	D3	7-28
MULTIPLY	MR	RR	SP		R	1C	7-28
MULTIPLY	M	RX	A SP		R	5C	7-28
MULTIPLY DECIMAL	MP	SS	A SP	D	ST	FC	8-10
MULTIPLY HALFWORD	MH	RX	A		R	4C	7-29
OR	OR	RR C			R	16	7-29
OR	O	RX C	A		R	56	7-29
OR (character)	OC	SS C	A		ST	D6	7-29
OR (immediate)	OI	SI C	A		ST	96	7-29
PACK	PACK	SS	A		ST	F2	7-30
SET CLOCK	SCK	S C	P A SP			B204	10-39
SET PROGRAM MASK	SPM	RR L				04	7-31
SET STORAGE KEY	SSK	RR	P A^1 SP	SO ¢		08	10-45
SET SYSTEM MASK	SSM	S	P A SP	SO		80	10-46
SHIFT AND ROUND DECIMAL	SRP	SS C	A	D DF	ST	F0	8-10
SHIFT LEFT DOUBLE	SLDA	RS C	SP	IF	R	8F	7-31
SHIFT LEFT DOUBLE LOGICAL	SLDL	RS	SP		R	8D	7-32
SHIFT LEFT SINGLE	SLA	RS C		IF	R	8B	7-32
SHIFT LEFT SINGLE LOGICAL	SLL	RS			R	89	7-33
SHIFT RIGHT DOUBLE	SRDA	RS C	SP		R	8E	7-33
SHIFT RIGHT DOUBLE LOGICAL	SRDL	RS	SP		R	8C	7-33
SHIFT RIGHT SINGLE	SRA	RS C			R	8A	7-34
SHIFT RIGHT SINGLE LOGICAL	SRL	RS			R	88	7-34
START I/O	SIO	S C	P	¢		9C00*	13-27
START I/O FAST RELEASE	SIOF	S C	P	¢		9C01*	13-27
STORE	ST	RX	A		ST	50	7-34

Instructions Arranged by Facility: Commercial Instruction Set (Part 2 of 3)

Name	Mne- monic	Characteristics				Op Code	Page No.
STORE CHANNEL ID	STIDC	S C	P	¢		B203	13-32
STORE CHARACTER	STC	RX	A		ST	42	7-34
STORE CHARACTERS UNDER MASK	STCM	RS	A		ST	BE	7-35
STORE CLOCK	STCK	S C	A	$	ST	B205	7-35
STORE CONTROL	STCTL	RS	P A SP		ST	B6	10-48
STORE CPU ID	STIDP	S	P A SP		ST	B202	10-48
STORE HALFWORD	STH	RX	A		ST	40	7-36
STORE MULTIPLE	STM	RS	A		ST	90	7-36
SUBTRACT	SR	RR C		IF	R	1B	7-36
SUBTRACT	S	RX C	A	IF	R	5B	7-36
SUBTRACT DECIMAL	SP	SS C	A	D DF	ST	FB	8-11
SUBTRACT HALFWORD	SH	RX C	A	IF	R	4B	7-37
SUBTRACT LOGICAL	SLR	RR C			R	1F	7-37
SUBTRACT LOGICAL	SL	RX C	A		R	5F	7-37
SUPERVISOR CALL	SVC	RR		¢		0A	7-38
TEST AND SET	TS	S C	A	$	ST	93	7-38
TEST CHANNEL	TCH	S C	P	¢		9F00*	13-33
TEST I/O	TIO	S C	P	¢		9D00*	13-34
TEST UNDER MASK	TM	SI C	A			91	7-38
TRANSLATE	TR	SS	A		ST	DC	7-39
TRANSLATE AND TEST	TRT	SS C	A	GM	R	DD	7-40
UNPACK	UNPK	SS	A		ST	F3	7-40
ZERO AND ADD	ZAP	SS C	A	D DF	ST	F8	8-12

Instructions Arranged by Facility: Commercial Instruction Set (Part 3 of 3)

Name	Mne- monic	Characteristics				Op Code	Page No.
COMPARE AND SWAP	CS	RS C SW	A SP	$	R ST	BA	7-12
COMPARE DOUBLE AND SWAP	CDS	RS C SW	A SP	$	R ST	BB	7-12

Instructions Arranged by Facility: Conditional Swapping

Name	Mne- monic	Characteristics				Op Code	Page No.
SET CLOCK COMPARATOR	SCKC	S CK	P A SP			B206	10-39
SET CPU TIMER	SPT	S CK	P A SP			B208	10-40
STORE CLOCK COMPARATOR	STCKC	S CK	P A SP		ST	B207	10-47
STORE CPU TIMER	STPT	S CK	P A SP		ST	B209	10-49

Instructions Arranged by Facility: CPU Timer and Clock Comparator

Name	Mnemonic	Characteristics				Op Code	Page No.
READ DIRECT	RDD	SI DC	P A[1]	$	SD	85	10-36
WRITE DIRECT	WRD	SI DC	P A[1]	$		84	10-54

Instructions Arranged by Facility: Direct Control

Name	Mnemonic	Characteristics				Op Code	Page No.
EXTRACT PRIMARY ASN	EPAR	RRE DU	Q	SO	R	B226	10-6
EXTRACT SECONDARY ASN	ESAR	RRE DU	Q	SO	R	B227	10-7
INSERT ADDRESS SPACE CONTROL	IAC	RRE C DU	Q	SO	R	B224	10-7
INSERT VIRTUAL STORAGE KEY	IVSK	RRE DU	Q A[1]	SO	R	B223	10-10
LOAD ADDRESS SPACE PARAMETERS	LASP	SSE C DU	P AS SP	SO		E500	10-12
MOVE TO PRIMARY	MVCP	SS C DU	Q A	SO ¢	ST	DA	10-22
MOVE TO SECONDARY	MVCS	SS C DU	Q A	SO ¢	ST	DB	10-22
MOVE WITH KEY	MVCK	SS C DU	Q A		ST	D9	10-24
PROGRAM CALL	PC	S DU	Q AT	Z[1] T ¢ GM	B R ST	B218	10-25
PROGRAM TRANSFER	PT	RRE DU	Q AT SP	Z[2] T ¢	B ST	B228	10-31
SET ADDRESS SPACE CONTROL	SAC	S DU	SP	SO ¢		B219	10-38
SET SECONDARY ASN	SSAR	RRE DU	AT	Z[3] T ¢	ST	B225	10-41

Instructions Arranged by Facility: Dual Address Space

Name	Mnemonic	Characteristics				Op Code	Page No.
INVALIDATE PAGE TABLE ENTRY	IPTE	RRE EF	P A[1]	$		B221	10-11
TEST PROTECTION	TPROT	SSE C EF	P A[1]			E501	10-52

Instructions Arranged by Facility: Extended Facility (without MVS Assist)

Name	Mnemonic	Characteristics				Op Code	Page No.
ADD NORMALIZED (extended)	AXR	RR C XP	SP	EU EO LS		36	9-6
LOAD ROUNDED (ext. to long)	LRDR	RR XP	SP	EO		25	9-12
LOAD ROUNDED (long to short)	LRER	RR XP	SP	EO		35	9-12
MULTIPLY (extended)	MXR	RR XP	SP	EU EO		26	9-13
MULTIPLY (long to extended)	MXDR	RR XP	SP	EU EO		27	9-13
MULTIPLY (long to extended)	MXD	RX XP	A SP	EU EO		67	9-13
SUBTRACT NORMALIZED (ext.)	SXR	RR C XP	SP	EU EO LS		37	9-14

Instructions Arranged by Facility: Extended-Precision Floating Point

Name	Mne- monic	Characteristics										Op Code	Page No.
ADD NORMALIZED (long)	ADR	RR	C	FP		SP	EU	EO		LS		2A	9-6
ADD NORMALIZED (long)	AD	RX	C	FP	A	SP	EU	EO		LS		6A	9-6
ADD NORMALIZED (short)	AER	RR	C	FP		SP	EU	EO		LS		3A	9-6
ADD NORMALIZED (short)	AE	RX	C	FP	A	SP	EU	EO		LS		7A	9-6
ADD UNNORMALIZED (long)	AWR	RR	C	FP		SP		EO		LS		2E	9-7
ADD UNNORMALIZED (long)	AW	RX	C	FP	A	SP		EO		LS		6E	9-7
ADD UNNORMALIZED (short)	AUR	RR	C	FP		SP		EO		LS		3E	9-7
ADD UNNORMALIZED (short)	AU	RX	C	FP	A	SP		EO		LS		7E	9-7
COMPARE (long)	CDR	RR	C	FP		SP						29	9-8
COMPARE (long)	CD	RX	C	FP	A	SP						69	9-8
COMPARE (short)	CER	RR	C	FP		SP						39	9-8
COMPARE (short)	CE	RX	C	FP	A	SP						79	9-8
DIVIDE (long)	DDR	RR		FP		SP	EU	EO	FK			2D	9-9
DIVIDE (long)	DD	RX		FP	A	SP	EU	EO	FK			6D	9-9
DIVIDE (short)	DER	RR		FP		SP	EU	EO	FK			3D	9-9
DIVIDE (short)	DE	RX		FP	A	SP	EU	EO	FK			7D	9-9
HALVE (long)	HDR	RR		FP		SP	EU					24	9-10
HALVE (short)	HER	RR		FP		SP	EU					34	9-10
LOAD (long)	LDR	RR		FP		SP						28	9-10
LOAD (long)	LD	RX		FP	A	SP						68	9-10
LOAD (short)	LER	RR		FP		SP						38	9-10
LOAD (short)	LE	RX		FP	A	SP						78	9-10
LOAD AND TEST (long)	LTDR	RR	C	FP		SP						22	9-11
LOAD AND TEST (short)	LTER	RR	C	FP		SP						32	9-11
LOAD COMPLEMENT (long)	LCDR	RR	C	FP		SP						23	9-11
LOAD COMPLEMENT (short)	LCER	RR	C	FP		SP						33	9-11
LOAD NEGATIVE (long)	LNDR	RR	C	FP		SP						21	9-11
LOAD NEGATIVE (short)	LNER	RR	C	FP		SP						31	9-11
LOAD POSITIVE (long)	LPDR	RR	C	FP		SP						20	9-12
LOAD POSITIVE (short)	LPER	RR	C	FP		SP						30	9-12
MULTIPLY (long)	MDR	RR		FP		SP	EU	EO				2C	9-13
MULTIPLY (long)	MD	RX		FP	A	SP	EU	EO				6C	9-13
MULTIPLY (short to long)	MER	RR		FP		SP	EU	EO				3C	9-13
MULTIPLY (short to long)	ME	RX		FP	A	SP	EU	EO				7C	9-13
STORE (long)	STD	RX		FP	A	SP					ST	60	9-14
STORE (short)	STE	RX		FP	A	SP					ST	70	9-14
SUBTRACT NORMALIZED (long)	SDR	RR	C	FP		SP	EU	EO		LS		2B	9-14
SUBTRACT NORMALIZED (long)	SD	RX	C	FP	A	SP	EU	EO		LS		6B	9-14
SUBTRACT NORMALIZED (short)	SER	RR	C	FP		SP	EU	EO		LS		3B	9-14
SUBTRACT NORMALIZED (short)	SE	RX	C	FP	A	SP	EU	EO		LS		7B	9-14
SUBTRACT UNNORMALIZED (long)	SWR	RR	C	FP		SP		EO		LS		2F	9-15
SUBTRACT UNNORMALIZED (long)	SW	RX	C	FP	A	SP		EO		LS		6F	9-15
SUBTRACT UNNORMALIZED (short)	SUR	RR	C	FP		SP		EO		LS		3F	9-15
SUBTRACT UNNORMALIZED (short)	SU	RX	C	FP	A	SP		EO		LS		7F	9-15

Instructions Arranged by Facility: Floating Point

Name	Mne-monic	Characteristics				Op Code	Page No.
MOVE INVERSE	MVCIN	SS MI	A		ST	E8	7-24

Instructions Arranged by Facility: Move Inverse

Name	Mne-monic	Characteristics				Op Code	Page No.
SET PREFIX	SPX	S MP	P A SP	$		B210	10-40
SIGNAL PROCESSOR	SIGP	RS C MP	P	$	R	AE	10-46
STORE CPU ADDRESS	STAP	S MP	P A SP		ST	B212	10-48
STORE PREFIX	STPX	S MP	P A SP		ST	B211	10-49

Instructions Arranged by Facility: Multiprocessing

Name	Mne-monic	Characteristics				Op Code	Page No.
INSERT PSW KEY	IPK	S PK	Q	G2	R	B20B	10-8
SET PSW KEY FROM ADDRESS	SPKA	S PK	Q			B20A	10-41

Instructions Arranged by Facility: PSW-Key Handling

Name	Mne-monic	Characteristics				Op Code	Page No.
CLEAR CHANNEL	CLRCH	S C RE	P	¢		9F01*	13-16

Instructions Arranged by Facility: Recovery Extensions

Name	Mne-monic	Characteristics				Op Code	Page No.
INSERT STORAGE KEY EXTENDED	ISKE	RRE EK	P A[1]		R	B229	10-9
RESET REFERENCE BIT EXTENDED	RRBE	RRE C EK	P A[1]			B22A	10-37
SET STORAGE KEY EXTENDED	SSKE	RRE EK	P A[1]	¢		B22B	10-45

Instructions Arranged by Facility: Storage-Key-Instruction Extensions

Name	Mne-monic	Characteristics				Op Code	Page No.
RESUME I/O	RIO	S C SR	P	¢		9C02*	13-26

Instructions Arranged by Facility: Suspend and Resume

Name	Mnemonic	Characteristics				Op Code	Page No.
TEST BLOCK	TB	RRE C TB	P A[1]	II $ GO	R	B22C	10-50

Instructions Arranged by Facility: Test Block

Name	Mnemonic	Characteristics				Op Code	Page No.
LOAD REAL ADDRESS	LRA	RX C TR	P A[1]		R	B1	10-21
PURGE TLB	PTLB	S TR	P	$		B20D	10-36
RESET REFERENCE BIT	RRB	S C TR	P A[1]	SO		B213	10-36
STORE THEN AND SYSTEM MASK	STNSM	SI TR	P A		ST	AC	10-50
STORE THEN OR SYSTEM MASK	STOSM	SI TR	P A SP		ST	AD	10-50

Instructions Arranged by Facility: Translation

APPENDIX C. CONDITION-CODE SETTINGS

This appendix lists the condition-code setting for instructions in the System/370 architecture which set the condition code. In addition to those instructions listed which set the condition code, the condition code may be changed by DIAGNOSE and the target of EXECUTE. The condition code is loaded by LOAD PSW, by SET PROGRAM MASK, and by an interruption. The condition code is set to zero by initial CPU reset and is loaded by the successful conclusion of the initial-program-loading sequence.

The condition codes for the vector facility are not included in this appendix. See the publication IBM System/370 Vector Operations, SA22-7125, for the condition codes set by vector instructions.

Some models may offer instructions which set the condition code and do not appear in this document, such as those provided for assists or as part of special or custom features.

	Condition Code			
Instruction	0	1	2	3
ADD, ADD HALFWORD	Zero	< zero	> zero	Overflow
ADD DECIMAL	Zero	< zero	> zero	Overflow
ADD LOGICAL	Zero, no carry	Not zero, no carry	Zero, carry	Not zero, carry
ADD NORMALIZED	Zero	< zero	> zero	--
ADD UNNORMALIZED	Zero	< zero	> zero	--
AND	Zero	Not zero	--	--
CLEAR CHANNEL	Reset signaled	--	Channel busy	Not operational
CLEAR I/O	No operation in progress	CSW stored	Channel busy	Not operational
COMPARE (gen, fl pt)	Equal	Low	High	--
COMPARE HALFWORD	Equal	Low	High	--
COMPARE AND SWAP	Equal	Not equal	--	--
COMPARE DECIMAL	Equal	Low	High	--
COMPARE DOUBLE AND SWAP	Equal	Not equal	--	--
COMPARE LOGICAL	Equal	Low	High	--
COMPARE LOGICAL CHARACTERS UNDER MASK	Equal	Low	High	--
COMPARE LOGICAL LONG	Equal	Low	High	--
CONNECT CHANNEL SET	Successful	Connected to another CPU	--	Not operational
DISCONNECT CHANNEL SET	Successful	Connected to another CPU	--	Not operational
EDIT, EDIT AND MARK	Zero	< zero	> zero	--
EXCLUSIVE OR	Zero	Not zero	--	--
HALT DEVICE	Interruption pending/busy	CSW stored	Channel working	Not operational
HALT I/O	Interruption pending	CSW stored	Burst operation terminated	Not operational
INSERT ADDRESS SPACE CONTROL	Zero	One	--	--
INSERT CHARACTERS UNDER MASK	All zeros	First bit one	First bit zero	--
LOAD ADDRESS SPACE PARAMETERS	Parameters loaded	Primary ASN not available	Secondary ASN not available or not authorized	Space-switch event
LOAD AND TEST (gen, fl pt)	Zero	< zero	> zero	--
LOAD COMPLEMENT (gen)	Zero	< zero	> zero	Overflow
LOAD COMPLEMENT (fl pt)	Zero	< zero	> zero	--
LOAD NEGATIVE (gen, fl pt)	Zero	< zero	--	--
LOAD POSITIVE (gen)	Zero	--	> zero	Overflow
LOAD POSITIVE (fl pt)	Zero	--	> zero	--
LOAD REAL ADDRESS	Translation available	ST entry invalid	PT entry invalid	Length violation
MOVE LONG	Length equal	Length low	Length high	Destructive overlap
MOVE TO PRIMARY, MOVE TO SECONDARY	Length =< 256	--	--	Length > 256
MOVE WITH KEY	Length =< 256	--	--	Length > 256

Summary of Condition-Code Settings (Part 1 of 2)

Instruction	Condition Code			
	0	1	2	3
OR	Zero	Not zero	--	--
RESET REFERENCE BIT, RESET REFERENCE BIT EXTENDED	R bit zero, C bit zero	R bit zero, C bit one	R bit one, C bit zero	R bit one, C bit one
RESUME I/O	Successful	--	--	Not operational
SET CLOCK	Set	Secure	--	Not operational
SHIFT AND ROUND DECIMAL	Zero	< zero	> zero	Overflow
SHIFT LEFT (DOUBLE/SINGLE)	Zero	< zero	> zero	Overflow
SHIFT RIGHT (DOUBLE/SINGLE)	Zero	< zero	> zero	--
SIGNAL PROCESSOR	Order accepted	Status stored	Busy	Not operational
START I/O, START I/O FAST RELEASE	Successful	CSW stored	Busy	Not operational
STORE CHANNEL ID	ID stored	CSW stored	Busy	Not operational
STORE CLOCK	Set	Not set	Error	Not operational
SUBTRACT, SUBTRACT HALFWORD	Zero	< zero	> zero	Overflow
SUBTRACT DECIMAL	Zero	< zero	> zero	Overflow
SUBTRACT LOGICAL	--	Not zero, no carry	Zero, carry	Not zero, carry
SUBTRACT NORMALIZED	Zero	< zero	> zero	--
SUBTRACT UNNORMALIZED	Zero	< zero	> zero	--
TEST AND SET	Left bit zero	Left bit one	--	--
TEST BLOCK	Usable	Not usable	--	--
TEST CHANNEL	Available	Interruption pending	Burst mode	Not operational
TEST I/O	Available	CSW stored	Busy	Not operational
TEST PROTECTION	Can fetch, can store	Can fetch, cannot store	Cannot fetch, cannot store	Translation not available
TEST UNDER MASK	All zeros	Mixed	--	All ones
TRANSLATE AND TEST	All zeros	Incomplete	Complete	--
ZERO AND ADD	Zero	< zero	> zero	Overflow

Explanation:

--	Not applicable
> zero	Result greater than zero
< zero	Result less than zero
=< 256	Equal to, or less than, 256
> 256	Greater than 256
High	First operand high
Low	First operand low
Length	Length of first operand

Summary of Condition-Code Settings (Part 2 of 2)

APPENDIX D. FACILITIES

This appendix lists the facilities in System/370. Every system includes a CPU, main storage, and the capability for at least one byte-multiplexer, block-multiplexer, or selector channel. The capability may be implemented by means of a separate physical unit or may be provided by sharing the physical unit with the CPU.

COMMERCIAL INSTRUCTION SET

Every CPU incorporates the commercial instruction set (listed in Appendix B) and the associated basic computing functions, including:

- Byte-oriented operands
- General registers
- Basic-control (BC) mode
- 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)
- Key-controlled protection
- Interval timer
- TOD clock
- Basic operator facilities

OTHER FACILITIES

Additionally, the following facilities are available:

BRANCH AND SAVE

Includes the BRANCH AND SAVE (BAS and BASR) instruction.

CHANNEL INDIRECT DATA ADDRESSING

Includes indirect-data-address words and the associated CCW flag, which facilitate storage addressing when virtual addresses are used.

CHANNEL-SET SWITCHING

Provides the ability to connect a channel set to any CPU in a multiprocessing configuration. It includes the instructions CONNECT CHANNEL SET and DISCONNECT CHANNEL SET.

CLEAR I/O

Provides the clear-I/O (CLRIO) function on a channel when the CLEAR I/O instruction is executed. When the CLRIO function is not implemented, CLEAR I/O is executed as TEST I/O.

COMMAND RETRY

Provides the capability in a channel to retry a command without the occurrence of an I/O interruption. The retry is initiated by the control unit.

CONDITIONAL SWAPPING

Includes the instructions COMPARE AND SWAP and COMPARE DOUBLE AND SWAP.

CPU TIMER AND CLOCK COMPARATOR

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 the instructions SET CLOCK COMPARATOR, STORE CLOCK COMPARATOR, SET CPU TIMER, and STORE CPU TIMER.

DIRECT CONTROL

Includes the external-signal facility and the read-write-direct facility, which contains the instructions READ DIRECT and WRITE DIRECT.

DUAL-ADDRESS SPACE (DAS)

Includes the following:

1. Dual-space control, which includes:

   a. An address-space control, PSW bit 16

   b. A primary ASN, bits 16-31 of control register 4

   c. A secondary ASN, bits 16-31 of control register 3

   d. A secondary-segment-table designation, in control register 7

2. DAS authorization mechanisms, which include the following:

   a. An extraction-authority control, bit 4 of control register 0

   b. A PSW-key mask, bits 0-15 of control register 3

   c. A secondary-space control, bit 5 of control register 0

   d. A subsystem-linkage control, bit 0 of control register 5

   e. An ASN-translation control, bit 12 of control register 14

   f. An authorization index, bits 0-15 of control register 4

   g. A space-switch-event-control bit, bit 31 of control register 1

3. PC-number translation, which uses the linkage-table designation in control register 5

4. ASN translation, which uses an ASN-first-table origin, bits 20-31 of control register 14

5. ASN authorization

6. DAS tracing

7. The following instructions:

   EXTRACT PRIMARY ASN (EPAR)
   EXTRACT SECONDARY ASN (ESAR)
   INSERT ADDRESS SPACE CONTROL (IAC)
   INSERT VIRTUAL STORAGE KEY (IVSK)

LOAD ADDRESS SPACE PARAMETERS (LASP)
MOVE TO PRIMARY (MVCP)
MOVE TO SECONDARY (MVCS)
MOVE WITH KEY (MVKC)
PROGRAM CALL (PC)
PROGRAM TRANSFER (PT)
SET ADDRESS SPACE CONTROL (SAC)
SET SECONDARY ASN (SSAR)

8. Nine new exception or event conditions which result in a program interruption. These conditions are:

   AFX-translation exception
   ASN-translation-specification exception
   ASX-translation exception
   EX-translation exception
   LX-translation exception
   PC-translation-specification exception
   Primary-authority exception
   Secondary-authority exception
   Space-switch event

   For page- and segment-translation exceptions, a bit is stored with the translation-exception address. This bit indicates whether the address was translated by using the primary or secondary segment-table designation.

The following System/370 instructions are changed or affected by the installation of DAS, as noted:

- Execution of the SET PSW KEY FROM ADDRESS instruction is permitted in the problem state, subject to the contents of bit positions 0-15 of control register 3. When the bit in the control register corresponding to the PSW-key value to be set is one, execution is allowed; otherwise, a privileged-operation exception is recognized. The contents of control register 3 are ignored in the supervisor state.

- Execution of the INSERT PSW KEY instruction is permitted in the problem state, subject to the extraction-authority control, bit 4 of control register 0. When the bit is one, execution is allowed; otherwise, a privileged-operation exception is recognized. The extraction-authority control is ignored in the supervisor state.

- LOAD REAL ADDRESS uses the contents of control register 7, instead of the contents of control register 1, when PSW bit 16 is one. Thus the second operand is translated either as a primary virtual address or as a secondary virtual address, depending on the mode specified in the PSW.

- The second-operand address of EXECUTE is defined to be an instruction address rather than a logical address. In secondary-space mode, it is thus unpredictable whether the target instruction is fetched from the primary space or the secondary space.

EXTENDED

Includes the instructions INVALIDATE PAGE TABLE ENTRY and TEST PROTECTION, the common-segment facility and the associated bit position in the segment-table entry, low-address protection and the associated control-register position for the low-address-protection control bit, and 12 MVS-dependent instructions. INVALIDATE PAGE TABLE ENTRY includes revisions to the READ DIRECT and WRITE DIRECT instructions to make the operand addresses real instead of logical.

EXTENDED-PRECISION FLOATING POINT

Includes the extended-precision floating-point instructions (listed in Appendix B).

EXTENDED REAL ADDRESSING

Provides for a 26-bit page-frame real address in the page-table entry for 4K-byte pages.

EXTERNAL SIGNALS

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.

FAST RELEASE

Provides the start-I/O-fast-release (SIOF) function on the channel when the START I/O FAST RELEASE instruction is executed. This function provides for fast release of the CPU, which occurs before the device-selection procedure is completed, reducing the CPU delay associated with the initiation of the I/O operation. When the SIOF function is not implemented, START I/O FAST RELEASE is executed as START I/O.

FLOATING POINT

Includes the floating-point instructions (listed in Appendix B) and the floating-point registers. The floating-point facility, together with the commercial instruction set, is sometimes referred to as the universal instruction set.

HALT DEVICE

Provides the halt-device (HDV) function on a channel when the HALT DEVICE instruction is executed. When the HDV function is not implemented, HALT DEVICE is executed as HALT I/O.

I/O EXTENDED LOGOUT

Provides for the storing of detailed channel-error information in a storage area designated by a pointer.

LIMITED CHANNEL LOGOUT

Provides four bytes of channel-status information for model-independent recovery from channel errors.

MOVE INVERSE

Includes the MOVE INVERSE instruction.

MULTIPROCESSING

Includes the following facilities, which permit the formation of a multiprocessing configuration:

- <u>Shared Main Storage</u>
- <u>Prefixing</u>
- <u>CPU-Address Identification</u>
- <u>CPU Signaling and Response</u>
- <u>TOD-Clock Synchronization</u>

These facilities include four extensions to the 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.

PSW-KEY HANDLING

Includes the instructions SET PSW KEY FROM ADDRESS and INSERT PSW KEY.

RECOVERY EXTENSIONS

Includes the following:

- Machine-check external-damage code at real locations 244-247, the external-damage-code-validity bit (bit 26 of the machine-check-interruption code), and the channel-not-operational indication in the machine-check external-damage code.
- The clear-channel (CLRCH) function in a channel when the CLEAR CHANNEL instruction is executed; when the CLRCH function is not implemented, CLEAR CHANNEL is executed as TEST CHANNEL.
- The full-channel-logout-valid bit (bit 15) and the interface-inoperative bit (bit 27) in the limited channel logout.

SEGMENT PROTECTION

Provides a segment-protection bit in the segment-table entry. When the bit is one, an attempt to store in the segment causes a protection exception to be recognized.

SERVICE SIGNAL

Provides an external interruption which is used by the service-call logical processor (SCLP) to signal to the control program.

START-I/O-FAST QUEUING

Provides for fast release of the CPU by the channel during the execution of START I/O FAST RELEASE and the queuing of the operation at the subchannel when the control unit or device is busy, rather than termination of the operation by means of an I/O interruption with a deferred-condition-code-1 indication. The queuing of the operation at the subchannel appears to the program as if no busy indication had been encountered. Includes the ability to store a nonzero value in the measurement byte at location 185.

STORAGE-KEY-INSTRUCTION EXTENSIONS

Provides the instructions INSERT STORAGE KEY EXTENDED, RESET REFERENCE BIT EXTENDED, and SET STORAGE KEY EXTENDED. These instructions provide 31-bit addresses and operate on the storage keys associated with a 4K-byte block of storage.

STORAGE-KEY 4K-BYTE BLOCK

Provides for a single key associated with each 4K-byte block of storage, and the storage-key-exception control, bit 7 of control register 0. When this facility is not installed, a separate storage key is associated with each 2K-byte block of storage.

SUSPEND AND RESUME

Provides a suspend bit in the CCW which may indicate that the channel program is to be suspended, as well as a bit in the CAW that controls whether the suspend bit should be examined and a new bit in the channel-status word which indicates that a channel program has been suspended. The instruction RESUME I/O causes a suspended channel program to be resumed.

TEST BLOCK

Includes the TEST BLOCK instruction for testing the usability of a 4K-byte block of main storage.

TRANSLATION

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. The DAT facility also includes controls for 4K-byte page size and 64K-byte segment size. Depending on the model, controls for 2K-byte page size or 1M-byte segment size, or both, may also be provided.

- _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 Noninitializing Manual 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.

VECTOR

The instructions and functions of the vector facility and its registers are described in the publication _IBM System/370 Vector Operations_, SA22-7125.

31-BIT IDAWS

Extends the size of the address field in the indirect-data-address word to 31 bits.

APPENDIX E. TABLE OF POWERS OF 2

PLUS		MINUS
1	0	1.
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.01562 5
128	7	0.00781 25
256	8	0.00390 625
512	9	0.00195 3125
1,024	10	0.00097 65625
2,048	11	0.00048 82812 5
4,096	12	0.00024 41406 25
8,192	13	0.00012 20703 125
16,384	14	0.00006 10351 5625
32,768	15	0.00003 05175 78125
65,536	16	0.00001 52587 89062 5
131,072	17	0.00000 76293 94531 25
262,144	18	0.00000 38146 97265 625
524,288	19	0.00000 19073 48632 8125
1,048,576	20	0.00000 09536 74316 40625
2,097,152	21	0.00000 04768 37158 20312 5
4,194,304	22	0.00000 02384 18579 10156 25
8,388,608	23	0.00000 01192 09289 55078 125
16,777,216	24	0.00000 00596 04644 77539 0625
33,554,432	25	0.00000 00298 02322 38769 53125
67,108,864	26	0.00000 00149 01161 19384 76562 5
134,217,728	27	0.00000 00074 50580 59692 38281 25
268,435,456	28	0.00000 00037 25290 29846 19140 625
536,870,912	29	0.00000 00018 62645 14923 09570 3125
1,073,741,824	30	0.00000 00009 31322 57461 54785 15625
2,147,483,648	31	0.00000 00004 65661 28730 77392 57812 5
4,294,967,296	32	0.00000 00002 32830 64365 38696 28906 25
8,589,934,592	33	0.00000 00001 16415 32182 69348 14453 125
17,179,869,184	34	0.00000 00000 58207 66091 34674 07226 5625
34,359,738,368	35	0.00000 00000 29103 83045 67337 03613 28125
68,719,476,736	36	0.00000 00000 14551 91522 83668 51806 64062 5
137,438,953,472	37	0.00000 00000 07275 95761 41834 25903 32031 25
274,877,906,944	38	0.00000 00000 03637 97880 70917 12951 66015 625
549,755,813,888	39	0.00000 00000 01818 98940 35458 56475 83007 8125
1,099,511,627,776	40	0.00000 00000 00909 49470 17729 28237 91503 90625
2,199,023,255,552	41	0.00000 00000 00454 74735 08864 64118 95751 95312 5
4,398,046,511,104	42	0.00000 00000 00227 37367 54432 32059 47875 97656 25
8,796,093,022,208	43	0.00000 00000 00113 68683 77216 16029 73937 98828 125
17,592,186,044,416	44	0.00000 00000 00056 84341 88608 08014 86968 99414 0625
35,184,372,088,832	45	0.00000 00000 00028 42170 94304 04007 43484 49707 03125
70,368,744,177,664	46	0.00000 00000 00014 21085 47152 02003 71742 24853 51562 5
140,737,488,355,328	47	0.00000 00000 00007 10542 73576 01001 85871 12426 75781 25
281,474,976,710,656	48	0.00000 00000 00003 55271 36788 00500 92935 56213 37890 625
562,949,953,421,312	49	0.00000 00000 00001 77635 68394 00250 46467 78106 68945 3125
1,125,899,906,842,624	50	0.00000 00000 00000 88817 84197 00125 23233 89053 34472 65625
2,251,799,813,685,248	51	0.00000 00000 00000 44408 92098 50062 61616 94526 67236 32812 5
4,503,599,627,370,496	52	0.00000 00000 00000 22204 46049 25031 30808 47263 33618 16406 25
9,007,199,254,740,992	53	0.00000 00000 00000 11102 23024 62515 65404 23631 66809 08203 125
18,014,398,509,481,984	54	0.00000 00000 00000 05551 11512 31257 82702 11815 83404 54101 5625
36,028,797,018,963,968	55	0.00000 00000 00000 02775 55756 15628 91351 05907 91702 27050 78125
72,057,594,037,927,936	56	0.00000 00000 00000 01387 77878 07814 45675 52953 95851 13525 39062 5
144,115,188,075,855,872	57	0.00000 00000 00000 00693 88939 03907 22837 76476 97925 56762 69531 25
288,230,376,151,711,744	58	0.00000 00000 00000 00346 94469 51953 61418 88238 48962 78381 34765 625
576,460,752,303,423,488	59	0.00000 00000 00000 00173 47234 75976 80709 44119 24481 39190 67382 8125
1,152,921,504,606,846,976	60	0.00000 00000 00000 00086 73617 37988 40354 72059 62240 69595 33691 40625
2,305,843,009,213,693,952	61	0.00000 00000 00000 00043 36808 68994 20177 36029 81120 34797 66845 70312 5
4,611,686,018,427,387,904	62	0.00000 00000 00000 00021 68404 34497 10088 68014 90560 17398 83422 85156 25
9,223,372,036,854,775,808	63	0.00000 00000 00000 00010 84202 17248 55044 34007 45280 08699 41711 42578 125
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 F. 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: $D34_{16} = 3380_{10}$

```
D =   13
     x16
     208
3 =  + 3
     211
     x16
    3376
4 =   +4
    3380
```

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

```
16 |3380                remainder
16 |211  ─────────►  4      ▲
16 |13   ─────────►  3      │
         ─────────►  D      │      3380₁₀=D34₁₆
```

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 subtract
   .1300
   -.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: $.8A7 = .540771_{10}$

$8A7_{16} = 2215_{10}$

$16^3 = 4096$

$$\begin{array}{r} .540771 \\ 4096\overline{)2215.000000} \end{array}$$

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

$$\begin{array}{rl} & .5408 \\ & \underline{\times 16} \\ 8 \leftarrow & \boxed{8}.6528 \\ & \underline{\times 16} \\ A \leftarrow & \boxed{10}.4448 \\ & \underline{\times 16} \\ 7 \leftarrow & \boxed{7}.1168 \end{array}$$

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

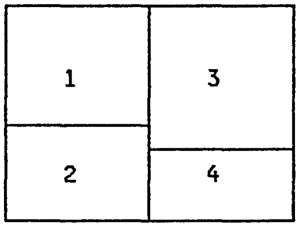
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	06	0C	12	18	1E	24	2A	30	36	3C	42	48	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 G. EBCDIC CHART

EXTENDED BINARY-CODED-DECIMAL INTERCHANGE CODE (EBCDIC)

The 256-position EBCDIC table shows graphic-character, control-character, and formatting-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 pattern for most characters, partition the 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. Each such position is indicated by a circled number in the upper right corner of the box for that position. The card-hole patterns for these positions are shown beneath the table. Bit-position numbers, bit patterns, and hexadecimal representations for these positions are found in the usual manner.

The EBCDIC table shows 94 graphic-character positions. Some products have used an 88-character, 63-character, or 62-character subset of these graphic characters.

The 94-character set consists of all graphic characters shown in the EBCDIC table. This character set can be used for interchange with other systems; those systems may use codes, other than EBCDIC, which have 94 graphic characters.

An 88-character set that has been used consists of the 94-character set with the graphic characters at 6A, 79, A1, C0, D0, and E0 hex omitted. This character set has been used for 44-key keyboard applications which require both uppercase and lowercase alphabetic characters.

A 63-character set that has been used consists of the 94-character set with the lowercase alphabetic characters omitted and with the graphic characters at 6A, 79, A1, C0, and D0 hex omitted. This character set has been used for interchange with other systems; those systems may have used codes, other than EBCDIC, which have 63 graphic characters.

A 62-character set that has been used consists of the 63-character set with the graphic character at E0 hex omitted. This character set has been used for 44-key keyboard applications which do not require lowercase alphabetic characters.

Thirteen positions (4A, 4F, 5A, 5B, 5F, 6A, 79, 7B, 7C, A1, C0, D0, and E0 hex) are defined in the table as Data Processing National Use positions. Each such position contains a shaded triangle in the top left corner of the box for that position. The graphic characters provided in these positions on printing and display devices may differ from one language to another or from one country to another. The characters provided for use in data-processing applications by the English (U.S.) version of EBCDIC are shown in the table.

The other graphic characters shown in the EBCDIC table are provided for data-processing applications in the English (U.S.) version of EBCDIC and in additional versions of EBCDIC in other languages which use a Latin-based alphabet. Products designed for data-processing applications in a language which does not use a Latin-based alphabet support character sets meeting the particular requirements of that language.

Word-processing products normally support a character set slightly different from the one shown in the table. Additionally, a number of application areas (such as printing and publishing, magnetic-ink character recognition, and some programming languages) also require unique character-set support.

Some examples of the use of the EBCDIC table are shown in the following figure:

Character	Type	Bit Pattern	Hex	Hole Pattern	
				Zone Punches	Digit Punches
SEL	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

Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	Digit Punches	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									Digit Punches	
0000	0	8-1	NUL ①	DLE ②	DS ③	④	SP ⑤	& ⑥	- ⑦	⑧					{ ⑨	} ⑩	\ ⑪	0 ⑫	8-1	
0001	1	1	SOH	DC1	SOS		RSP		/ ⑬		a	j	~		A	J	⑭ NSP	1	1	
0010	2	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2	2	
0011	3	3	ETX	DC3	WUS	IR					c	l	t		C	L	T	3	3	
0100	4	4	SEL	RES/ ENP	BYP/ INP	PP					d	m	u		D	M	U	4	4	
0101	5	5	HT	NL	LF	TRN					e	n	v		E	N	V	5	5	
0110	6	6	RNL	BS	ETB	NBS					f	o	w		F	O	W	6	6	
0111	7	7	DEL	POC	ESC	EOT					g	p	x		G	P	X	7	7	
1000	8	8	GE	CAN	SA	SBS					h	q	y		H	Q	Y	8	8	
1001	9	8-1	SPS	EM	SFE	IT				`	i	r	z		I	R	Z	9	9	
1010	A	8-2	RPT	UBS	SM/SW	RFF	¢	!	¦ ⑮	:					SHY				8-2	
1011	B	8-3	VT	CU1	CSP	CU3	.	$	,	#									8-3	
1100	C	8-4	FF	IFS	MFA	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/ ITB	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	④	12-11-0-9-8-1	⑦	11	⑩	11-0	⑬	0-1
②	12-11-9-8-1	⑤	No Punches	⑧	12-11-0	⑪	0-8-2	⑭	11-0-9-1
③	11-0-9-8-1	⑥	12	⑨	12-0	⑫	0	⑮	12-11

Formatting Character Representations

NSP	Numeric Space
RSP	Required Space
SP	Space
SHY	Syllable Hyphen

Control Character Representations

ACK	Acknowledge	E1X	End of Text	RFF	Required Form Feed
BEL	Bell	FF	Form Feed	RNL	Required New Line
BS	Backspace	FS	Field Separator	RPT	Repeat
BYP/INP	Bypass/Inhibit Presentation	GE	Graphic Escape	SA	Set Attribute
CAN	Cancel	HT	Horizontal Tab	SBS	Subscript
CR	Carriage Return	IFS	Interchange File Separator	SEL	Select
CSP	Control Sequence Prefix	IGS	Interchange Group Separator	SFE	Start Field Extended
CU1	Customer Use 1	IR	Index Return	SI	Shift In
CU3	Customer Use 3	IRS	Interchange Record Separator	SM/SW	Set Mode/Switch
DC1	Device Control 1	IT	Indent Tab	SO	Shift Out
DC2	Device Control 2	IUS/ITB	Interchange Unit Separator/ Intermediate Transmission Block	SOH	Start of Heading
DC3	Device Control 3			SOS	Start of Significance
DC4	Device Control 4	LF	Line Feed	SPS	Superscript
DEL	Delete	MFA	Modify Field Attribute	STX	Start of Text
DLE	Data Link Escape	NAK	Negative Acknowledge	SUB	Substitute
DS	Digit Select	NBS	Numeric Backspace	SYN	Synchronous Idle
EM	End of Medium	NL	New Line	TRN	Transparent
ENQ	Enquiry	NUL	Null	UBS	Unit Backspace
EO	Eight Ones	POC	Program-Operator Communication	VT	Vertical Tab
EOT	End of Transmission			WUS	Word Underscore
ESC	Escape	PP	Presentation Position		
ETB	End of Transmission Block	RES/ENP	Restore/Enable Presentation		

Special Graphic Characters

¢	Cent Sign	,	Comma
.	Period, Decimal Point	%	Percent
<	Less-than Sign	_	Underscore
(	Left Parenthesis	>	Greater-than Sign
+	Plus Sign	?	Question Mark
|	Logical OR	`	Grave Accent
&	Ampersand	:	Colon
!	Exclamation Point	#	Number Sign
$	Dollar Sign	@	At Sign
*	Asterisk	'	Prime, Apostrophe
)	Right Parenthesis	=	Equal Sign
;	Semicolon	"	Quotation Mark
¬	Logical NOT	~	Tilde
-	Minus Sign, Hyphen	{	Opening Brace
/	Slash	}	Closing Brace
¦	Vertical Line	\	Reverse Slant

APPENDIX H. CHANGES AFFECTING COMPATIBILITY BETWEEN SYSTEM/360 AND SYSTEM/370

This appendix summarizes those changes included in the System/370 architecture that may affect whether or not a program written according to the System/360 architecture will operate on models implementing the System/370 architecture described in this publication. Not included are descriptions of System/370 functions which are compatible extensions, that is, (1) those that are suppressed on initialization, such as block multiplexing, and (2) those that are specified in such a manner that they cause program exceptions on System/360, such as new instructions.

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 preferred codes for USASCII-8 are generated for decimal results. When PSW bit 12 is zero, the preferred codes 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 preferred codes for EBCDIC.

Bit 12 of the PSW is handled in System/370 as follows:

- In models that do not have the translation facility installed, a one in PSW bit position 12 causes a program interruption for specification exception.
- In models that have the translation facility installed, a one in PSW bit position 12 causes the CPU to operate in the extended-control (EC) mode.

OPERATION CODES OF I/O INSTRUCTIONS

In System/360, the operation codes of the four I/O instructions (HALT I/O, START I/O, TEST CHANNEL, and TEST I/O) are one byte in length, and bits 8-15 of the I/O instructions are ignored. In System/370, the operation codes of all I/O instructions are the first two bytes of the instruction. System/360 programs that execute I/O instructions in which any of bits 8-15 is not zero may perform a different function when executed on a System/370 CPU, as explained below.

Halt I/O

In System/370, HALT I/O (HIO) is assigned the operation code 9E00 hex and HALT DEVICE (HDV) the operation code 9E01. Because bits 8-14 are ignored in both instructions, an instruction executed as HALT I/O in System/360 will still be executed as HALT I/O in System/370 if the third hex digit is any value and the fourth hex digit is an <u>even</u> value. However, in System/370, if bit 15 of the instruction is one, the function performed will be the HIO function or the HDV function, depending on the design of the channel.

Start I/O

In System/370, START I/O is assigned the operation code 9C00 and RESUME I/O is assigned the operation code 9C02. Therefore, an instruction executed as START I/O in System/360 will be executed as RESUME I/O in System/370 if bits 8-15 of the instruction contain the value 02 hex and the suspend-and-resume facility is installed. When the suspend-and-resume facility is installed, operation codes in the range 9C03 through 9CFF cause an operation exception to be recognized. If the suspend-and-resume facility is not installed, bits 8-14 of the instruction are always ignored, and bit 15 is ignored when the block-multiplexing-control bit (bit 0 of control register 0) is zero at the time the instruction is executed.

Test Channel

In System/370, TEST CHANNEL is assigned the operation code 9F00 and CLEAR CHANNEL (CLRCH) the operation code 9F01 hex. Because bits 8-14 of the instruction are ignored in both instructions, an instruction executed as TEST CHANNEL in System/360 will still be executed as TEST CHANNEL in System/370 if the third hex digit is any value and the fourth hex digit is an even value. However, in System/370, if bit 15 of the instruction is one, the CLRCH function is performed if the recovery-extension facility is installed; otherwise, the TCH function is performed.

LOGOUT

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, 244-255, 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) area, 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.

In System/360, logout is not permitted on data check. System/370 permits logout to occur when the channel causes an I/O interruption with the data-check indication.

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 effects of command retry usually are not significant; however, the following is a list of some of the differences which command retry can cause:

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.

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.

VALIDITY OF DATA

In System/360, the contents of main storage are preserved when power is turned off. In System/370, because main storage may be volatile or nonvolatile, the program must not depend on the validity of data in main storage after system power has been lost or turned off and then restored.

APPENDIX I. CHANGES AFFECTING COMPATIBILITY WITHIN SYSTEM/370

This appendix summarizes those changes included in the System/370 architecture that may affect whether or not a program written according to the original System/370 architecture will operate on models implementing the architecture described in this publication. Not included here are descriptions of compatible extensions, such as new facilities incorporated in System/370 that make use of unassigned operation codes and formats.

READ DIRECT AND WRITE DIRECT

When the instruction INVALIDATE PAGE TABLE ENTRY is installed, the following changes apply:

- Both READ DIRECT and WRITE DIRECT are changed to use real instead of logical addresses.

- Program-event recording does not apply to the storage alteration performed by READ DIRECT.

STORE ACCESSES

The following changes are made as to when an access to storage for storing can take place.

- When the execution of the instruction is nullified or suppressed because of certain program exceptions, an interlocked-update reference may occur at the operand location. Originally no storage access was permitted. In some of these situations, the channel may observe intermediate results which differ from the final result. See the section "Exceptions to Nullification and Suppression" in Chapter 5, "Program Execution."

- When the mask in STORE CHARACTERS UNDER MASK is zero, an interlocked-update reference may occur at the byte location designated by the operand address. Originally no storage access was permitted.

- When the result of comparison in COMPARE AND SWAP or COMPARE DOUBLE AND SWAP is unequal, an interlocked-update reference may occur at the operand location. Originally no storage access was permitted.

- When the result of the store operation is defined to be unpredictable, such as for STORE CLOCK with the clock in the error state, the store access may be omitted.

Whether or not a store access takes place is visible to the program in four ways: an access exception may be indicated, the change bit may be set, a PER storage-alteration event may be indicated, and, for stores that are part of an interlocked-update reference, the channel may observe the distinct accesses for fetching and storing. The fetch and store parts of an interlocked-update reference appear interlocked to other CPUs.

FETCH ACCESSES

Originally the definition required that, with the exception of some compare instructions, access exceptions on fetching be indicated for the unused portion of an operand. The changed definition permits the indication of the access exception for the unused parts to be unpredictable, except that an access exception still must be indicated for TEST UNDER MASK, INSERT CHARACTERS UNDER MASK, and COMPARE LOGICAL CHARACTERS UNDER MASK when the mask is zero.

OPERAND-ACCESS CONSISTENCY

Originally the access for the operand of LOAD MULTIPLE was specified to be doubleword-concurrent; that is, all bytes within a doubleword appear to all CPUs to be accessed concurrently. This definition is changed to require doubleword concurrency only if the operand is designated on a word boundary.

The restriction is removed that, during the padding portion of a MOVE LONG execution, another CPU can observe the operand to be stored only once and only in the left-to-right sequence.

CHANGE BIT

Originally the System/370 architecture specified that the change bit be set to one each time information is stored in the corresponding storage block. This definition is changed as follows:

- The change bit now is necessarily set to one only when the contents of the corresponding storage block are changed. In situations where execution of the instruction can be completed without making a store access, such as in MOVE (MVC) with coincident operands or in OR (OI) with an immediate operand of zeros, the change bit may be unaffected. However, even when the change bit is not set, any applicable access exceptions or PER storage-alteration events are still indicated.

- The change bit may be set to one as a result of those situations described in the section "Store Accesses" in this appendix.

- Because of CPU retry, the change bit may be set to one for locations which the program has not accessed.

SUBCHANNEL INTERRUPTION-PENDING STATE

Originally only status associated with the termination of an I/O operation at the subchannel could cause the subchannel to enter the interruption-pending state. Status not associated with the termination of an I/O operation at the subchannel was held pending at the device, and the subchannel would be available. The changed definition allows status not associated with the termination of an I/O operation at the subchannel to be accepted into the subchannel. As a result of this change, a subchannel that is shared among multiple devices may cause condition code 2 to be returned to a START I/O, START I/O FAST RELEASE, or TEST I/O even if no previous START I/O or START I/O FAST RELEASE had been executed specifying the same device. This busy state persists until the interruption condition is cleared.

START I/O AND START I/O FAST RELEASE

Originally the System/370 architecture specified START I/O and START I/O FAST RELEASE as having the operation codes 9C00 and 9C01, respectively, with bits 8-14 of the operation code ignored by the CPU. Now, however, when the suspend-and-resume facility is installed, bits 8-14 of the operation code for START I/O and START I/O FAST RELEASE are no longer ignored by the CPU.

Operation codes 9CX0, 9CX2, 9CX4, 9CX6, 9CX8, 9CXA, 9CXC, and 9CXE (with X representing any hex digit) all were executed as START I/O. Similarly, operation codes 9CX1, 9CX3, 9CX5, 9CX7, 9CX9, 9CXB, 9CXD, and 9CXF all were executed as START I/O FAST RELEASE. When the suspend-and-resume facility is installed, only operation code 9C00 is executed as START I/O, and only operation code 9C01 is executed as START I/O FAST RELEASE; operation code 9C02 is executed as RESUME I/O, and all operation codes in the range 9C03 through 9CFF cause an operation exception to be recognized.

IBM System/370
Principles of Operation

Order No. GA22-7000-10

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ________________________________

Number of latest Newsletter associated with this publication: ________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the front cover or title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure-sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

International Business Machines Corporation
Department E57
P.O. Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

IBM System/370
Principles of Operation

Order No. GA22-7000-10

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ____________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the front cover or title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure-sensitive or other gummed tape to seal this form.

GA22-7000-10

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department E57
P.O. Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

IBM®

IBM System/370 Principles of Operation (File No. S370-01) Printed in USA GA22-7000-10

Publication Number
GA22-7000-10

File Number
S370-01

Printed in
USA

IBM

GA22-7000-10